Eventos

Después de la introducción a DOM, es hora de explorar esta área, abordando eventos en JavaScript. En este capítulo, vamos a explorar el uso de eventos sin DOM, con DOM-0 (inventado por Netscape) y con DOM-2.

Vamos a ver cómo configurar estos eventos, usar, modificar su comportamiento, etc. Después de este capítulo, serás capaz empezar a interactuar con el usuario, lo que permite crear páginas web interactivas que pueden responder a las diversas acciones realizadas ya sea por el visitante o por el navegador.

¿Cuáles son los eventos?

Los eventos pueden activar una función cuando una acción se ha producido o no. Por ejemplo, puede ser la aparición de una ventana de `alert()` cuando el usuario cambia de área o zona de una página Web.

"Zona" es un término un tanto vago, es mejor hablar de elemento (HTML en la mayoría de los casos). Así, puedes agregar un evento a un elemento de tu página web (por ejemplo, un) para garantizar la activación de un código javascript cuando el usuario realizase una acción sobre el tema en cuestión.

Teoría

- Lista de los eventos

Hay numerosos eventos, todos más o menos útiles. Aquí está la lista de los principales eventos y acciones en que se activan para realizar:

Nombre del evento	Acción que se ejecuta
click	Pulsar y soltar sobre el elemento
dblclick	Doble pulsación sobre el elemento
mouseover	Hace entrar el cursor en el elemento
mouseout	Hace salir el cursor del elemento
mousedown	Pulsar sin soltar el botón izquierdo sobre el elemento
mouseup	Soltar el botón izquierdo del ratón del elemento

mousemove	Desplazar el cursor sobre el elemento
keydown	Pulsar sin soltar una tecla sobre el elemento
keyup	Soltar una tecla del elemento
keypress	Pulsar y soltar una tecla
focus	Focalizar en un elemento
blur	Cancelar la focalización en un elemento
change	Cambiar el valor de un elemento específico (formularios de entrada, casilla de verificación, etc.)
select	Seleccionar el contenido de un campo de texto (input, textarea, etc.)

Como se dijo anteriormente que `MouseDown` y `MouseUp` se disparan con el botón izquierdo del ratón. Esto no es del todo exacto, éstos pueden desencadenar eventos con otros botones del ratón como el clic de la rueda o el botón derecho. Sin embargo, esto no funciona con todos los navegadores como Firefox que ha decidido bloquear esta posibilidad. El uso de estos dos eventos por lo tanto está limitado generalmente al botón izquierdo del ratón.

Sin embargo, esto no es todo, también hay dos elementos específicos para <form>, como sigue:

Nombre del evento	Acción que se ejecuta
submit	Envía el formulario
reset	Reinicia el formulario

solo se enumeran algunos eventos existentes, pero rápidamente se aprenden a usar después de un breve paso por lo que se conoce como el foco.

- De vuelta al foco

Foco define lo que se puede llamar la "focalización" de un elemento. Cuando un elemento se focaliza, recibirá todos los eventos de tu teclado. Un ejemplo sencillo es utilizar un texto <input> de tipo texto: si hace clic en él, tiene el foco. También se dice que está focalizado, y si escribes caracteres en el teclado los verás aparecer en la entrada en cuestión.

El foco se puede aplicar a numerosos elementos, así que si presionas la tecla Tab de tu teclado mientras estás en una página web, tendrás un elemento de destino o focalizado que recibirá todo lo que escribas en en teclado. Por ejemplo, si tienes un enlace de foco y pulsas la tecla Intro en el teclado te redirigirá a la URL del enlace.

Práctica

- Utilización de los eventos

Ahora que has visto los teóricos (y aburrido) eventos, vamos a ser capaces de pasar a alguna práctica. Sin embargo, en un primer momento, es una cuestión de cómo se descubre un evento en particular y cómo reacciona, así que veremos cómo utilizarlos sin DOM, que es considerablemente más limitado.

Evento click en un único :

Código: HTML

```
<span onclick="alert('Hola Click');"> Púlseme </ span>
```

Como se puede ver, es suficiente hacer clic en el texto para que se muestre el cuadro de diálogo. Para lograr este hemos añadido a nuestro un atributo que contiene dos letras "on" y el nombre de nuestro evento "click", así obtenemos "onclick".

Este atributo tiene un valor que es un código JavaScript, que puede escribir casi cualquier cosa que quieras, pero deben tener el atributo entre comillas sencillas.

- La palabra clave `this`

Esta palabra clave se supone que no se utilizará ahora, pero siempre es bueno conocerla para los eventos. Se trata de una propiedad que apunta al objeto en uso. Así que si utilizas esta palabra clave cuando se activa un evento, el objeto al que apunta será el elemento que desencadena el evento. Ejemplo:

Código: HTML

```
<span onclick = "alert ('Este es el contenido del elemento que has pulsado: \ n \ n '+ this.innerHTML) "> Púlseme</ span>
```

Esta palabra clave no ayudará a corto plazo, para su uso con eventos DOM o DOM 1. Si intentas utilizar DOM-2 entonces podrías tener algunos problemas con Internet Explorer. Investigamos una solución más adelante.

- De vuelta al foco

Para mostrar lo que es un buen foco, aquí hay un ejemplo que mostrará cómo se ve en un `input` tradicional y un enlace.

