Tema 7. Eventos

- Modelo básico de eventos
- Manejadores como atributos XHTML
- Manejadores semánticos
- Flujo de eventos
- Propagación de eventos
- Eventos DOM
- Manejadores de eventos de IE
- Manejadores de eventos de DOM
- El objeto *event*
- Propiedades de *event* en IE
- Propiedades de *event* en DOM

Introducción

- La interacción de JavaScript con HTML se controla a través de eventos que se producen cuando el usuario o el navegador manipula la página Web.
- Se produce un evento cuando
  - Se abre una página, se obtiene el foco, se pulsa un botón, se pulsa una tecla
  - Se pulsa sobre un elemento de la página
  - Se pasa con el cursor del ratón por encima de un elemento, etc.
- JavaScript permite asociar tareas determinadas a los eventos producidos mediante los controladores de eventos.
- El nivel 1 de DOM no incluye especificaciones relativas a los eventos JavaScript.
- El nivel 2 de DOM incluye ciertos aspectos relacionados con los eventos
- El nivel 3 de DOM incluye la especificación completa de los eventos de JavaScript. No obstante, la especificación de nivel 3 de DOM se publicó en el año 2004, más de diez años después de que los primeros navegadores incluyeran los eventos.
- Por este motivo, muchas de las propiedades y métodos actuales relacionados con los eventos son incompatibles con los de DOM. De hecho, navegadores como Internet Explorer tratan los eventos siguiendo su propio modelo incompatible con el estándar.

Modelo básico de eventos

- El modelo simple de eventos se introdujo en 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 con todos los navegadores y por tanto, el único que permite crear aplicaciones que funcionan de la misma manera en todos los navegadores.
- Cada elemento XHTML tiene definida 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 y un mismo elemento XHTML puede tener asociados diferentes eventos.
- El nombre del controlador de eventos en HTML se construye mediante el prefijo *on* seguido del nombre del evento.
  - El evento de pinchar un elemento con el ratón se denomina *onclick*
  - El evento asociado a la acción de mover el ratón se denomina *onmousemove*.

Listado de eventos (I)

Controlador	Descripción evento	Etiquetas
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

Listado de eventos (II)

Controlador	Descripción evento	Etiquetas
`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 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 como atributos XHTML

- La forma más sencilla de incluir un manejador de evento es mediante un atributo de XHTML.
- El método consiste en incluir un atributo XHTML con el nombre del controlador de evento que se quiere procesar.
- El contenido del atributo es una cadena de texto que contiene todas las instrucciones JavaScript que se ejecutan cuando se produce el evento.
- Ejemplos:

```
1. <input type="button" value="pulsa" onclick="funcion()"/>

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

3. <div style="padding: .2em; width: 150px; height: 60px; border:
thin solid silver" onmouseover="this.style.borderColor='black'"
onmouseout="this.style.borderColor='silver'">
...
</div>
```

Ejemplo 1

```
<html>
<head>
<title>Exemple 1</title>
<script type="text/javascript">
function reacciona(campo)
{
    campo.value="";
    alert("¡Introduzca un valor!")
    campo.focus()
}
</script>
</head>
<body>
<form method=post>
<input type=text name=campo onfocus="reacciona(this)">
</form>
</body>
</html>
```

Ejemplo 2

```
<html>
<head>
<title>Exemple 2</title>
<script type="text/javascript">
function hola(){
  nombre = prompt('Introduzca su nombre:','')
  alert('¡Hola ' + nombre + '!')
}
function adios(){
  alert('¡Adios ' + nombre + '!')
}
</script>
</head>
<body onload="hola()" onunload="adios()">
<h2>Página de carga y descarga</h2>
</body>
</html>
```

