JavaScript Avançat: Esdeveniments.

IES Son Ferrer

JS Avançat: Esdeveniments

Les aplicacions web basades en JS poden utilitzar un **model de programació basada en esdeveniments**. Aquí **s'espera a què l'usuari faci "alguna cosa" i en aquest moment un script respon a aquesta acció** processant la informació i generant un resultat.

En la programació orientada a esdeveniments, les aplicacions esperen que es produeixin els esdeveniments. Aquesta forma de programar és la meś utilitzada en els nostres temps.

Per tant, JS permet realitzar scripts amb dos mètodes de programació: seqüencial i basada en esdeveniments.

JS Avançat: Esdeveniments

Exemples d'esdeveniments: pulsar una tecla, moure el ratol·lí o fer clic amb ell, seleccionar un element d'un formulari o sortir d'un d'ells, redimensionar la finestra del navegador, etc., etc., etc.

JS permet assignar una funció a cadascun dels esdeveniments. D'aquesta forma, quan es produeix qualsevol esdeveniment, JS executa la seva funció associada. Aquest tipus de funcions es denominen "***event handlers***" ("manejadors d'esdeveniments").

Model d’esdeveniments

El nivell 1 de DOM no inclou especificacions relatives als esdeveniments JS. El nivell 2 de DOM inclou certs aspectes relacionats amb els esdeveniments i el **nivell 3 de DOM inclou l'especificació completa dels esdeveniments de JS**. Desafortunadament, l'especificació de nivell 3 de DOM **es va publicar l'any 2004**, més de deu anys després que els primers navegadors incloguessin els esdeveniments.

Sempre ha estat complicat programar aplicacions web per les **incompatibilitats entre navegadors**.

Una de les incompatibilitats més importants es dóna precisament en el model d’esdeveniments del navegador.

Models bàsic d'esdeveniments.

Cada element (o etiqueta) té la seva pròpia llista de possibles esdeveniments.

Diferents etiquetes poden tenir el mateix esdeveniment, òbviament.

Els esdeveniments solen tenir un nom començat per `on`, seguit pel nom en anglès de l'acció associada a l'esdeveniment. Exemples:

- Fer clic sobre un element amb el ratol·lí: `onclick`
- Moure el ratol·lí sobre un element: `onmousemove`

Models bàsic d'esdeveniments.

Esdeveniment	Descripció	Elements per als quals està definit
`onblur`	Element perd el focus	`<button>`, `<input>`, `<label>`, `<select>`, `<textarea>`, `<body>`
`onchange`	Es modifica l'element	`<input>`, `<select>`, `<textarea>`
`onclick`	Fer clic amb el ratolí	Tots els elements
`ondblclick`	Fer doble clic amb el ratolí	Tots els elements
`onfocus`	Element guanya el focus	`<button>`, `<input>`, `<label>`, `<select>`, `<textarea>`, `<body>`

Models bàsic d'esdeveniments.

Esdeveniment	Descripció	Elements per als quals està definit
`onkeydown`	Prémer una tecla (sense deixar anar)	Elements de formulari i `<body>`
`onkeypress`	Prémer una tecla	Elements de formulari i `<body>`
`onkeyup`	Deixar anar una tecla clicada	Elements de formulari i `<body>`
`onload`	La pàgina s'ha carregat completament	`<body>`
`onmousedown`	Prémer (sense deixar anar) un botó del ratolí	Tots els elements

Models bàsic d'esdeveniments.

Esdeveniment	Descripció	Elements per als quals està definit
`onmousemove`	Moure el ratolí	Tots els elements
`onmouseover`	El ratolí *"entra"* en l'element (passa per sobre de l'element)	Tots els elements
`onmouseout`	El ratolí *"surt"* de l'element (passa per sobre d'un altre element)	Tots els elements
`onmouseup`	Deixar anar el botó que estava espitjat al ratolí	Tots els elements

Models bàsic d'esdeveniments.