Código: HTML

```
<input id = "entrada" type = "text" size = "50" value = "Haga clic aquí!"
onfocus = "this.value = 'Pulse ahora la tecla Tab. '; "onblur =" this.value ='
Pulse aquí '; "/>
<br /> <br/>

<a href = "#" onfocus = "document.getElementById ('input').value = 'Ahora
tiene el foco en el vínculo'"> Un enlace</ a>
```

Como puedes ver, cuando se hace clic en `input`, "posee" el foco: se ejecuta el evento y por lo tanto muestra un texto diferente al que solicitará al pulsar la tecla Tab. Al pulsar la tecla Tab se permite mover el foco al siguiente elemento. Es evidente que, al pulsar este botón se pierda el foco de `input`, que desencadena el evento `onblur` (lo que significa la pérdida de foco) y pone el foco en el vínculo que muestra su mensaje a través del evento de foco.

- Bloquear la acción por defecto de ciertos eventos

Ahora, un pequeño problema: cuando se quiere aplicar un evento `click` en un enlace, ¿qué es lo que está pasando? Mira por ti mismo:

Código: HTML

```
<a href = "http://www.barzanallana.es" onclick = "alert ('Ha hecho clic') ">
Púlsame </ a>
```

Si has probado el código, te habrás dado cuenta de que la función `alert()` ha funcionado bien, pero después de que se ha redirigido al sitio barzanallana.es, o si desea bloquear esta redirección, para ello, sólo tienes que añadir el código de retorno `return false;` en nuestro evento `click:.`

Código: HTML

```
<a href = "http://www.barzanallana.es" onclick = "alert ('Ha hecho clic');
return false "> Púlseme</ a>
```

Aquí el `return false` es sólo para bloquear la acción predeterminada del evento que lo desencadena. Ten en cuenta que el uso de `return true;` te permite ejecutar el evento como si nada hubiera pasado. Claramente, como si no usáramos `return false`;. Esto puede ser

útil si se utiliza, por ejemplo, `confirm()` en tu evento.

- El uso de "javascript:" en los enlaces, una técnica prohibida

En algunos casos, tendrás que crear vínculos sólo para darles un evento de `clic` y no para proporcionar un enlace al que redirigir. En tales casos, es común ver código como este:

Código: HTML

```
<a href="javascript: alert('Ha hecho click');"> Púlseme</ a>
```

Te recomendamos encarecidamente que no lo hagas. Si lo haces de todos modos, no digas que ha aprendido Javascript través de este curso. Se trata de un viejo método para insertar JavaScript directamente en el atributo `href` del enlace, con sólo añadir javascript: al comienzo del atributo. Esta técnica es obsoleta y estaríamos decepcionados si la usas. Se ofrece una alternativa:

Código: HTML

```
<a href = "#" onclick = "alert ('Ha hecho clic en'); return false"> Pulse</ a>
```

Específicamente, ¿qué cambios se han realizado? Fue reemplazado el `javascript:` primero por una almohadilla (#) y luego puesto nuestro código Javascript en el evento correspondiente (`click`). Además, se libera el atributo `href`, que nos permite, si es necesario, dejar un enlace para quienes no activen Javascript o incluso para los que para abrir enlaces usan una nueva ventana.

Pero ¿por qué `return false;`? El hecho es que las redirecciones son hacia la parte superior de la página web, que no es lo que queremos. Por lo tanto, se bloquea esta redirección con nuestro fragmento de código.

Ahora ya sabes que el uso de javascript: en enlaces está prohibido y esto ya es una buena cosa. Sin embargo, ten en cuenta que el uso de un único enlace para desencadenar un evento de `clic` no es una buena cosa, más bien prefieres el uso de una etiqueta `<button>` a la que se le quitó el estilo CSS.

La etiqueta <a> está diseñada para redirigir a una página web y no para usarla como desencadenante de eventos a través de DOM.

Eventos mediante DOM

Ahora que hemos visto el uso sin eventos de DOM, vamos a utilizarlos a través de la interfaz implementada por Netscape llamada DOM-0 y base del estándar actual: DOM-2.

- DOM-0

Esta interfaz es antiguo, pero no es necesariamente carece de interés. Es muy útil para crear eventos y a veces puede ser preferible al DOM-2. Comenzamos por crear un código simple con un evento de `clic`:

Código: HTML

```
<span id="pulseme"> Haga  clic </span>
<script>
        var elemento = document.getElementById ('pulseme');
        element.onclick = function () {
        alert ("Hizo clic")
        };
</script>
```

Por lo tanto, vamos a pasar por lo que hemos hecho en este código:

- En primer lugar, se obtiene todo elemento HTML con identidad `pulseme`;
- Se accede a la propiedad `onclick` que se le asigna una función anónima;
- En la misma función, se hace una llamada a la función `alert()` con un texto de parámetro.

Como puedes ver, ahora los eventos se definen no en el código HTML, sino directamente en JavaScript. Cada evento estándar tiene una propiedad cuyo nombre, es precedido por las letras "on". Esta propiedad no tiene por valor más que un código JavaScript, ya sea el nombre de una función o una función anónima.

