

Eventos

Hasta ahora, todas las aplicaciones y scripts que se han creado tienen algo en común: se ejecutan desde la primera instrucción hasta la última de forma secuencial. Gracias a las estructuras de control de flujo (`if`, `for`, `while`) es posible modificar ligeramente este comportamiento y repetir algunos trozos del script y saltarse otros trozos en función de algunas condiciones.

Este tipo de aplicaciones son poco útiles, ya que no interactúan con los usuarios y no pueden responder a los diferentes *eventos* que se producen durante la ejecución de una aplicación. Afortunadamente, las aplicaciones web creadas con el lenguaje JavaScript pueden utilizar el modelo de *programación basada en eventos*.

En este tipo de programación, los scripts se dedican a esperar a que el usuario "*haga algo*" (que pulse una tecla, que mueva el ratón, que cierre la ventana del navegador). A continuación, el script responde a la acción del usuario normalmente procesando esa información y generando un resultado.

Los eventos hacen posible que los usuarios transmitan información a los programas. JavaScript define numerosos eventos que permiten una interacción completa entre el usuario y las páginas/aplicaciones web. La pulsación de una tecla constituye un evento, así como pinchar o mover el ratón, seleccionar un elemento de un formulario, redimensionar la ventana del navegador, etc.

JavaScript permite asignar una función a cada uno de los eventos. De esta forma, cuando se produce cualquier evento, JavaScript ejecuta su función asociada. Este tipo de funciones se denominan "*event handlers*" en inglés y suelen traducirse por "*manejadores de eventos*".

1. Modelos de eventos

Crear páginas y aplicaciones web siempre ha sido mucho más complejo de lo que debería serlo debido a las incompatibilidades entre navegadores. A pesar de que existen decenas de estándares para las tecnologías empleadas, los navegadores no los soportan completamente o incluso los ignoran.

Las principales incompatibilidades se producen en el lenguaje XHTML, en el soporte de hojas de estilos CSS y sobre todo, en la implementación de JavaScript. De todas ellas, la incompatibilidad más importante se da precisamente en el modelo de eventos del navegador. Así, existen hasta tres modelos diferentes para manejar los eventos dependiendo del navegador en el que se ejecute la aplicación.

1.1. Modelo básico de eventos

Este modelo simple de eventos se introdujo para la versión 4 del estándar HTML y se considera parte del nivel más básico de DOM. Aunque sus características son limitadas, es el único modelo que es compatible en todos los navegadores y por tanto, el único que permite crear aplicaciones que funcionan de la misma manera en todos los navegadores.

1.2. Modelo de eventos estándar

Las versiones más avanzadas del estándar DOM (DOM nivel 2) definen un modelo de eventos completamente nuevo y mucho más poderoso que el original. Todos los navegadores modernos lo incluyen, salvo Internet Explorer.

1.3. Modelo de eventos de Internet Explorer

Internet Explorer utiliza su propio modelo de eventos, que es similar pero incompatible con el modelo estándar. Se utilizó por primera vez en Internet Explorer 4 y Microsoft decidió seguir utilizándolo en el resto de versiones, a pesar de que la empresa había participado en la creación del estándar de DOM que define el modelo de eventos estándar.

Modelo básico de eventos

1. Tipos de eventos

En este modelo, cada elemento o etiqueta XHTML define 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 XHTML diferentes y un mismo elemento XHTML puede tener asociados varios eventos diferentes.

El nombre de cada evento se construye mediante el prefijo `on`, seguido del nombre en inglés de la acción asociada al evento. Así, el evento de pinchar un elemento con el ratón se denomina `onclick` y el evento asociado a la acción de mover el ratón se denomina `onmousemove`.