Esdeveniment	Descripció	Elements per als quals està definit
`onreset`	Inicialitzar el formulari (esborrar totes les seves dades)	`<form>`
`onresize`	S'ha modificat la mida de la finestra del navegador	`<body>`
`onselect`	Seleccionar un text	`<input>`, `<textarea>`
`onsubmit`	Enviar el formulari	`<form>`
`onunload`	S'abandona la pàgina (per exemple al tancar el navegador)	`<body>`

Models bàsic d'esdeveniments.

Esdeveniments més utilitzats: `onload`, `onclick`, `onmouseover`, `onmouseout` i `onsubmit`.

Alguns dels esdeveniments anteriors (`onclick`, `onkeydown`, `onkeypress`, `onreset` i `onsubmit`) permeten evitar el comportament per defecte de l'esdeveniment si es retorna el valor `false`, tal i com es veurà més envant.

Les accions típiques que realitza un usuari en una pàgina web poden donar lloc a una successió d'esdeveniments. En fer clic per exemple sobre un botó de tipus `<input type="submit">` es desencadenen els esdeveniments `onmousedown`, `onclick`, `onmouseup` i `onsubmit` de forma consecutiva.

Models bàsic d’esdeveniments.

Un esdeveniment de JS per si mateix no té utilitat. **Perquè els esdeveniments resultin útils, s'han d'associar funcions o codi JS**: quan es produeix un esdeveniment s'executa el codi indicat per respondre a l’esdeveniment que ha succeït.

Les funcions o codi JS que es defineixen per a cada esdeveniment es denominen "***event handlers***" (manejadors d’esdeveniments). Hi ha diverses formes diferents d'indicar els controladors:

Manejadors com atributs dels elements HTML.

Manejadors com funcions JS externes.

Manejadors "semàntics".

Models bàsic d'esdeveniments.

Manejadors com atributs dels elements HTML.

En aquest cas, el codi s'inclou en un atribut del propi element HTML.

```
<input type="button" value="Clica'm i veuràs" onclick="alert('Gràcies per
  clicar');" />
<div onclick="alert('Has clicat amb el ratolí');"
  onmouseover="alert('Acabes de passar el ratolí per damunt');">
    Pots clicar sobre aquest element o simplement passar el ratolí per
  damunt
</div>
<body onload="alert('La pàgina s'ha carregat completament');">
    ...
</body>
```

Recordeu que per accedir al DOM s'ha d'haver carregat la pàgina completament...

Models bàsic d’esdeveniments.

Manejadors d’esdeveniments i variable `this`.

JS defineix una variable especial anomenada `this` que es crea automàticament i que s'empra en algunes tècniques avançades de programació. En els esdeveniments s’utilitza per referir-se al element HTML que ha provocat l'esdeveniment.

Models bàsic d'esdeveniments.

Manejadors d'esdeveniments i variable this.

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

Models bàsic d’esdeveniments.

Manejadors com funcions JS externes.

Si es realitzen aplicacions complexes, com ara la validació d'un formulari, és aconsellable agrupar tot el codi JS en una funció externa i cridar a aquesta funció des de l'element HTML.

```
function mostraMissatge() {
     alert('Gràcies per clicar');
}

<input type="button" value="Clica’m i veuràs"
   onclick="mostraMissatge()" />
```

Models bàsic d'esdeveniments.

Manejadors com funcions JS externes.

El principal inconvenient d'aquest mètode és que en les funcions externes no es pot seguir utilitzant la variable `this` i per tant, cal passar aquesta variable com a paràmetre a la funció:

```
<div style="width:150px; height:60px; border:thin solid silver"
   onmouseover="ressalta(this)" onmouseout="ressalta(this)">
     Secció de continguts ...
</div>

function ressalta(element) {
    if (element.style.borderColor === 'silver')
        element.style.borderColor = 'black';
    else
        element.style.borderColor = 'silver';
}
```

Models bàsic d’esdeveniments.