En resumen, en todos los casos, es necesario proporcionar una función que contiene el código que se ejecuta cuando se activa el evento. Al borrar un evento con DOM-0, simplemente es suficiente asignar una función anónima vacía:

Código: JavaScript

```
element.onclick = function () {};
```

Esto es todo para los eventos DOM-0, ahora volvemos al núcleo de los acontecimientos: DOM-2 y al `objet Event`.

DOM-2

En primer lugar, ¿por qué DOM-2 y no DOM o DOM-0 en absoluto? En método sin el DOM, es simple: no se puede utilizar el objeto `event` que aporta una gran cantidad de información sobre el evento activado. Por tanto, es aconsejable dejar de lado este método (se ha reseñado simplemente para conocerlo).

En cuanto a DOM-0, tiene dos problemas principales: es viejo, y no permite crear varias veces el mismo evento.

DOM-2, permite la creación múltiple de un mismo evento y también maneja el objeto `event`. Es decir el objeto, el DOM-2 es útil,

¿Está claro que siempre debo utilizar DOM-2?

No siempre. La técnica sin DOM se ha de desterrar, mientras que el uso de DOM-0 es ampliamente posible, todo depende de tu código. Por otra parte, en la mayoría de los casos, puedes elegir DOM-0 por su facilidad de uso y velocidad de ejecución. Esta no es, en general, cuando tienes que crear múltiples eventos del mismo tipo (`clic`, por ejemplo), que usarás DOM-2

Sin embargo, hay una situación en la que siempre se debe usar DOM-2, es cuando se crea un *script* JavaScript que deseas distribuir. ¿Por qué? Digamos que tu `script` agrega (con DOM-0) un evento de clic al elemento <body>, si la persona que decide agregar el script a la página ya ha creado este evento (con DOM-0, por supuesto), entonces habrá una reescritura desde esta versión del DOM que no acepta más de un evento del mismo tipo.

- DOM-2 según los estándares web

En cuanto a las otras interfaces de eventos, un ejemplo con el evento `click`:

Código: HTML

```
<span id="pulsame"> Haga clic </span>
<script>
        var elemento = document.getElementById ('pulsame');
        element.addEventListener ('click', function () {
        alert ("Hizo clic")
        }, false);
</script>
```

En concreto, ¿qué cambia con respecto a DOM-0? No usamos una propiedad, sino un método llamado `addEventListener()` (que no funciona en Internet Explorer 8 y versiones anteriores) que toma tres parámetros:

- Nombre del evento (sin las letras "on");

- La función a ejecutar ;
- Un booleano para especificar si se desea utilizar la fase de captura o la de burbujeo. Explicaremos este concepto posteriormente. Sólo sabe que se utiliza generalmente `false` para parámetro.

Una explicación se necesita poco para quienes no han entendido el código anterior: hemos utilizado el método `addEventListener()`, que se anota en tres líneas:

- La primera línea contiene la llamada al método `addEventListener()`, el primer parámetro, y la inicialización de la función anónima para el segundo parámetro;
- La segunda línea contiene el código de la función anónima;
- La tercera línea contiene la llave de cierre de la función anónima, y el tercer parámetro.

Este código se escribe, más rápidamente:

Código: JavaScript

```
var elemento = document.getElementById ('pulsame');
var miFuncion = function () {
        alert ("me hizo clic")
};

element.addEventListener ('click', miFuncion, false);
```

Como se indicó anteriormente, DOM-2 permite la creación de múltiples eventos idénticos para el mismo elemento, por lo que puedes hacer esto:

Código: HTML

```
<span id="pulsame"> Haga clic </ span>
        <script>
        var elemento = document.getElementById ('pulsame'); // Primero haga clic en el evento
        element.addEventListener ('click', function () {
        alert ("Uno");
        }, false); // Segundo evento click
        element.addEventListener ('click', function () {
        alert ("Dos");
        }, false);
        </ script>
```

Si ejecutas este código, los acontecimientos pueden haber sido provocados en el orden de creación, sin embargo, no es necesariamente el caso para cada prueba. En efecto, la orden de disparo es un poco al azar.

Volviendo ahora a la supresión de eventos. Esto se realiza con el método `removeEventListener()` y de una forma muy simple:

Código: JavaScript

```
element.addEventListener ('click', miFuncion, false); // Crear el evento
element.removeEventListener ('click', miFuncion, false) // Eliminar el evento repasando los mismos parámetros
```

Toda supresión de evento con DOM-2 se hace con los mismos parámetros utilizados en su creación. Sin embargo, esto no funciona con la misma facilidad con funciones anónimas. Todos los eventos DOM-2 creados en una función anónima son particularmente difíciles de eliminar, ya que deben tener una referencia a la función en cuestión, que generalmente no es el caso con una función anónima.