La siguiente tabla resume los eventos más importantes definidos por JavaScript:

Evento	Descripción	Elementos para los que está definido
<code>onblur</code>	Deseleccionar el elemento	<code><button></code> , <code><input></code> , <code><label></code> , <code><select></code> , <code><textarea></code> , <code><body></code>
<code>onchange</code>	Deseleccionar un elemento que se ha modificado	<code><input></code> , <code><select></code> , <code><textarea></code>
<code>onclick</code>	Pinchar y soltar el ratón	Todos los elementos
<code>ondblclick</code>	Pinchar dos veces seguidas con el ratón	Todos los elementos
<code>onfocus</code>	Seleccionar un elemento	<code><button></code> , <code><input></code> , <code><label></code> , <code><select></code> , <code><textarea></code> , <code><body></code>
<code>onkeydown</code>	Pulsar una tecla (sin soltar)	Elementos de formulario y <code><body></code>

Evento	Descripción	Elementos para los que está definido
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
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	<body>

Evento	Descripción	Elementos para los que está definido
	ventana del navegador	
onselect	Seleccionar un texto	<input>, <textarea>
onsubmit	Enviar el formulario	<form>
onunload	Se abandona la página (por ejemplo al cerrar el navegador)	<body>

Los eventos más utilizados en las aplicaciones web tradicionales son `onload` para esperar a que se cargue la página por completo, los eventos `onclick`, `onmouseover`, `onmouseout` para controlar el ratón y `onsubmit` para controlar el envío de los formularios.

Algunos eventos de la tabla anterior (`onclick`, `onkeydown`, `onkeypress`, `onreset`, `onsubmit`) permiten evitar la "acción por defecto" de ese evento. Más adelante se muestra en detalle este comportamiento, que puede resultar muy útil en algunas técnicas de programación.

Las acciones típicas que realiza un usuario en una página web pueden dar lugar a una sucesión de eventos. Al pulsar por ejemplo sobre un botón de tipo `<input type="submit">` se desencadenan los eventos `onmousedown`, `onclick`, `onmouseup` y `onsubmit` de forma consecutiva.

2. Manejadores de eventos

Un evento de JavaScript por sí mismo carece de utilidad. Para que los eventos resulten útiles, se deben asociar funciones o código JavaScript a cada evento. De esta forma, cuando se produce un evento se ejecuta el código indicado, por lo que la aplicación puede *responder* ante cualquier evento que se produzca durante su ejecución.

Las funciones o código JavaScript que se definen para cada evento se denominan "*manejador de eventos*" y como JavaScript es un lenguaje muy flexible, existen varias formas diferentes de indicar los manejadores:

- Manejadores como atributos de los elementos XHTML.
- Manejadores como funciones JavaScript externas.
- Manejadores "semánticos".

2.1. Manejadores de eventos como atributos 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. En el siguiente ejemplo, se quiere mostrar un mensaje cuando el usuario pinche con el ratón sobre un botón:

```
<input type="button" value="Pinchame y verás" onclick="alert('Gracias por pinchar');" />
```

En este método, se definen atributos XHTML con el mismo nombre que los eventos que se quieren manejar. El ejemplo anterior sólo quiere controlar el evento de pinchar con el ratón, cuyo nombre es `onclick`. Así, el elemento XHTML para el que se quiere definir este evento, debe incluir un atributo llamado `onclick`.

El contenido del atributo es una cadena de texto que contiene todas las instrucciones JavaScript que se ejecutan cuando se produce el evento. En este caso, el código JavaScript es muy sencillo (`alert('Gracias por pinchar');`), ya que solamente se trata de mostrar un mensaje.

En este otro ejemplo, cuando el usuario pincha sobre el elemento `<div>` se muestra un mensaje y cuando el usuario pasa el ratón por encima del elemento, se muestra otro mensaje:

```
<div onclick="alert('Has pinchado con el ratón');" onmouseover="alert('Abas de pasar el ratón por encima');">
```

Puedes pinchar sobre este elemento o simplemente pasar el ratón por encima

```
</div>
```

Este otro ejemplo incluye una de las instrucciones más utilizadas en las aplicaciones JavaScript más antiguas:

```
<body onload="alert('La página se ha cargado completamente');">
```

...

</body>

El mensaje anterior se muestra después de que la página se haya cargado completamente, es decir, después de que se haya descargado su código HTML, sus imágenes y cualquier otro objeto incluido en la página.