Manejadors "semàntics".

Els mètodes que s'han vist per afegir controladors d'esdeveniments (com atributs HTML i com a funcions externes) tenen un greu inconvenient: **"embruten" el codi HTML** de la pàgina.

Com és conegut, una de les bones pràctiques bàsiques en el disseny de pàgines i aplicacions web és la **separació dels continguts (HTML) i el seu aspecte o presentació (CSS)**. Sempre que sigui possible, també es recomana separar els continguts (HTML) i el seu comportament o programació (JS).

Models bàsic d'esdeveniments.

Manejadors "semàntics".

Hi ha un mètode alternatiu per definir els *event handlers* de JS. Aquesta tècnica és una evolució del mètode de les funcions externes, ja que es basa en utilitzar les propietats DOM dels elements HTML.

```
<input id="clicable" type="button" value="Clica'm i veuràs"
   onclick="alert('Gràcies per clicar');" />
```

Es pot transformar en:

```
// Element HTML
   <input id="clicable" type="button" value="Clica'm i veuràs" />
// Funció externa
function mostraMissatge() {
     alert('Gràcies per clicar');
}
// Assignar la funció externa a l'element
document.getElementById("clicable").onclick = mostraMissatge;
```

Models bàsic d’esdeveniments.

Manejadors "semàntics".

Anau alerta perquè un error típic és el següent:

```
/* Assignar una funció externa a un esdeveniment d’un element */
  document.getElementById("clicable").onclick = mostraMissatge;
/* Executar una funció i guardar el seu resultat en una
  propietat d’un element */
document.getElementById("clicable").onclick = mostraMissatge();
```

Models bàsic d’esdeveniments.

Manejadors "semàntics".

L'únic inconvenient d'aquest mètode és que la pàgina s'ha de carregar completament abans que es puguin utilitzar les funcions DOM que assignen els controladors als elements HTML:

```
window.onload = function() {
      document.getElementById("clicable").onclick = mostraMissatge;
}
```

La tècnica anterior utilitza el concepte de **funcions anònimes**:

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

Models bàsic d'esdeveniments.

Manejadors "semàntics".

En el següent exemple, s'afegeixen esdeveniments als elements de tipus `input=text` d'un formulari complex:

```
function ressalta() {
    // Codi JavaScript
}
window.onload = function() {
    let formulari = document.getElementById("formulari");
    let campsInput = formulari.getElementsByTagName("input");
    for (let i=0; i<campsInput.length; i++) {
        if (campsInput[i].type == "text") {
            campsInput[i].onclick = ressalta;
        }
    }
}
```

Models bàsic d’esdeveniments.

Manejadors "semàntics".

Un altre avantatge addicional d'aquesta tècnica és que les funcions externes poden utilitzar la variable `this` referida a l'element que origina l'esdeveniment.

```
<input id="clicable" type="button" value="Clica’m i veuràs" />
// Funció externa
function mostraMissatge() {
    alert(this.value);
}
// Assignar la funció externa a l’element
document.getElementById("clicable").onclick = mostraMissatge;
```

Flux d’esdeveniments.

A més dels esdeveniments bàsics que s'han vist, els navegadors inclouen un mecanisme relacionat anomenat **flux d'esdeveniments o "*event flow*"**. El flux d'esdeveniments **permet que diversos elements diferents puguin respondre a un mateix esdeveniment**.

Si en una pàgina HTML es defineix un element `<div>` amb un botó al seu interior, quan l'usuari fa clic sobre el botó, el navegador permet assignar una funció de resposta al botó, una altra funció de resposta al `<div>` que el conté i una altra funció de resposta a la pàgina completa. D'aquesta manera, un sol esdeveniment (la pulsació d'un botó) provoca la resposta de tres elements de la pàgina (incloent la pròpia pàgina).