DOM-2 según Internet Explorer

Hasta ahora no hemos tenido casi ningún problema con Internet Explorer, respetaba las normas, pero esto no podía durar para siempre. Y ahora `addEventListener()` y `removeEventListener()` no funcionan. Entonces, ¿qué debería utilizarlos? Los métodos `attachEvent()` y `detachEvent()` se utilizan del mismo modo que los métodos estándar, excepto que el tercer parámetro no existe y que el evento debe ser el prefijo "on"). Ejemplo:

Código: JavaScript

```
//  Se crea el evento
element.attachEvent('onclick', function() { alert('Prueba'); });
// Se suprime el evento, repasando los mismos parámetros
element.detachEvent('onclick', function() {alert('Prueba);
});
```

Internet Explorer, a partir de la versión 9 es compatible con los métodos estándar. A diferencia de versiones anteriores tienes que hacer juegos malabares entre los métodos estándar y los métodos de IE. Por lo general, se utiliza un código como el siguiente para gestionar compatibilidad entre navegadores:

Código: JavaScript

```
function addEvent(elemento, event, func) {

    if (elemento.addEventListener) {// Si el elemento tiene el método addEventListener ()
    element.addEventListener (evento, func, false);
```

```
        } else { // Si nuestro elemento no tiene el método
        addEventListener ()
        element.attachEvent ('on' + evento, func);
        }
}
addEvent (elemento, 'click', function () {
// Tu código
});
```

Fases de captura y burbuja

Recuerda que nuestro método `addEventListener()` toma tres parámetros. Dijimos que volveríamos sobre el uso del tercer parámetro posteriormente. Esto ha llegado.

Captura, burbuja. ¿De qué vamos a hablar? . Estas son dos etapas distintas de la ejecución de un evento. La primera captura (*capture* en Inglés) se ejecuta antes del comienzo del evento, mientras que la segunda, burbuja (*bubbling* en Inglés), se ejecuta después de que el evento se activó. Estos dos parámetros definen el sentido de propagación de los eventos.

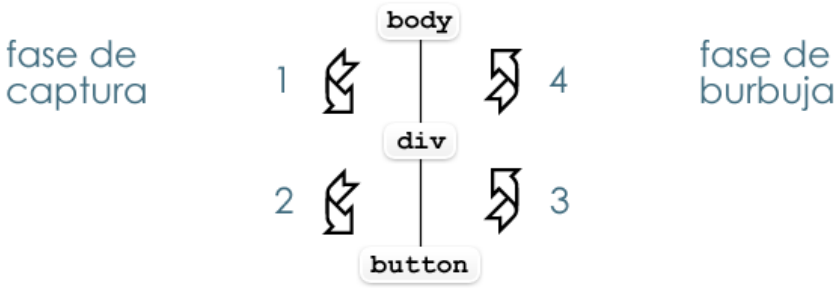
¿Pero qué es la propagación de un evento? Para explicar esto, considera un ejemplo con estos dos elementos HTML:

Código: HTML

```
<div>
<span> texto </span>
</div>
```

Si se le asigna una función al evento `click` de cada uno de estos dos elementos y se hace clic sobre el texto, ¿qué evento se disparará por primera vez? Buena pregunta.

Nuestra respuesta se encuentra en las fases de captura y burbuja. Si decides usar la captura, el evento se disparará primero y luego viene . Sin embargo, si se utiliza burbuja, se dispara primero el evento seguido de .



Aquí está un poco de código que pone en práctica el uso de estas dos fases:

Código: HTML

```
<div id="capt1">
        <span id="capt2">Instrucciones para la fase de captura. </ span>
</div>
<div id="burb1">
        <span id="burb2">Instrucciones para la fase de burbuja. </ span>
</div>
<script>

var     capt1 = document.getElementById ('capt1')
        capt2 = document.getElementById ('capt2')
        burb1 = document.getElementById ('burb1')
        burb2 = document.getElementById ('burb2');

        capt1.addEventListener ('click', function () {alert ("El evento de div
        acaba de desencadenarse.");
        },true);

        capt2.addEventListener ('click', function () {alert ("El evento de
        span acaba de desencadenarse.");
        },true);

        burb1.addEventListener ('click', function () {alert ("El evento de div
        acaba de desencadenarse.");
        },false);

        burb2.addEventListener('click', function() {alert ("El evento de
        span acaba de desencadenarse.");
```

```
        },false);
</script>
```

- El lado práctico

Bueno, ya sabes para qué sirven estas estas dos fases, sin embargo, aunque en teoría este sistema es útil en la práctica casi nunca sirve por la sencilla razón de que el método `attachEvent()` en Internet Explorer sólo soporta la fase de burbuja, por lo que se suele poner el último parámetro `addEventListener()` como `false`.

Otra cosa, los eventos sin DOM o con DOM-0 no generan más que la fase de burbuja.

Dicho esto, no te preocupes, no lo has aprendido en vano. Siempre es bueno saberlo, tanto para el conocimiento general de Javascript, como también para comprender lo que se está haciendo cuando se está codificando.

Objeto Event. Generalidades

Ahora que hemos visto cómo crear y borrar eventos, podemos pasar al objeto `Event`.