El evento `onload` es uno de los más utilizados ya que, como se vio en el capítulo de DOM, las funciones que permiten acceder y manipular los nodos del árbol DOM solamente están disponibles cuando la página se ha cargado completamente.

2.2. Manejadores de eventos y variable `this`

JavaScript define una variable especial llamada `this` que se crea automáticamente y que se emplea en algunas técnicas avanzadas de programación. En los eventos, se puede utilizar la variable `this` para referirse al elemento XHTML que ha provocado el evento. Esta variable es muy útil para ejemplos como el siguiente:

Cuando el usuario pasa el ratón por encima del `<div>`, el color del borde se muestra de color negro. Cuando el ratón *sale* del `<div>`, se vuelve a mostrar el borde con el color gris claro original.

Elemento `<div>` original:

```
<div id="contenidos" style="width:150px; height:60px; border:thin solid silver">
```

Sección de contenidos...

```
</div>
```

Si no se utiliza la variable `this`, el código necesario para modificar el color de los bordes, sería el siguiente:

```
<div id="contenidos" style="width:150px; height:60px; border:thin solid silver" onmouseover="document.getElementById('contenidos').style.borderColor='black';" onmouseout="document.getElementById('contenidos').style.borderColor='silver';">
```

Sección de contenidos...

</div>

El código anterior es demasiado largo y demasiado propenso a cometer errores. Dentro del código de un evento, JavaScript crea automáticamente la variable `this`, que hace referencia al elemento XHTML que ha provocado el evento. Así, el ejemplo anterior se puede reescribir de la siguiente manera:

```
<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>

El código anterior es mucho más compacto, más fácil de leer y de escribir y sigue funcionando correctamente aunque se modifique el valor del atributo `id` del `<div>`.

2.3. Manejadores de eventos como funciones externas

La definición de los manejadores de eventos en los atributos XHTML es el método más sencillo pero menos aconsejable de tratar con los eventos en JavaScript. El principal inconveniente es que se complica en exceso en cuanto se añaden algunas pocas instrucciones, por lo que solamente es recomendable para los casos más sencillos.

Si se realizan aplicaciones complejas, como por ejemplo la validación de un formulario, es aconsejable agrupar todo el código JavaScript en una función externa y llamar a esta función desde el elemento XHTML.

Siguiendo con el ejemplo anterior que muestra un mensaje al pinchar sobre un botón:

```
<input type="button" value="Pinchame y verás" onclick="alert('Gracias por pinchar');" />
```

Utilizando funciones externas se puede transformar en:

```
function muestraMensaje() {
```



```
    alert('Gracias por pinchar');  
  
}
```

```
<input type="button" value="Pinchame y verás" onclick="muestraMensaje()" />
```

Esta técnica consiste en extraer todas las instrucciones de JavaScript y agruparlas en una función externa. Una vez definida la función, en el atributo del elemento XHTML se incluye el nombre de la función, para indicar que es la función que se ejecuta cuando se produce el evento.

La llamada a la función se realiza de la forma habitual, indicando su nombre seguido de los paréntesis y de forma opcional, incluyendo todos los argumentos y parámetros que se necesiten.

El principal inconveniente de este método es que en las funciones externas no se puede seguir utilizando la variable `this` y por tanto, es necesario pasar esta variable como parámetro a la función:

```
function resalta(elemento) {  
  
    switch(elemento.style.borderColor) {  
  
        case 'silver':  
  
            case 'silver silver silver silver':  
  
            case '#c0c0c0':  
  
                elemento.style.borderColor = 'black';  
  
                break;  
  
            case 'black':  
  
            case 'black black black black':
```

```

    case '#000000':

        elemento.style.borderColor = 'silver';

        break;

    }

}

<div style="width:150px; height:60px; border:thin solid silver" onmouseover="resalta(this)" onmouseout="resalta(this)">

    Sección de contenidos...