Flux d’esdeveniments.

L'ordre en què s'executen els esdeveniments assignats a cada element de la pàgina és el que constitueix el flux d'esdeveniments.

Existeixen dos models de flux d’esdeveniments:

Event bubbling.

Event capturing.

Flux d’esdeveniments. *Event bubbling*.

En aquest model de flux d'esdeveniments, **l'ordre que se segueix és des de l'element més específic fins l'element menys específic**.

En els exemples que venen s'empra la següent pàgina HTML:

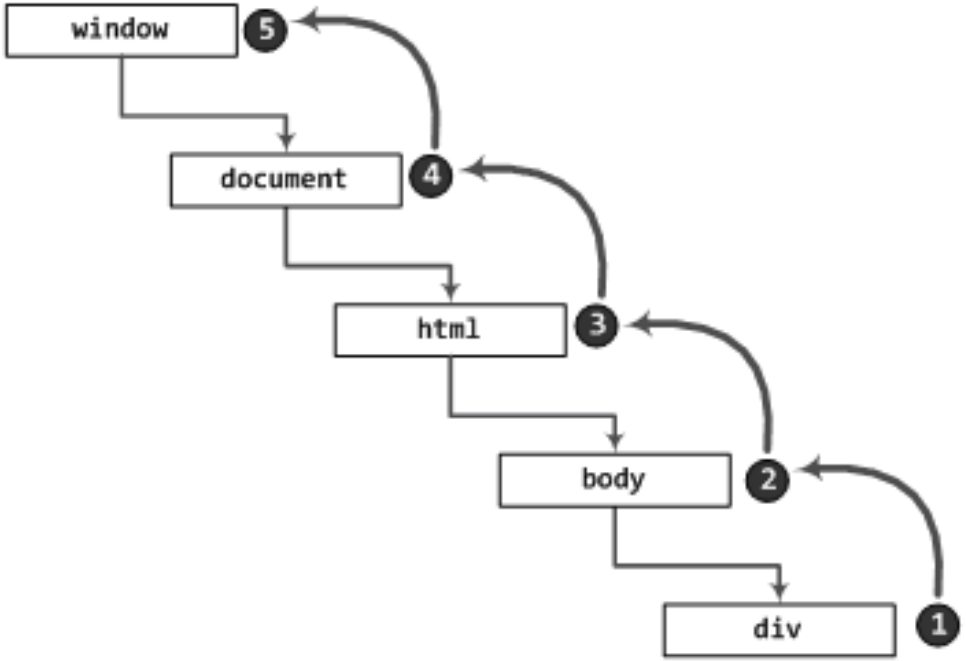
```
<html onclick="processaEsdeveniment()">
      <head><title>Exemple de flux d’esdeveniments</title></head>
      <body onclick="processaEsdeveniment()">
          <div onclick="processaEsdeveniment()">Clica aquí</div>
      </body>
</html>
```

Flux d’esdeveniments. *Event bubbling*.

```
<!DOCTYPE html>
<html onclick="processaEsdeveniment(this)">
    <head>
        <title>Exemple de flux d’esdeveniments</title>
        <script>
            function procesaEvento(objecte) {
                alert(objecte.tagName);
            }
        </script>
    </head>

    <body onclick="processaEsdeveniment(this)">
      <div onclick="processaEsdeveniment(this)">Clica aquí</div>
    </body>
</html>
```

Flux d’esdeveniments. *Event bubbling*.

Quan es fa clic sobre el text "Clica aquí" que es troba dins del `<div>`, s'executen els següents esdeveniments en l'ordre que mostra el següent esquema:



Flux d’esdeveniments. *Event capturing*.

Hi ha una altra fase del processament d'esdeveniments anomenada "captura". Poques vegades s'utilitza en codi real, però de vegades pot ser útil.