Ejemplo 3

```
<html>
<head>
<title>Exemple 3</title>
<script language="JavaScript">
function cambia_imagen(){
  document.images[0].src="mundo.png";
  window.status='La imagen cambia';
}
function imagen_normal(){
  document.images.door.src = "entrar.jpg";
   window.status='Ejemplo de eventos onMouseOver y onMouseOut'
}
</script>
<body>
<img src="door2.gif"  name="door" order=0 height=200 width=200
   onmouseover="cambia_imagen();" onmouseout="imagen_normal()">
</body>
</html>
```

Comportamiento

- Algunos de los controladores anteriores (*onclick, onkeydown, onkeypress, onreset y onsubmit*) permiten evitar el comportamiento por defecto del evento si se devuelve el valor *false*.
- Por otra parte, las acciones típicas que realiza un usuario en una página web pueden dar lugar a una sucesión de eventos.
- Si se pulsa por ejemplo sobre un botón de tipo *submit* se desencadenan los eventos
  - *onmousedown*
  - *onmouseup*
  - *onclick*
  - *onsubmit.*

Manejadores semánticos

- Al crear páginas web se recomienda separar los contenidos (XHTML) de la presentación (CSS). También se recomienda separar los contenidos (XHTML) de la programación (JavaScript).
- Mezclar JavaScript y XHTML complica excesivamente el código fuente de la página, dificulta su mantenimiento y reduce la semántica del documento final producido.
- Para realizar esta separación, asignaremos funciones a las propiedades DOM de los elementos XHTML. El código XHTML resultante es muy *"limpio",* ya que no se mezcla con el código JavaScript.
- *La* técnica de los manejadores semánticos consiste en:
  - Asignar un identificador único al elemento XHTML mediante el atributo *id*.
  - Crear una función de JavaScript encargada de manejar el evento.
  - Asignar la función a un controlador de evento del elemento XHTML mediante DOM.
- Además, las funciones externas pueden utilizar *this* para referirse al elemento que origina el evento.

Ejemplo 4

```
<html>
<head>
<title>Ejemplo 4</title>
<script type="text/javascript">
function mostrar() {
  alert('Pulsado');
}
</script>
</head>
<body>
<form>
<input id="boton" type="button" value="Pulsar" />
<script type="text/javascript">
document.getElementById("boton").onclick = mostrar;
</script>
</form>
</body>
</html>
```

Consideraciones

- Nótese en el ejemplo anterior que al controlador de eventos se le asigna la función sin paréntesis, :
  - `document.getElementById("boton").onclick = mostrar;`
- Si se añaden los paréntesis al final, en realidad se está invocando la función y asignando el valor devuelto por la función al evento *onclick* de elemento.
- Este método tiene el inconveniente de que los manejadores se asignan mediante las funciones DOM, que solamente se pueden utilizar después de que el elemento utilizado se haya cargado.
- 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*.
- Esta técnica utiliza una función anónima para asignar instrucciones al controlador *onload* de la página para asegurar que el código se ejecutará después de que la página se haya cargado.
  - `window.onload = function() {...}`

Ejemplo 5

```
<html>
<head>
<title>Exercici 5</title>
<script type="text/javascript">
window.onload=function(){
  document.getElementById("boton").onclick = mostrar;
}
function mostrar() {
  alert('Pulsado');
}
</script>
</head>
<body>
<form><input id="boton" type="button" value="Pulsar" /></form>
</body>
</html>
```

Flujo de eventos

- El flujo de eventos permite que varios elementos diferentes puedan responder a un mismo evento.
- Si en una página HTML se define un elemento <*tag1*> con otro en su interior <*tag2*> y este a su vez contiene otro <*tag3*>, cuando el usuario pulsa sobre <*tag3*>, el navegador permite asignar:
  - una función de respuesta al elemento <*tag3*>
  - otra función de respuesta al elemento <*tag2*>
  - otra función al elemento <*tag1*>
  - y finalmente, otra a la propia página
- De esta forma, un solo evento provocaría la respuesta de 4 elementos de la página (incluyendo la propia página).
- El orden en el que se ejecutan los eventos asignados a cada elemento de la página es lo que constituye el flujo de eventos.
- Existen diferencias en el flujo de eventos según el navegador utilizado.