</div>

```

En el ejemplo anterior, la función externa es llamada con el parámetro `this`, que dentro de la función se denomina `elemento`. La complejidad del ejemplo se produce sobre todo por la forma en la que los distintos navegadores almacenan el valor de la propiedad `borderColor`.

Mientras que Firefox almacena (en caso de que los cuatro bordes coincidan en color) el valor `black`, Internet Explorer lo almacena como `black black black black` y Opera almacena su representación hexadecimal `#000000`.

2.2.4. Manejadores de eventos semánticos

Los métodos que se han visto para añadir manejadores de eventos (como atributos XHTML y como funciones externas) tienen un grave inconveniente: *"ensucian"* el código XHTML de la página.

Como es conocido, una de las buenas prácticas básicas en el diseño de páginas y aplicaciones web es la separación de los contenidos (XHTML) y su aspecto o presentación (CSS). Siempre que sea posible, también se recomienda separar los contenidos (XHTML) y su comportamiento o programación (JavaScript).

Mezclar el código JavaScript con los elementos XHTML solamente contribuye a complicar el código fuente de la página, a dificultar la modificación y

mantenimiento de la página y a reducir la semántica del documento final producido.

Afortunadamente, existe un método alternativo para definir los manejadores de eventos de JavaScript. Esta técnica es una evolución del método de las funciones externas, ya que se basa en utilizar las propiedades DOM de los elementos XHTML para asignar todas las funciones externas que actúan de manejadores de eventos. Así, el siguiente ejemplo:

```
<input id="pinchable" type="button" value="Pinchame y verás" onclick="alert('Gracias por pinchar');" />
```

Se puede transformar en:

```
// Función externa
```

```
function muestraMensaje() {  
  
    alert('Gracias por pinchar');  
  
}
```

```
// Asignar la función externa al elemento
```

```
document.getElementById("pinchable").onclick = muestraMensaje;
```

```
// Elemento XHTML
```

```
<input id="pinchable" type="button" value="Pinchame y verás" />
```

La técnica de los manejadores semánticos consiste en:

1. Asignar un identificador único al elemento XHTML mediante el atributo `id`.
2. Crear una función de JavaScript encargada de manejar el evento.
3. Asignar la función externa al evento correspondiente en el elemento deseado.

El último paso es la clave de esta técnica. En primer lugar, se obtiene el elemento al que se desea asociar la función externa:

```
document.getElementById("pinchable");
```

A continuación, se utiliza una propiedad del elemento con el mismo nombre que el evento que se quiere manejar. En este caso, la propiedad es `onclick`:

```
document.getElementById("pinchable").onclick = ...
```

Por último, se asigna la función externa mediante su nombre **sin paréntesis**. Lo más importante (y la causa más común de errores) es indicar solamente el nombre de la función, es decir, prescindir de los paréntesis al asignar la función:

```
document.getElementById("pinchable").onclick = muestraMensaje;
```

Si se añaden los paréntesis después del nombre de la función, en realidad se está ejecutando la función y guardando el valor devuelto por la función en la propiedad `onclick` de elemento.

```
// Asignar una función externa a un evento de un elemento
```

```
document.getElementById("pinchable").onclick = muestraMensaje;
```

```
// Ejecutar una función y guardar su resultado en una propiedad de un elemento
```

```
document.getElementById("pinchable").onclick = muestraMensaje();
```

La gran ventaja de este método es que el código XHTML resultante es muy "*limpio*", ya que no se mezcla con el código JavaScript. Además, dentro de las funciones externas asignadas sí que se puede utilizar la variable `this` para referirse al elemento que provoca el evento.

El único inconveniente de este método es que la página se debe cargar completamente antes de que se puedan utilizar las funciones DOM que asignan los manejadores a los elementos XHTML. Una de las formas más sencillas de asegurar que cierto código se va a ejecutar después de que la página se cargue por completo es utilizar el evento `onload`:

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

La técnica anterior utiliza el concepto de *funciones anónimas*, que no se va a estudiar, pero que permite crear un código compacto y muy sencillo. Para asegurarse que un código JavaScript va a ejecutarse después de que la página se haya cargado completamente, sólo es necesario incluir esas instrucciones entre los símbolos { y }:

```
window.onload = function() {  
  
    ...  
  