L'estàndard d’esdeveniments del DOM descriu tres fases de propagació d'esdeveniments:

- Fase de captura: l'esdeveniment se propaga cap abaix, cap a l'element de destí.
- Fase de destinació: l'esdeveniment arriba a l'element.
- Fase de bombolles: l'esdeveniment se propaga cap amunt des de l’element destí.

Esdeveniments. *Handlers* i *listeners*.

Hem vist el que és un "*event handler*" o controlador d'esdeveniments (funcions que responen als esdeveniments que es produeixen). També hem vist 3 formes de definir-los:

1. Codi JS dins d'un atribut del propi element HTML.
2. Definició de l'esdeveniment en el propi element HTML però el *handler* (“manejador”) és una funció externa.
3. Manejadors semàntics assignats mitjançant DOM sense necessitat de modificar el codi HTML de la pàgina.

Qualsevol d'aquests tres models funciona correctament en tots els navegadors disponibles en l'actualitat. La forma d'assignar i "desassignar" *event handlers* múltiples és igual en tots els navegadors, fins i tot el IE/Edge des de la versió 9 (ja que abans utilitzava uns altres mètodes)

Esdeveniments. *Handlers* i *listeners*.

Amb aquestes 3 formes de definir els "*event handler*" no tenim possibilitat d'executar-los des de la fase de captura. Per això necessitam 2 mètodes nous del DOM que ens ho permeten fer ...

Esdeveniments. *Event Handlers* de DOM.

L'especificació DOM defineix dos mètodes denominats `addEventListener()` i `removeEventListener()` per associar i desassociar controladors d'esdeveniments.

Es requereixen tres paràmetres: el nom del "*event listener*", una referència a la funció encarregada de processar l'esdeveniment i el tipus de flux d'esdeveniments al que s'aplica (*capturing* o *bubbling*).

El primer argument no és el nom complet de l'esdeveniment com hem fet fins ara, sinó que s'ha d'eliminar el prefix on. És a dir, per a l'esdeveniment `onclick`, ara s'ha d'utilitzar `click`.

Si el tercer paràmetre és `true`, el controlador es fa servir des de la fase de ***capture***. Si és `false` el controlador s'associa a la fase de ***bubbling***.

Esdeveniments. *Event Handlers* de DOM.

```
function mostraMissatge() {
    alert("Has clicat el ratolí");
}
function mostraAltreMissatge() {
    alert("Has clicat el ratolí i per això se mostren aquests
  missatge");
}
  let elDiv = document.getElementById("div_principal");
  elDiv.addEventListener("click", mostraMissatge, true);
  elDiv.addEventListener("click", mostraAltreMissatge, true);
```