En primer lugar, ¿para qué sirve este objeto? Para proporcionar una gran cantidad de información sobre el evento que está activado. Por ejemplo, puedes saber las teclas pulsadas, las coordenadas del cursor, el elemento que desencadenó el evento ... Las posibilidades son infinitas. Este objeto es muy especial en el sentido de que sólo está disponible cuando se activa un evento. El acceso al mismo solo puede hacerse con una función ejecutada por un evento, esto se hace de la siguiente manera con DOM-0:

Código: JavaScript

```
elemento.onclick = function (e) { // El argumento "e" recuperará una referencia hacia el objeto "Event"
alert (e.type) // Muestra el tipo de evento (click, mouseover, etc)
};
```

Y de esta manera con DOM-2:

Código: JavaScript

```
elemento.addEventListener ('click', function (e) { // El argumento "e" recuperará una referencia al objeto "Event"
alert(e.type) // Muestra el tipo de evento (click, mouseover, etc)
}, false);
```

Cuando se transmite un objeto `Event` como parámetro, generalmente se llama e. Es importante tener en cuenta que el objeto `Event` se puede recuperar en un argumento que no sea e. Por ejemplo recuperar en un argumento con nombre `test_hello, hola`, o si no, después de todo, el objeto `Event` simplemente se pasa por referencia al argumento de tu función, lo que permite elegir el nombre que desees.

En Internet Explorer (todas las versiones anteriores a la 9), si deseas utilizar DOM-0 encontrarás que el objeto `Event`sólo es accesible mediante `window.event`, con lo que no es necesario (para IE, por supuesto) utilizar un argumento de la función ejecutada por el evento. A la inversa, si está utilizando DOM-2, no es necesario emplear `window.event`. Para mantener la compatibilidad con otros navegadores, por lo general, se utiliza este código en la función realizada por el evento: `e = e || window.event;`.

Funcionalidades del objeto Event

Ya has descubierto la propiedad `type` que te permite saber qué tipo de evento se dispara. Ahora, pasamos a descubrir otras propiedades y métodos que posee este objeto (ten cuidado, no se presenta todo, solo lo esencial):

- Obtener el elemento del actual evento disparado

Una de las propiedades más importantes de nuestro objeto de destino se llama `target`. Permite recuperar una referencia al elemento cuyo evento se activó (como `this` para eventos sin DOM o con DOM-0), por lo que puede muy bien cambiar el contenido de un elemento sobre el que se haga clic:

Código: HTML

```
<span id="pulsame"> Pulse aquí </ span>
<script>
        var  pulsame = document.getElementById ('pulsame');
        pulsame.addEventListener ('click', function (e)
        {e.target.innerHTML = 'Ha hecho clic en';
        }, false);
</script>
```

Como siempre hay un problema en alguna parte, Internet Explorer (versiones anteriores a la 9) no son compatibles con esta propiedad. O más bien, mantienen a su manera la propiedad `srcElement`. Aquí está el mismo código que el anterior pero compatible con todos los navegadores (como Internet Explorer):

Código: HTML

```
<span id="pulsame"> Pulse aquí</ span>
<script>
        function addEvent (element, event, func) { // Se reutiliza nuestra
        función de compatibilidad para los eventos DOM-2
                if (element.addEventListener) {
                        element.addEventListener(event, func, false);
                } else {
                element.attachEvent('on' + event, func);
                }
        }

        var pulsame = document.getElementById ('pulsame');

        addEvent (pulsame, 'click', function (e) {
        var target = e.target | | e.srcElement // Si has olvidado esta
        especificación del operador OU, ir al capítulo de condiciones
        target.innerHTML = 'Ha hecho clic en';
        });
</script>
```

Ten en cuenta que aquí tenemos un código compatible con Internet Explorer debido a que el objetivo era lograr la compatibilidad, pero para el resto de códigos no lo haremos si no es necesario. Es por lo tanto recomendable usar un navegador actualizado para probar todos los códigos de este curso.

- Obtener el elemento en el origen del evento desencadenante

¿No se trata de la misma cosa? pues no. Para explicarlo simplemente, eventos aplicados a un elemento primario pueden propagarse a sí mismos y a los elementos hijos en el caso de los eventos `mouseover, mouseout, mousemove, click`... y otros eventos menos utilizados. Mira este ejemplo para comprender mejor:

Código: HTML

```
<p id="resultado"></p>
<div id="padre1">
        Padre
        <div id="child1"> Niño N º 1 </div>
        <div id="child2"> Niño N ° 2 </div>
</div>

<script>
        var  padre1 = document.getElementById ('padre1')
        result = document.getElementById ('result');
```

```
        padre1.addEventListener ('mouseover', function (e) {
        result.innerHTML = "El desencadenador del  evento \ "MouseOver
        \" tiene la id: "+ e.target.id;
        }, false);
</script>
```

Al probar este ejemplo, puedes haber notado que la propiedad de destino devuelve siempre el disparador del evento, y lo que se pretende obtener es el elemento sobre el que se ha aplicado el evento. En otras palabras, queremos conocer el elemento en el origen de este evento, y no sus hijos.

La solución es simple: utilizar la propiedad `currentTarget` en lugar de `target`. Pruébala después de cambiar esta línea, la id indicada nunca va a cambiar:

Código: JavaScript

```
result.innerHTML = "El desencadenador de eventos \ "MouseOver \" tiene
el ID: "+ e.currentTarget.id;
```

Aunque esta propiedad parece atractiva para ciertas aplicaciones, es muy difícil de poner en práctica, para Internet Explorer (versiones anteriores a la 9) no lo soportan y no hay alternativa correcta (excepto en el uso de esta palabra clave con DOM-0 o sin DOM).