}
```

En el siguiente ejemplo, se añaden eventos a los elementos de tipo `input=text` de un formulario complejo:

```
function resalta() {  
  
    // Código JavaScript  
  
}  
  
window.onload = function() {  
  
    var formulario = document.getElementById("formulario");  
  
    var camposInput = formulario.getElementsByTagName("input");  
  
    for(var i=0; i<camposInput.length; i++) {  
  
        if(camposInput[i].type == "text") {  
  
            camposInput[i].onclick = resalta;  
  
        }  
  
    }  
  
}
```

3. Obteniendo información del evento (objeto event)

Normalmente, los manejadores de eventos requieren información adicional para procesar sus tareas. Si una función por ejemplo se encarga de procesar el evento `onclick`, quizás necesite saber en qué posición estaba el ratón en el momento de pinchar el botón.

No obstante, el caso más habitual en el que es necesario conocer información adicional sobre el evento es el de los eventos asociados al teclado. Normalmente, es muy importante conocer la tecla que se ha pulsado, por ejemplo, para diferenciar las teclas normales de las teclas especiales (`ENTER`, `tabulador`, `Alt`, `Ctrl`, etc.).

JavaScript permite obtener información sobre el ratón y el teclado mediante un objeto especial llamado `event`. Desafortunadamente, los diferentes navegadores presentan diferencias muy notables en el tratamiento de la información sobre los eventos.

La principal diferencia reside en la forma en la que se obtiene el objeto `event`. Internet Explorer considera que este objeto forma parte del objeto `window` y el resto de navegadores lo consideran como el único argumento que tienen las funciones manejadoras de eventos.

Aunque es un comportamiento que resulta muy extraño al principio, todos los navegadores modernos excepto Internet Explorer crean *mágicamente* y de forma automática un argumento que se pasa a la función manejadora, por lo que no es necesario incluirlo en la llamada a la función manejadora. De esta forma, para utilizar este "*argumento mágico*", sólo es necesario asignarle un nombre, ya que los navegadores lo crean automáticamente.

En resumen, en los navegadores tipo Internet Explorer, el objeto `event` se obtiene directamente mediante:

```
var evento = window.event;
```

Por otra parte, en el resto de navegadores, el objeto `event` se obtiene *mágicamente* a partir del argumento que el navegador crea automáticamente:

```
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.

3.1. Información sobre el evento

La propiedad `type` indica el tipo de evento producido, lo que es útil cuando una misma función se utiliza para manejar varios eventos:

```
var tipo = evento.type;
```

La propiedad `type` devuelve el tipo de evento producido, que es igual al nombre del evento pero sin el prefijo `on`.

Mediante esta propiedad, se puede rehacer de forma más sencilla el ejemplo anterior en el que se resaltaba una sección de contenidos al pasar el ratón por encima:

```
function resalta(elEvento) {  
  
    var evento = elEvento || window.event;  
  
    switch(evento.type) {  
  
        case 'mouseover':
```

```
        this.style.borderColor = 'black';

        break;

    case 'mouseout':

        this.style.borderColor = 'silver';

        break;

    }

}

window.onload = function() {

    document.getElementById("seccion").onmouseover = resalta;

    document.getElementById("seccion").onmouseout = resalta;

}
```

```
<div id="seccion" style="width:150px; height:60px; border:thin solid silver">

    Sección de contenidos...

</div>
```


3.2. Información sobre los eventos de teclado

De todos los eventos disponibles en JavaScript, los eventos relacionados con el teclado son los más incompatibles entre diferentes navegadores y por tanto, los más difíciles de manejar. En primer lugar, existen muchas diferencias entre los navegadores, los teclados y los sistemas operativos de los usuarios, principalmente debido a las diferencias entre idiomas.

Además, existen tres eventos diferentes para las pulsaciones de las teclas (`onkeyup`, `onkeypress` y `onkeydown`). Por último, existen dos tipos de teclas: las teclas *normales* (como letras, números y símbolos normales) y las teclas *especiales* (como ENTER, Alt, Shift, etc.)