Nom de l'esdeveniment

Funció que processa l'esdeveniment

true -> fase de *capture*
false -> fase de *bubbling*

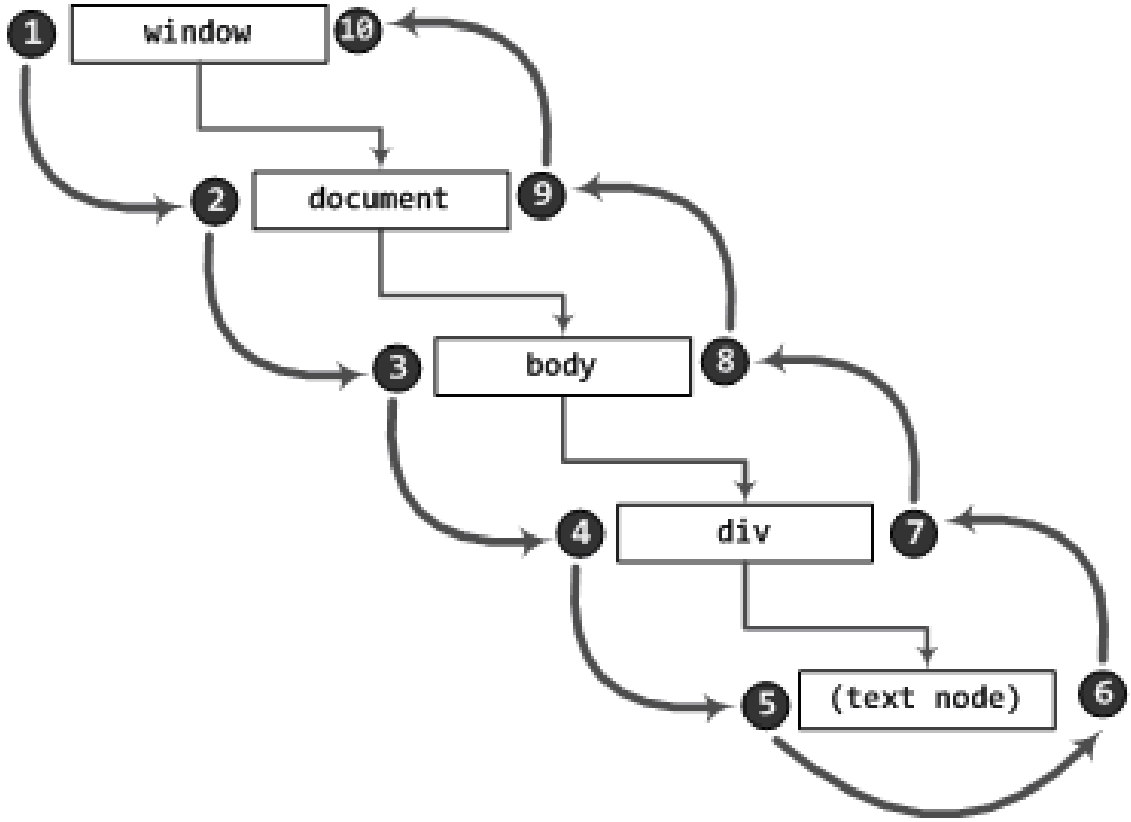
Exemple amb les 3 fases

```
<body>
    <form>FORM
      <div>DIV
        <p>P</p>
      </div>
    </form>

    <script>
      for (let elem of document.querySelectorAll('*')) {
        elem.addEventListener("click", capturar, true);
        elem.addEventListener("click", bombollejar);
      }

      function capturar(){
          alert(`Capturant ${this.tagName}`)
      }

      function bombollejar(elem){
          alert(`Bombollejant ${this.tagName}`)
      }
    </script>
</body>
```

Si feu clic a <p>, la seqüència serà:

1. HTML → BODY → FORM → DIV (fase de captura, el primer *listener*)
2. P (fase objectiu, es desencadena dues vegades, ja que hem definit dos *listener*: captura i bombolles)
3. DIV → FORM → BODY → HTML (fase de bombolles, el segon *listener*).

Exemple amb les 3 fases



Esdeveniments. *Event Handlers* de DOM.

Si s'associa una funció a un flux d'esdeveniments determinat, aquesta funció només es pot desvincular en el mateix tipus de flux d'esdeveniments. Si es considera el següent exemple:

```
function mostraMissatge() {
     alert("Has clicat el ratolí");
}
let elDiv = document.getElementById("div_principal");
elDiv.addEventListener("click", mostraMissatge, false);
// Més endavant se decideix desassociar la funció de l'esdeveniment
elDiv.removeEventListener("click", mostraMissatge, true);
```

La darrera instrucció intenta desvincular la funció `mostraMissatge` en el flux d'esdeveniments de *capture*, mentre que a l'associar-la, es va indicar el flux d'esdeveniments de *bubbling*. Encara que l'execució de l'aplicació no s'atura i no es produeix cap error, **la darrera instrucció no té cap efecte**.

Tampoc és possible desvincular funcions anònimes associades a esdeveniments.

Esdeveniments. L'objecte *event*.