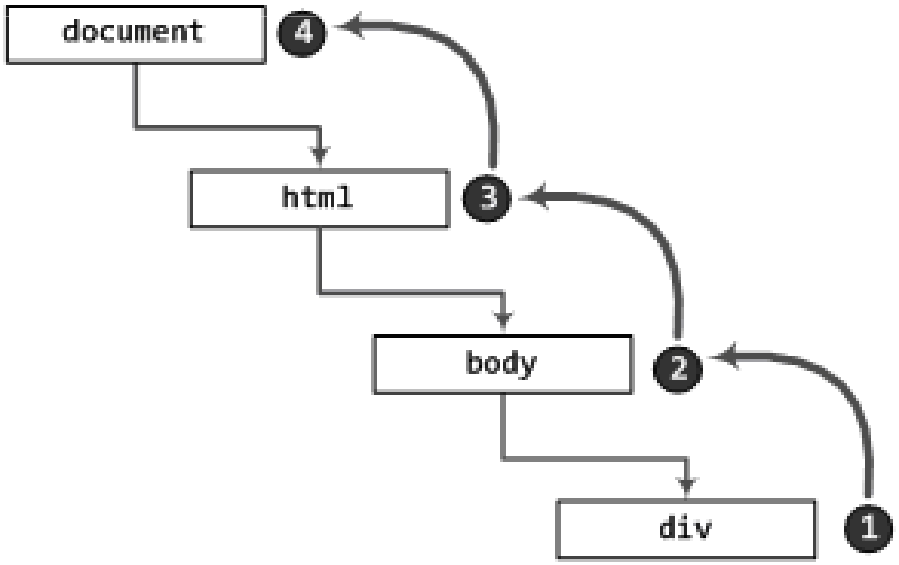
Propagación de eventos

- En este modelo de flujo de eventos, el orden que se sigue es desde el elemento más específico hasta el elemento menos específico.
- En los próximos ejemplos se emplea la siguiente página HTML:

```
<html onclick="handler()">
<head>
<title>Ejemplo de flujo de eventos</title>
</head>
<body onclick="handler()">
<div onclick="handler">Pincha aqui</div>
</body>
</html>
```