Cuando un usuario pulsa una tecla normal, se producen tres eventos seguidos y en este orden: `onkeydown`, `onkeypress` y `onkeyup`. El evento `onkeydown` se corresponde con el hecho de pulsar una tecla y no soltarla; el evento `onkeypress` es la propia pulsación de la tecla y el evento `onkeyup` hace referencia al hecho de soltar una tecla que estaba pulsada.

La forma más sencilla de obtener la información sobre la tecla que se ha pulsado es mediante el evento `onkeypress`. La información que proporcionan los eventos `onkeydown` y `onkeyup` se puede considerar como más técnica, ya que devuelven el código interno de cada tecla y no el carácter que se ha pulsado.

A continuación, se incluye una lista con todas las propiedades diferentes de todos los eventos de teclado tanto en Internet Explorer como en el resto de navegadores:

- Evento `keydown`:
 - Mismo comportamiento en todos los navegadores:
 - Propiedad `keyCode`: código interno de la tecla
 - Propiedad `charCode`: no definido
- Evento `keypress`:
 - Internet Explorer:
 - Propiedad `keyCode`: el código del carácter de la tecla que se ha pulsado
 - Propiedad `charCode`: no definido
 - Resto de navegadores:
 - Propiedad `keyCode`: para las teclas normales, no definido. Para las teclas especiales, el código interno de la tecla.

- Propiedad `charCode`: para las teclas normales, el código del carácter de la tecla que se ha pulsado. Para las teclas especiales, `0`.
- Evento `keyup`:
 - Mismo comportamiento en todos los navegadores:
 - Propiedad `keyCode`: código interno de la tecla
 - Propiedad `charCode`: no definido

Para convertir el código de un carácter (no confundir con el código interno) al carácter que representa la tecla que se ha pulsado, se utiliza la función `String.fromCharCode()`.

A continuación, se incluye un script que muestra toda la información sobre los tres eventos de teclado:

```
window.onload = function() {  
  
    document.onkeyup = muestraInformacion;  
  
    document.onkeypress = muestraInformacion;  
  
    document.onkeydown = muestraInformacion;  
  
}
```

```
function muestraInformacion(elEvento) {  
  
    var evento = window.event || elEvento;  
  
    var mensaje = "Tipo de evento: " + evento.type + "<br>" +  
  
        "Propiedad keyCode: " + evento.keyCode + "<br>" +  
  
        "Propiedad charCode: " + evento.charCode + "<br>" +
```

```
        "Carácter pulsado: " + String.fromCharCode(evento.charCod  
e);
```

```
        info.innerHTML += "<br>-----<br>" + me  
nsaje
```

```
    }
```

```
...
```

```
<div id="info"></div>
```

Al pulsar la tecla **a** en el navegador Firefox, se muestra la siguiente sucesión de eventos:

Tipo de evento: keydown

Propiedad keyCode: 65

Propiedad charCode: 0

Carácter pulsado: ?

Tipo de evento: keypress

Propiedad keyCode: 0

Propiedad charCode: 97

Carácter pulsado: a

Tipo de evento: keyup

Propiedad keyCode: 65

Propiedad charCode: 0

Carácter pulsado: ?

Al pulsar la tecla A (la misma tecla, pero habiendo activado previamente las mayúsculas) se muestra la siguiente sucesión de eventos en el navegador Firefox:

Tipo de evento: keydown

Propiedad keyCode: 65

Propiedad charCode: 0

Carácter pulsado: ?

Tipo de evento: keypress

Propiedad keyCode: 0

Propiedad charCode: 65

Carácter pulsado: A

Tipo de evento: keyup

Propiedad keyCode: 65

Propiedad charCode: 0

Carácter pulsado: ?

En los eventos `keydown` y `keyup`, la propiedad `keyCode` sigue valiendo lo mismo en los dos casos. El motivo es que `keyCode` almacena el código interno de la tecla, por lo que si se pulsa la misma tecla, se obtiene el mismo código, independientemente de que una misma tecla puede producir caracteres diferentes (por ejemplo mayúsculas y minúsculas).