Normalment, la **funció que processa l'esdeveniment necessita informació relativa a l'esdeveniment produït**: la tecla que s'ha clicat, la posició del ratolí, l'element que s'ha produït l'esdeveniment, etc.

L'**objecte** `event` és el mecanisme definit pels navegadors per p**roporcionar tota aquesta informació**.

Esdeveniments. L’objecte *event*.

Quan un esdeveniment es produeix en HTML, l'esdeveniment pertany a un determinat objecte d'esdeveniment. Per exemple un esdeveniment del clic del ratolí pertany a l'objecte MouseEvent.

Tots els objectes d'esdeveniment es basen en l'objecte Event i hereten totes les seves propietats i mètodes

El funcionament dels navegadors que segueixen els estàndards pot semblar "màgic", ja que en la declaració de la funció que gestionarà l’esdeveniment no fa falta passar cap paràmetre. En realitat, **els navegadors** que segueixen els estàndards **creen automàticament aquest paràmetre i el passen sempre a la funció encarregada de gestionar l'esdeveniment**.

Esdeveniments. Propietats de *event* per DOM.

- Una llista dels Event Object més freqüents la podem veure a https://www.w3schools.com/jsref/obj_events.asp
- I en quant als esdeveniments del DOM, així com les propietats i mètodes que tenen segons els tipus d'Event Object https://www.w3schools.com/jsref/dom_obj_event.asp

Esdeveniments. Propietats de *event* per DOM.

Per exemple, la propietat `type` serveix per obtenir el tipus d'esdeveniment:

```
function processaEsdeveniment() {
     if(event.type == "click") {
       alert("Has clicat el ratolí");
     }
     else if(event.type == "mouseover") {
       alert("Has mogut el ratolí");
     }
}
elDiv.onclick = processaEsdeveniment;
elDiv.onmouseover = processaEsdeveniment;
```

Mentre que el gestor de l'esdeveniment inclou el prefix `on` en el seu nom, el tipus d'esdeveniment retornat per la propietat `type` prescindeix d'aquest prefix. Per això en l'exemple anterior es compara el seu valor amb `click` i `mouseover` i no amb `onclick` i `onmouseover` .

Esdeveniments. Propietats de *event* per DOM.

La propietat `keyCode` per obtenir el codi corresponent al caràcter de la tecla que s'ha clicat. La tecla pitjada no sempre representa un caràcter alfanumèric. P. e., quan es clica la tecla `ENTER` s'obté el codi `13`. Barra d'espai codi `32` i tecla d'esborrat codi `8`.

Una forma més immediata de comprovar si s'han clicat algunes tecles especials, és utilitzar les propietats `shiftKey` , `altKey` i `ctrlKey`.

Per obtenir la posició del ratolí respecte de la part visible de la finestra, s'empren les propietats `clientX` i `clientY`. De la mateixa manera, per obtenir la posició del punter del ratolí respecte de la pantalla completa, s'empren les propietats `screenX` i `screenY`.

Esdeveniments. Propietats de *event* per DOM.

Forma en què s'obté l'element que origina l'esdeveniment:

```
// Navegadors que segueixen els estàndards
let objectiu = event.target;
```

Esdeveniments. Propietats de *event* per DOM.

Una de les propietats més interessants és la possibilitat d'impedir que es completi el comportament normal d'un esdeveniment. En altres paraules, amb JS és possible no mostrar cap caràcter quan es prem una tecla, no enviar un formulari després de prémer el botó d'enviament, no carregar cap pàgina en prémer un enllaç, etc.

```
event.returnValue = false;
// També ho podem fer amb el mèdode preventDefault
event.preventDefault();
```

En el model básic d'esdeveniments també és possible impedir el comportament per defecte d'alguns esdeveniments. Si per exemple en un element `<textarea>` s'utilitza el següent manejador d'esdeveniments:

```
<textarea onkeypress="return false;"></textarea>
```

Esdeveniments. Propietats de *event* per DOM.