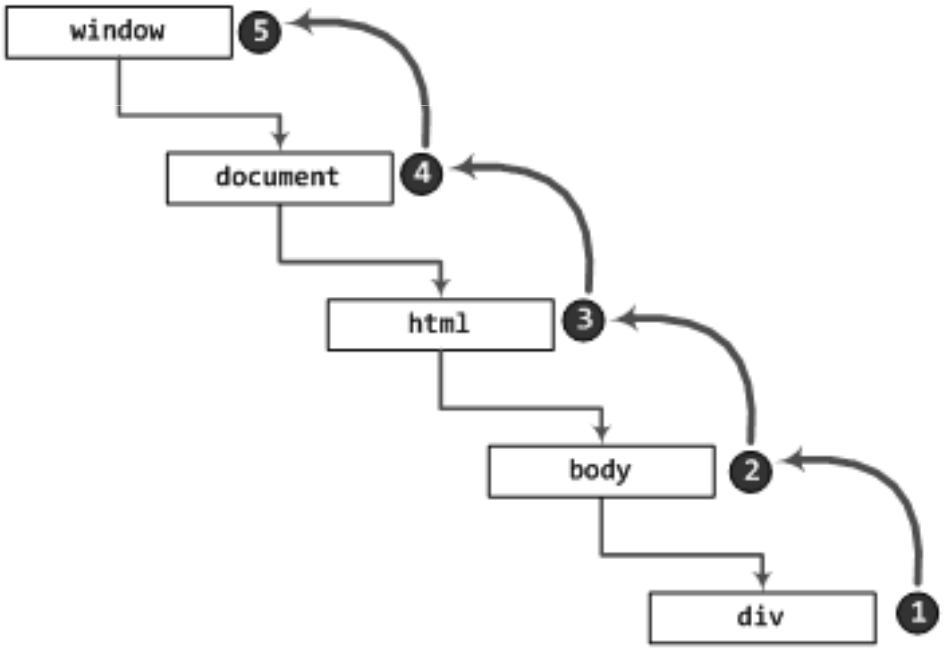
Explorer

- El primer evento que se tiene en cuenta es el generado por el que contiene el mensaje. A continuación el navegador recorre los ascendentes del elemento hasta que alcanza el nivel superior, que es el elemento *document*.
- Este modelo de flujo de eventos es el que incluye el navegador Internet Explorer.



Mozilla

- Los navegadores de la familia Mozilla (por ejemplo Firefox) también soportan este modelo, pero ligeramente modificado.
- El anterior ejemplo en un navegador de la familia Mozilla presenta el siguiente flujo de eventos:

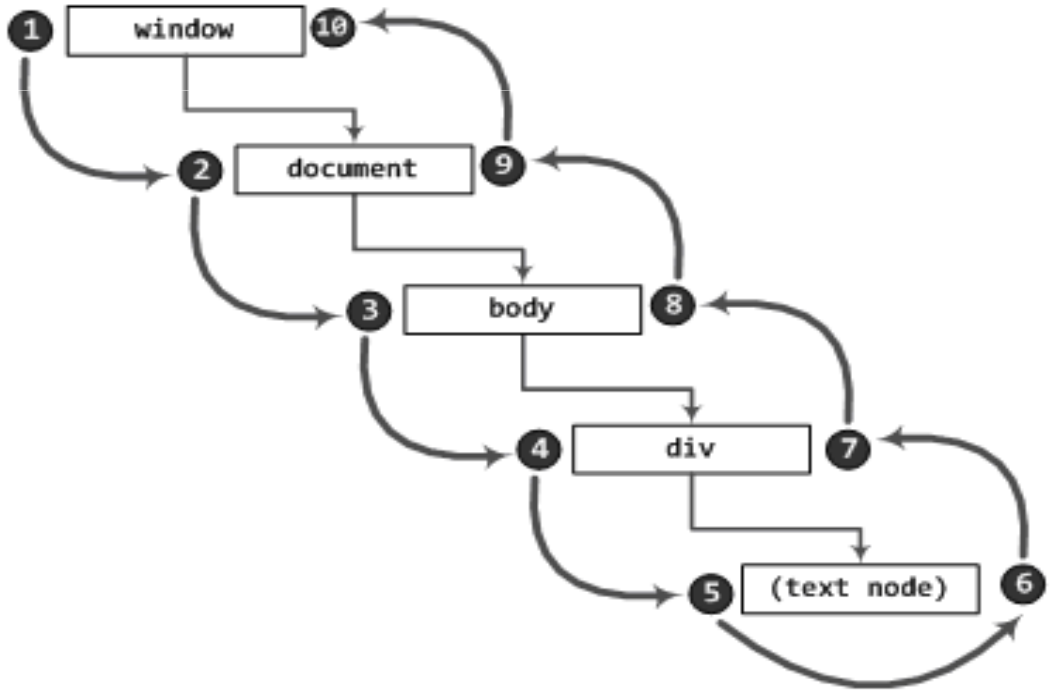


Eventos DOM (I)

- El flujo de eventos definido en la especificación DOM soporta:
  - Propagación de eventos (*event bubbling)*
  - Captura de eventos *(event capturing)*
    - En ese otro modelo, el flujo de eventos se define desde el elemento menos específico hasta el elemento más específico
- La captura de eventos se ejecuta en primer lugar.
- Los dos flujos de eventos recorren todos los objetos DOM desde el objeto *document* hasta el elemento más específico y viceversa.
- Además, la mayoría de navegadores que implementan los estándares, continúan el flujo hasta el objeto *window*.