En el evento `keypress`, el valor de la propiedad `charCode` varía, ya que el carácter `a`, no es el mismo que el carácter `A`. En este caso, el valor de `charCode` coincide con el código ASCII del carácter pulsado.

Siguiendo en el navegador Firefox, si ahora se pulsa una tecla *especial*, como por ejemplo el tabulador, se muestra la siguiente información:

Tipo de evento: keydown

Propiedad keyCode: 9

Propiedad charCode: 0

Carácter pulsado: ?

Tipo de evento: keypress

Propiedad keyCode: 9

Propiedad charCode: 0

Carácter pulsado: ?

Tipo de evento: keyup

Propiedad keyCode: 9

Propiedad charCode: 0

Carácter pulsado: ?

Las teclas especiales no disponen de la propiedad `charCode`, ya que solamente se guarda el código interno de la tecla pulsada en la propiedad `keyCode`, en este caso el código 9. Si se pulsa la tecla Enter, se obtiene el código 13, la tecla de la flecha superior produce el código 38, etc. No obstante, dependiendo del teclado utilizado para pulsar las teclas y dependiendo de la disposición de las teclas en función del idioma del teclado, estos códigos podrían variar.

A continuación, se muestra el resultado de la ejecución del mismo ejemplo anterior en el navegador Internet Explorer. Al pulsar la tecla a, se obtiene la siguiente información:

Tipo de evento: keydown

Propiedad keyCode: 65

Propiedad charCode: undefined

Carácter pulsado:

Tipo de evento: keypress

Propiedad keyCode: 97

Propiedad charCode: undefined

Carácter pulsado:

Tipo de evento: keyup

Propiedad keyCode: 65

Propiedad charCode: undefined

Carácter pulsado:

La propiedad `keyCode` en el evento `keypress` contiene el código ASCII del carácter de la tecla, por lo que se puede obtener directamente el carácter mediante `String.fromCharCode(keyCode)`.

Si se pulsa la tecla A, la información mostrada es idéntica a la anterior, salvo que el código que muestra el evento `keypress` cambia por 65, que es el código ASCII de la tecla A:

Tipo de evento: keydown

Propiedad keyCode: 65

Propiedad charCode: undefined

Carácter pulsado:

Tipo de evento: keypress

Propiedad keyCode: 65

Propiedad charCode: undefined

Carácter pulsado:

Tipo de evento: keyup

Propiedad keyCode: 65

Propiedad charCode: undefined

Carácter pulsado:

Al pulsar una tecla *especial* como el tabulador, Internet Explorer muestra la siguiente información:

Tipo de evento: keydown

Propiedad keyCode: 9

Propiedad charCode: undefined

Carácter pulsado:

Los códigos mostrados para las teclas especiales coinciden con los de Firefox y el resto de navegadores, pero recuerda que pueden variar en función del teclado que se utiliza y en función de la disposición de las teclas para cada idioma.

Por último, las propiedades `altKey`, `ctrlKey` y `shiftKey` almacenan un valor booleano que indica si alguna de esas teclas estaba pulsada al producirse el evento del teclado. Sorprendentemente, estas tres propiedades funcionan de la misma forma en todos los navegadores:

```
if(evento.altKey) {  
  
    alert('Estaba pulsada la tecla ALT');  
  
}
```

A continuación, se muestra el caso en el que se pulsa la tecla `Shift` y sin soltarla, se pulsa sobre la tecla que contiene el número 2 (en este caso, se refiere a la tecla que se encuentra en la parte superior del teclado y por tanto, no se refiere a la que se encuentra en el teclado numérico). Tanto Internet Explorer como Firefox muestran la misma secuencia de eventos:

Tipo de evento: keydown

Propiedad keyCode: 16

Propiedad charCode: 0

Carácter pulsado: ?

Tipo de evento: keydown

Propiedad keyCode: 50

Propiedad charCode: 0

Carácter pulsado: ?