- Recuperar la posición del cursor

La posición del cursor es una información muy importante, mucha gente la usa durante muchos *scripts* arrastrar y soltar. En general, sirve para obtener la posición del cursor relativa a la esquina superior izquierda de la página web,. También es posible recuperar su posición relativa a la esquina superior izquierda de la pantalla. Sin embargo, en este tutorial nos limitaremos a la página web. Consulta la documentación del objeto `Event` si deseas obtener más información.

Para recuperar la posición de nuestro cursor hay dos propiedades: `clientX` para la posición horizontal y `clientY` para la posición vertical. Debido a que la posición del cursor cambia con cada movimiento del ratón, es lógico decir el evento que mejor se adapta a la mayoría de los casos es `mousemove`.

Está desaconsejado tratar de ejecutar la función `alert()` en un evento `mousemove` ya que rápidamente te sentirás abrumado con ventanas. Como de costumbre, he aquí un ejemplo para que puedas comprender:

Código: HTML

```
<div id="posicion"> </div>
<script>
```

```
        var posicion = document.getElementById ('posicion');
        document.addEventListener ('mousemove', function (e) {
        posicion.innerHTML = 'Posición X' + e.clientX + "px <br/> Posición
        Y: '+ e.clientY +' px ';
        }, false);
</script>
```

No es muy complicado, es posible que encuentres interés bastante limitado en este código, pero cuando se saben manipular las propiedades CSS de los elementos, puedes por ejemplo, ordenar elementos HTML con el cursor. Esto ya será mucho más útil.

- Recuperar el elemento en relación a un evento de ratón

Esta vez vamos a ver una propiedad un poco más "exótica" que puede ser muy útil. Se trata de `relatedTarget` y se utiliza con los eventos `mouseover` y `mouseout`.

Esta propiedad cumple con dos funciones diferentes según el evento utilizado. En el caso `mouseout`, proporciona el objeto del elemento en el cual el cursor se ha introducido, y con el evento `mouseover`, dará el objeto del elemento del que el cursor acaba de salir. He aquí un ejemplo que ilustra cómo funciona:

Código: HTML

```
<p id="resultado"> </p>

<div id="padre1">
        Padre: 1 <br />
        Mouseover en el niño
        <div id="child1"> Niño N º 1 </ div>
</div>

<div id="padre2">
        Padre 2 <br />
        Mouseout en el niño
        <div id="child2"> Niño N ° 2 </ div>
</div>

<script>
var     child1 = document.getElementById ('child1')
        child2 = document.getElementById ('child2')
        resultado = document.getElementById ('resultado');

child1.addEventListener ('mouseover', function (e) {
```

```
        result.innerHTML = "El elemento de la izquierda justo antes del
        cursor, que no entra sobre  el niño # 1 es: "+ e.relatedTarget.id;
        }, false);

child2.addEventListener ('mouseout', function (e) {
        result.innerHTML = "El elemento volado inmediatamente después
        del cursor que ha dejado el niño # 2 es: "+ e.relatedTarget.id;
        }, false);
</script>
```

Para Internet Explorer (todas las versiones anteriores a la 9), debes utilizar las propiedades y `fromElement` y `ToElement` como sigue:

Código: JavaScript

```
child1.attachEvent ('onmouseover', function (e) {
result.innerHTML = "El elemento dejado justo antes de que elcursor
no entre al hijo #1 es: "+ e.fromElement.id;
});
child2.attachEvent ('onmouseout', function (e) {
result.innerHTML = "Elemento que sobrevuela el cursor justo después
que ha dejado el hijo #2 es: "+ e.toElement.id;
});
```

- Recuperar las teclas pulsadas por el usuario

La recuperación de las pulsaciones de teclado se hace a través de tres eventos diferentes. Dicho así, parece complejo, pero verás que al final todo es mucho más sencillo de lo que parece.

Los eventos `KeyUp` y `KeyDown` están diseñados para capturar las pulsaciones de teclado. Así que es posible detectar cuando se pulsa una tecla, por ejemplo A, e incluso el botón de la tecla Ctrl. La diferencia entre ambas radica en el orden de los acontecimientos desencadenados: `Keyup` se activa cuando se suelta una tecla mientras que `keydown` se activa al pulsar una tecla (como `mousedown`). Sin embargo, ten cuidado con estos dos eventos:

El evento `keypress`, tiene un propósito completamente diferente: capturará las teclas que escriben un carácter, olvidate de Ctrl, Alt y otras como éstas, que no muestran un carácter. Entonces inevitablemente, uno se pregunta ¿se puede utilizar para eventos? Su ventaja radica en su capacidad para detectar las combinaciones de teclas. Así que, si ejecutas la combinación Mayús + A, el evento `keypress` detectará una A mayúscula donde los eventos `KeyDown` y `KeyUp` se dispararán dos veces, una vez para la tecla `Shift` y una segunda vez para la tecla A.