Eventos DOM (II)

- El elemento más específico del flujo de eventos no es el que desencadena la ejecución de los eventos, sino el nodo de tipo *textNode* que contiene el .
- El hecho de combinar los dos flujos de eventos, provoca que el nodo más específico pueda ejecutar dos eventos de forma consecutiva.



Manejadores de eventos de IE

- La familia de navegadores de Internet Explorer emplean los métodos
  - *attachEvent*() → asociar manejadores de eventos
  - *detachEvent*() → desasociar manejadores de eventos.
- Todos los elementos y el objeto *window* disponen de estos métodos.
- Los métodos requieren dos parámetros:
  - nombre del evento que se quiere manejar
  - referencia a la función encargada de procesar el evento.
- El nombre del evento se debe indicar con el prefijo *on* incluido
  - `objeto.attachEvent(nom_controlador,funcion_handler)`
  - `objeto.detachEvent(nom_controlador,funcion_handler)`

Ejemplos de uso

```
function mostrar() {
  alert("Pulsado");
}
var Odiv = document.getElementById("div_tag");
Odiv.attachEvent("onclick", mostrar);
//Si queremos eliminar la asociación
ODiv.detachEvent("onclick", mostrar);
```

```
function mostrar1() {
  alert("También pulsado");
}
var Odiv = document.getElementById("div_tag");
Odiv.attachEvent("onclick", mostrar);
//Podemos asociar más de un manejador al mismo envento
Odiv.attachEvent("onclick", mostrar2);
//Ahora, al producirse el evento se ejecutan los dos manejadores
```

Manejadores de eventos de DOM

- La especificación DOM define otros dos métodos similares a los disponibles para Internet Explorer
  - *addEventListener()* → asociar manejadores de eventos
  - *removeEventListener()* → desasociar manejadores de eventos
- Los métodos requieren tres parámetros:
  - nombre del *"event listener"*
  - referencia a la función manejadora
  - tipo de flujo de eventos al que se aplica.
- El primer argumento es el nombre del controlador del evento sin el prefijo *on*.
- El tercer parámetro define el comportamiento del flujo de eventos. Si es:
  - *true* → Captura de eventos
  - *false* → Propagación de eventos.
- Si se asocia una función a un flujo de eventos determinado, esa función sólo se puede desasociar en el mismo tipo de flujo de eventos

Ejemplos de uso

```
function mostrar() {
  alert("Pulsado");
}
var Odiv = document.getElementById("div_tag");
Odiv.addEventListener("click", mostrar, false);
//Si queremos eliminar la asociación
Odiv.removeEventListener("click", mostrar, false);
```

```
function mostrar1() {
  alert("También pulsado");
}
var Odiv = document.getElementById("div_tag");
Odiv.addEventListener("click", mostrar, true);
//Podemos asociar más de un manejador al mismo envento
Odiv.addEventListener("click", mostrar2, true);
//Ahora, al producirse el evento se ejecutan los dos manejadores
```

Ejemplo 6 (I)

```
<html>
<head>
<title>Exemple 6</title>
<script type="text/javascript">
function cambiaImg1(){
  this.src = "blue.jpg";
}
function cambiaImg2(){
  this.src = "green.jpg";
}
function doble(){
  this.width*=2;
  this.height*=2;
}
function mitad(){
  this.width/=2;
  this.height/=2;
}
```

Ejemplo 6 (II)

```
function cambiaEvent(){
  var img=document.getElementById("imagen1");
   img.removeEventListener("mouseover",cambiaImg1,true);
   img.removeEventListener("mouseout",cambiaImg2,true);
   img.addEventListener("mouseover",doble,true);
   img.addEventListener("mouseout",mitad,true);
}
window.onload=function(){
  var img=document.getElementById("imagen1");
   img.addEventListener("mouseover",cambiaImg1,true);
   img.addEventListener("mouseout",cambiaImg2,true);
  document.addEventListener("click",cambiaEvent,true);
}
</script></head>
<body>
<img id="imagen1" src="blue.jpg" width="400" height="300" border="0">
</body>
</html>
```