Tipo de evento: keypress

Propiedad keyCode: 0

Propiedad charCode: 34

Carácter pulsado: "

Tipo de evento: keyup

Propiedad keyCode: 50

Propiedad charCode: 0

Carácter pulsado: ?

Tipo de evento: keyup

Propiedad keyCode: 16

Propiedad charCode: 0

Carácter pulsado: ?

El evento `keypress` es el único que permite obtener el *carácter realmente pulsado*, ya que al pulsar sobre la tecla 2 habiendo pulsado la tecla Shift previamente, se obtiene el carácter `"`, que es precisamente el que muestra el evento `keypress`.

El siguiente código de JavaScript permite obtener de forma correcta en cualquier navegador el carácter correspondiente a la tecla pulsada:

```
function manejador(elEvento) {  
  
    var evento = elEvento || window.event;  
  
    var caracter = evento.charCode || evento.keyCode;  
  
    alert("El carácter pulsado es: " + String.fromCharCode(caracter));  
  
}
```



```
document.onkeypress = manejador;
```

3.3. Información sobre los eventos de ratón

La información más relevante sobre los eventos relacionados con el ratón es la de las coordenadas de la posición del puntero del ratón. Aunque el origen de las coordenadas siempre se encuentra en la esquina superior izquierda, el punto que se toma como referencia de las coordenadas puede variar.

De esta forma, es posible obtener la posición del ratón respecto de la pantalla del ordenador, respecto de la ventana del navegador y respecto de la propia página HTML (que se utiliza cuando el usuario ha hecho *scroll* sobre la página). Las coordenadas más sencillas son las que se refieren a la posición del puntero respecto de la ventana del navegador, que se obtienen mediante las propiedades `clientX` y `clientY`:

```
function muestraInformacion(elEvento) {  
  
    var evento = elEvento || window.event;  
  
    var coordenadaX = evento.clientX;  
  
    var coordenadaY = evento.clientY;  
  
    alert("Has pulsado el ratón en la posición: " + coordenadaX + ", " + co  
ordenadaY);  
  
}  
  
document.onclick = muestraInformacion;
```

Las coordenadas de la posición del puntero del ratón respecto de la pantalla completa del ordenador del usuario se obtienen de la misma forma, mediante las propiedades `screenX` y `screenY`:

```
var coordenadaX = evento.screenX;  
  
var coordenadaY = evento.screenY;
```

En muchas ocasiones, es necesario obtener otro par de coordenadas diferentes: las que corresponden a la posición del ratón respecto del origen de la página. Estas coordenadas no siempre coinciden con las coordenadas respecto del origen de la ventana del navegador, ya que el usuario puede hacer *scroll* sobre la página web. Internet Explorer no proporciona estas coordenadas de forma directa, mientras que el resto de navegadores sí que lo hacen. De esta forma, es necesario detectar si el navegador es de tipo Internet Explorer y en caso afirmativo realizar un cálculo sencillo:

```
// Detectar si el navegador es Internet Explorer
```

```
var ie = navigator.userAgent.toLowerCase().indexOf('msie')!=-1;
```

```
if(ie) {
```

```
    coordenadaX = evento.clientX + document.body.scrollLeft;
```

```
    coordenadaY = evento.clientY + document.body.scrollTop;
```

```
}
```

```
else {
```

```
    coordenadaX = evento.pageX;
```

```
    coordenadaY = evento.pageY;
```

```
}
```

```
alert("Has pulsado el ratón en la posición: " + coordenadaX + ", " + coordenadaY + " respecto de la página web");
```

La variable `ie` vale `true` si el navegador en el que se ejecuta el script es de tipo Internet Explorer (cualquier versión) y vale `false` en otro caso. Para el resto de navegadores, las coordenadas respecto del origen de la página se obtienen mediante las propiedades `pageX` y `pageY`. En el caso de Internet Explorer, se obtienen sumando la posición respecto de la ventana del navegador

(clientX, clientY) y el desplazamiento que ha sufrido la página
(document.body.scrollLeft, document.body.scrollTop).