¿Qué propiedad uso para recuperar un carácter de repente? Si tuviéramos que enumerar todas las propiedades que pueden aportar valor, habría tres: `keyCode`, `charCode` y `which`. Estas propiedades devuelven el código ASCII correspondiente a la tecla pulsada. Sin embargo, la propiedad `keyCode` es más que suficiente en todos los casos, como se puede ver en el ejemplo::

Código: HTML

```
<p>
id = "campo" <input type="text" />
</p>
        <table>
        <tr>
        <td> pulsa </ td>
        <td id="down"> </ td>
        </tr>
        <tr>
        <td> presiona</ td>
        <td id="press"> </ td>
        </tr>
        <tr>
        <td> suelta </ td>
        <td id="up"> </ td>
        </tr>
        </table>

<script>
var campo = document.getElementById ("campo")
pulsa = document.getElementById ("pulsa")
presiona = document.getElementById ('presiona')
suelta = document.getElementById ("suelta");

document.addEventListener ('KeyDown', function (e) {
pulsa.innerHTML = e.keyCode;
}, false);
document.addEventListener ('pulsa', function (e) {
presiona.innerHTML = e.keyCode;
}, false);
document.addEventListener ('suelta', function (e) {
suelta.innerHTML = e.keyCode;
}, false);
</script>
```

- No quiero obtener un código, pero sí el carácter

En este caso, sólo hay una solución, el método: `fromCharCode`. Necesita un número infinito de argumentos como parámetros. Sin embargo, por razones un tanto peculiares que se discutirán más adelante, ten en cuenta que este método se utiliza con la cadena de prefijo, como sigue:

Código: JavaScript

```
String.fromCharCode (/ * valor * /);
```

Este método está diseñado para convertir los valores ASCII a caracteres legibles. Así que ten cuidado al utilizar este método con un evento de pulsación de teclas para evitar mostrar, por ejemplo, el código del carácter correspondiente a la tecla Ctrl, no va a funcionar. Por último, he aquí un pequeño ejemplo:

Código: JavaScript

```
String.fromCharCode (84, 101, 115, 116) // Muestra: text
```

- Bloquear la acción por defecto de ciertos eventos

Hemos visto que es posible bloquear la acción por omisión de ciertos eventos, como la redirección de un enlace a una página Web. Sin DOM-2, esta operación era muy simple, basta con escribir `return false`;. Con el objeto `Event` se ve todo muy simple, es suficiente llamar al método `preventDefault()`. Tomemos el ejemplo que hemos utilizado para los eventos sin DOM y utilizaremos por lo tanto este método:

Código: HTML

```
<a id="link" href="http://www.barzanallana.es"> Pulse aquí</ a>
<script>
        var link = document.getElementById ('link');
        link.addEventListener ('click', function (e) {
        e.preventDefault () // Se bloquea la acción predeterminada de
        este evento
        alert ('Ha hecho clic en');
        }, false);
</script>
```

Es muy sencillo y funciona perfectamente con todos los navegadores, excepto para las versiones de Internet Explorer anteriores a la 9. Para IE, tendrás que usar la propiedad `returnValue` y asignar el valor false para bloquear la acción por defecto:

Código: JavaScript

```
e.returnValue = false;
```

Para un código compatible con todos los navegadores, hay que que usar:

Código: JavaScript

```
e.returnValue = false;
si (e.preventDefault) {
e.preventDefault ();
}
```

Solución de problemas de herencia de eventos

En Javascript, existe un problema común que nos proponemos investigar y resolver para evitar muchos problemas cuando aparezca.

- El problema

En lugar de explicar el problema, vamos a constatarlo. Así que toma este código HTML y CSS:

Código: HTML

```
<div id="miDiv">
<div> Texto 1 </ div>
<div> Texto 2 </ div>
<div> Texto 3 </ div>
<div> Texto 4 </ div>
</div>
<div id="resultados"> </ div>
```

Código: CSS

```
#miDiv, #resultados {
margen: 50px;

}
#miDiv {
padding: 10px;
width: 200px;
text-align: center;
background-color: # 000;
}
```

```
#miDiv div {
margin: 10px;
background-color: # 555;
}
```

Ahora vamos a ver lo que queremos. Nuestro objetivo aquí es asegurarse de detectar cuando el cursor entra en nuestro elemento #miDiv y cuando sale. Así que vas a pensar que no hay nada más fácil que ejecutar el código:

Código: JavaScript

```
var miDiv = document.getElementById ('miDiv')
resultados = document.getElementById ("resultados");

miDiv.onmouseover = function () {
resultoads.innerHTML + = "El cursor ha entrado.";
};

miDiv.onmouseout = function () {
results.innerHTML + = "El cursor está fuera.";
};
```

¿Qué sucede al probarlo? ¿Has intentado mover el cursor por toda la superficie de #miDiv? En realidad, hay demasiadas líneas en nuestros resultados. no veo de donde puede venir esta situación. Nadie ve a primera vista desde donde pueden entrar. De hecho, la preocupación es estúpida y ha sido muy discutida a través de este capítulo, lee esto:

Citación:

Los eventos aplicados a un elemento primario pueden propagarse a los elementos hijos es el caso de los eventos `mouseover, mouseout, mousemove, click`... y otros eventos menos utilizados.

Aquí está nuestro problema: los hijos heredan propiedades aplicadas a los eventos antes mencionados, al mover el cursor desde el #miDiv a un hijo, se activará el evento `mouseout` sobre `#myDiv` y el evento `mouseover` sobre hijo .

La solución

Para superar este problema, hay una solución bastante complicada. Recuerda `relatedTarget`, propiedad abordada en este capítulo . Su objetivo es detectar cuál es el elemento al que se mueve el cursor o de qué elemento proviene.