El objeto *event*

- Cuando se produce un evento, la función que procesa el evento puede necesitar información relativa al evento producido:
  - la tecla que se ha pulsado,
  - la posición del ratón,
  - el elemento que ha producido el evento, etc.
- El objeto *event* es el mecanismo definido por los navegadores para proporcionar toda esa información. Se trata de un objeto que se crea automáticamente cuando se produce un evento y que se destruye de forma automática cuando se han ejecutado todas las funciones asignadas al evento.
- Internet Explorer permite el acceso al objeto *event* a través del objeto *window*:

```
ODiv.onclick = function() {
  var elEvento = window.event;
}
```

- El estándar DOM especifica que el objeto *event* es el único parámetro que se debe pasar a las funciones encargadas de procesar los eventos:

```
ODiv.onclick = function(evento) {
...
}
```

- Las propiedades de *event* difieren según el navegador IE o tipo Mozilla.

Propiedades de *event* en IE (I)

Propiedad/ Método	Devuelve	Descripción
`altKey`	Boolean	Devuelve `true` si se ha pulsado la tecla `ALT` y `false` en otro caso
`button`	Número entero	El botón del ratón que ha sido pulsado. Posibles valores: `0` – Ningún botón pulsado `1` – Se ha pulsado el botón izquierdo `2` – Se ha pulsado el botón derecho `3` – Se pulsan a la vez el botón izquierdo y el derecho `4` – Se ha pulsado el botón central `5` – Se pulsan a la vez el botón izquierdo y el central `6` – Se pulsan a la vez el botón derecho y el central `7` – Se pulsan a la vez los 3 botones
`cancelBubble`	Boolean	Si se establece un valor `true`, se detiene el flujo de eventos de tipo *bubbling*
`clientX`	Número entero	Coordenada X de la posición del ratón respecto del área visible de la ventana
`clientY`	Número entero	Coordenada Y de la posición del ratón respecto del área visible de la ventana
`ctrlKey`	Boolean	Devuelve `true` si se ha pulsado la tecla `CTRL` y `false` en otro caso
`fromElement`	Element	El elemento del que sale el ratón (para ciertos eventos de ratón)
`keyCode`	Número entero	En el evento `keypress`, indica el carácter de la tecla pulsada. En los eventos `keydown` y `keyup` indica el código numérico de la tecla pulsada
`offsetX`	Número entero	Coordenada X de la posición del ratón respecto del elemento que origina el evento

Propiedades de *event* en IE (II)

offsetY	Número entero	Coordenada Y de la posición del ratón respecto del elemento que origina el evento
repeat	Boolean	Devuelve true si se está produciendo el evento keydown de forma continuada y false en otro caso
returnValue	Boolean	Se emplea para cancelar la acción predefinida del evento
screenX	Número entero	Coordenada X de la posición del ratón respecto de la pantalla completa
screenY	Número entero	Coordenada Y de la posición del ratón respecto de la pantalla completa
shiftKey	Boolean	Devuelve true si se ha pulsado la tecla SHIFT y false en otro caso
srcElement	Element	El elemento que origina el evento
toElement	Element	El elemento al que entra el ratón (para ciertos eventos de ratón)
type	Cadena de texto	El nombre del evento
x	Número entero	Coordenada X de la posición del ratón respecto del elemento padre del elemento que origina el evento
y	Número entero	Coordenada Y de la posición del ratón respecto del elemento padre del elemento que origina el evento

Propiedades de *event* en DOM (I)

Propiedad/Método	Devuelve	Descripción
`altKey`	Boolean	Devuelve `true` si se ha pulsado la tecla ALT y `false` en otro caso
`bubbles`	Boolean	Indica si el evento pertenece al flujo de eventos de *bubbling*
`button`	Número entero	El botón del ratón que ha sido pulsado. Posibles valores: 0 – Ningún botón pulsado 1 – Se ha pulsado el botón izquierdo 2 – Se ha pulsado el botón derecho 3 – Se pulsan a la vez el botón izquierdo y el derecho 4 – Se ha pulsado el botón central 5 – Se pulsan a la vez el botón izquierdo y el central 6 – Se pulsan a la vez el botón derecho y el central 7 – Se pulsan a la vez los 3 botones
`cancelable`	Boolean	Indica si el evento se puede cancelar
`cancelBubble`	Boolean	Indica si se ha detenido el flujo de eventos de tipo *bubbling*
`charCode`	Número entero	El código unicode del carácter correspondiente a la tecla pulsada
`clientX`	Número entero	Coordenada X de la posición del ratón respecto del área visible de la ventana

Propiedades de *event* en DOM (II)

`clientY`	Número entero	Coordenada Y de la posición del ratón respecto del área visible de la ventana
`ctrlKey`	Boolean	Devuelve `true` si se ha pulsado la tecla `CTRL` y `false` en otro caso
`currentTarget`	Element	El elemento que es el objetivo del evento
`detail`	Número entero	El número de veces que se han pulsado los botones del ratón
`eventPhase`	Número entero	La fase a la que pertenece el evento: `0` – Fase capturing `1` – En el elemento destino `2` – Fase bubbling
`isChar`	Boolean	Indica si la tecla pulsada corresponde a un carácter
`keyCode`	Número entero	Indica el código numérico de la tecla pulsada
`metaKey`	Número entero	Devuelve `true` si se ha pulsado la tecla `META` y `false` en otro caso
`pageX`	Número entero	Coordenada X de la posición del ratón respecto de la página

Propiedades de *event* en DOM (III)

`pageY`	Número entero	Coordenada Y de la posición del ratón respecto de la página
`preventDefault()`	Función	Se emplea para cancelar la acción predefinida del evento
`relatedTarget`	Element	El elemento que es el objetivo secundario del evento (relacionado con los eventos de ratón)
`screenX`	Número entero	Coordenada X de la posición del ratón respecto de la pantalla completa
`screenY`	Número entero	Coordenada Y de la posición del ratón respecto de la pantalla completa
`shiftKey`	Boolean	Devuelve `true` si se ha pulsado la tecla `SHIFT` y `false` en otro caso
`stopPropagation()`	Función	Se emplea para detener el flujo de eventos de tipo *bubbling*
`target`	Element	El elemento que origina el evento
`timeStamp`	Número	La fecha y hora en la que se ha producido el evento
`type`	Cadena de texto	El nombre del evento

IE versus DOM

- En Internet Explorer, todas las propiedades salvo *repeat* son de lectura/escritura.
- En DOM, la mayoría de propiedades del objeto *event* son de sólo lectura. Solamente *altKey*, *button* y *keyCode* son de lectura y escritura.
- La tecla *META* es una tecla especial que se encuentra en algunos teclados de ordenadores muy antiguos. Actualmente, en los ordenadores tipo PC se asimila a la tecla *Alt* o a la *tecla de Windows,* mientras que en los ordenadores tipo Mac se asimila a la tecla *Command.*

Ejemplo 7 (I)

```
<html>
<head>
<title>Exemple 6</title>
<script type="text/javascript">
function cambiaImg(evento){
  if(evento.type=="mouseover")
    this.src = "green.jpg";
   else if(evento.type=="mouseout")
    this.src = "blue.jpg";
}
function posicion(evento){
  alert("( "+evento.screenX+" , "+evento.screenY+" )");
}
```

Ejemplo 7 (II)

```
window.onload=function(){
  var img=document.getElementById("imagen1");
   img.addEventListener("mouseover",cambiaImg,true);
   img.addEventListener("mouseout",cambiaImg,true);
   document.addEventListener("click",posicion,true);
}
</script>
</head>
<body>
<img id="imagen1" src="blue.jpg" width="400" height="300"
   border="0">
</body>
</html>
```