Por lo tanto, tenemos dos casos:

En el caso del evento `mouseover`, se ha de detectar el origen del cursor. Si el cursor viene de un hijo de `#miDiv` entonces el código del evento no debe ser ejecutado. Si proviene de un elemento fuera de `#miDiv` la ejecución de código puede efectuarse.

En el caso de `mouseout`, el principio es similar, excepto que aquí tenemos que detectar el destino del cursor. En el caso de que el destino del cursor sea un hijo de #miDiv entonces el código de evento no se ejecuta, de lo contrario se ejecutará sin problemas.

Ponlo en práctica con el evento `mouseover` para comenzar. Éste es el código original:

Código: JavaScript

```
miDiv.onmouseover = function () {
results.innerHTML + = "El cursor ha entrado.";
};
```

Ahora tenemos que conseguir el elemento de origen. Dado que `relatedTarget` no es compatible con versiones anteriores de Internet Explorer, vamos a tener que ajustar un poco:

Código: JavaScript

```
miDiv.onmouseover = function(e) {

e = e || window.event; // Compatibilidad IE
var relatedTarget = e.relatedTarget || e.fromElement; // Idem

results.innerHTML += "El cursor acaba de entrar.";
};
```

Ahora, necesitamos saber si el elemento es un hijo directo de `miDiv` o no. La solución es volver a lo largo de sus elementos padres hasta `miDiv` en el elemento <body> que designa el elemento HTML más alto en nuestro documento. Por lo tanto, se usa un bucle `while`:

Código : JavaScript

```
miDiv.onmouseover = function(e) {

    e = e || window.event; // Compatibilidad IE
    var relatedTarget = e.relatedTarget || e.fromElement; // Idem

    while (relatedTarget != miDiv && relatedTarget.nodeName !
    = 'BODY') {
```

```
        relatedTarget = relatedTarget.parentNode;
        }

        results.innerHTML += “El cursor acaba de entrar.";
};
```

Así, encontramos en nuestra variable `relatedTarget` el primer elemento que coincide con los criterios, sea `miDiv` o <body>. Sólo tenemos que introducir una condición que ejecutará el código de nuestro evento sólo si la variable `relatedTarget` no apunta al elemento `myDiv`:

Código: JavaScrip

```
miDiv.onmouseover = function(e) {

        e = e || window.event; // Compatibilidad IE
        var relatedTarget = e.relatedTarget || e.fromElement; // Idem

        while (relatedTarget != miDiv && relatedTarget.nodeName !
        ='BODY') {
        relatedTarget = relatedTarget.parentNode;
        }

        if (relatedTarget != miDiv) {
        results.innerHTML += "El cursor acaba de entrar.";
        }
};
```

Sin embargo, todavía hay un escenario pequeño que no se ha logrado y que puede causar problemas. <body> no necesariamente cubre toda la página web en el navegador, de modo que el cursor puede venir de un elemento situado incluso más alto que <body>. Esto corresponde a la etiqueta <html> - es el elemento `document` de Javascript - así que tenemos que hacer un pequeño cambio para que quede claro que si el cursor llega de `document` no necesariamente viene de `miDiv`:

Código: JavaScript

```
miDiv.onmouseover = function(e) {

        e = e || window.event; // Compatibilidad IE
        var relatedTarget = e.relatedTarget || e.fromElement; // Idem

        while (relatedTarget != myDiv && relatedTarget.nodeName !
        = 'BODY' && relatedTarget != document) {
        relatedTarget = relatedTarget.parentNode;
```

```
        }

        if (relatedTarget != miDiv) {
        results.innerHTML += "El cursor acaba de entrar.";
        }
};
```

Ahora nuestro evento `mouseover` trabaja como queríamos. Lo que nos queda por adaptar, es el código para el evento `mouseout`. En este evento se utiliza el mismo código para `mouseover`, considerando:

- El texto que se muestra no es el mismo;
- Ya no usamos `fromElement` sino `toElement` porque queremos el elemento de destino. Lo que nos da esto:

Código: JavaScript

```
        miDiv.onmouseout = function(e) {

                e = e || window.event; // Compatibilidad IE
                var relatedTarget = e.relatedTarget || e.toElement; // Idem

                while (relatedTarget != miDiv &&
                relatedTarget.nodeName !='BODY' && relatedTarget !=
                document) {
                relatedTarget = relatedTarget.parentNode;
                }

                if (relatedTarget != miDiv) {
                results.innerHTML += "Le curseur vient de sortir.<br />";
                }
        };
```

Resumen

- Los eventos se utilizan para llamar a una función a partir de una acción generada por el usuario o no.

- Existen diferentes eventos para detectar ciertas acciones como hacer clic, navegación, campos de formulario y teclado de control.

- DOM-0 es la antigua forma de captura de eventos. DOM-2 presenta el objeto `Event` y el famoso método `addEventListener()`. Se debe tener cuidado, porque Internet Explorer no reconoce el método `attachEvent ()`.

- El objeto `Event` permite obtener todo tipo de información relacionada con el evento activado: tipo, desde que elemento a ha desencadenado, posición del cursor, ... También es posible bloquear la acción de un evento con `preventDefault ()`.

- A veces, un evento aplicado a uno de los padres se extiende a sus hijos. Esta herencia de los eventos puede provocar un comportamiento inesperado.