L'objecte `event` també permet aturar completament l'execució del flux normal d'esdeveniments:

```
event.cancelBubble = true;

// el mateix amb el mètode stopPropagation
event.stopPropagation();
```

En aturar el flux d'esdeveniments pendents, s'invaliden i no s'executen els esdeveniments que resten des d'aquest moment fins que es recorren tots els elements pendents fins a l'element `window`.

Esdeveniments. Tipus d'esdeveniments.

Tradicionalment l'especificació de DOM definia 4 grans grups d'esdeveniments:

- **Esdeveniments de ratolí**: s'originen quan l'usuari fa servir el ratolí per realitzar algunes accions.
- **Esdeveniments de teclat**: s'originen quan l'usuari prem sobre qualsevol tecla del teclat.
- **Esdeveniments HTML**: s'originen quan es produeixen canvis en la finestra del navegador o quan es produeixen certes interaccions entre el client i el servidor.
- **Esdeveniments DOM**: s'originen quan es produeix un canvi en l'estructura DOM de la pàgina. També es denominen "esdeveniments de mutació".

Actualment aquesta classificació s'ha afinat molt més, com podem veure a https://www.w3schools.com/jsref/obj_events.asp

Tipus d'esdeveniments. Esdeveniments de ratolí.

Esdeveniment	Descripció
`click`	Es produeix quan es fa clic en el botó esquerre del ratolí. També es produeix quan el focus de l'aplicació està situat en un botó i es prem la tecla `ENTER`
`dblclick`	Es produeix quan es fa clic dues vegades el botó esquerre del ratolí
`mousedown`	Es produeix quan es fa clic qualsevol botó del ratolí
`mouseup`	Es produeix quan es deixa anar qualsevol botó del ratolí que hagi estat clicat
`mousemove`	Es produeix (de forma contínua) quan el punter del ratolí es troba sobre un element
`mouseover`	Es produeix quan el punter del ratolí es troba fora d'un element i l'usuari mou el punter cap a un lloc a l'interior de l'element
`mouseout`	Es produeix quan el punter del ratolí es troba a l'interior d'un element i l'usuari mou el punter a un lloc fora d'aquest element

NOTA: Els esdeveniments que es produeixen quan el ratolí interactua amb el document HTML pertanyen a l'objecte **MouseEvent** (https://www.w3schools.com/jsref/obj_mouseevent.asp)

Tipus d'esdeveniments. Esdeveniments de ratolí.

L'objecte `event` conté les següents propietats per als esdeveniments de ratolí:

- Les **coordenades del ratolí** (totes les coordenades diferents relatives als diferents elements)
- La propietat `type`
- La propietat `target` (DOM)
- Les propietats `shiftKey` , `ctrlKey` , `altKey` i `metaKey` (només DOM)
- La propietat `button` (només en els esdeveniments `mousedown` , `mousemove` , `mouseout` ,`mouseover` i `mouseup`)

Tipus d’esdeveniments. Esdeveniments de ratolí.

En els navegadors que suporten l'estàndard DOM, només hi ha una propietat denominada `relatedTarget` . En l'esdeveniment `mouseout` , `relatedTarget` apunta a l'element al qual s'ha mogut el ratolí. En l'esdeveniment `mouseover` , `relatedTarget` apunta a l'element des del qual s'ha mogut el punter del ratolí.

Tipus d’esdeveniments. Esdeveniments de ratolí.

Quan es prem un botó del ratolí, la seqüència d'esdeveniments que es produeix és la següent: `mousedown` , `mouseup` , `click` . Per tant, la seqüència d'esdeveniments necessària per arribar al doble clic arriba a ser tan complexa com la següent: `mousedown` , `mouseup` , `click` ,`mousedown` , `mouseup` , `click` , `dblclick` .

Tipus d’esdeveniments. Esdeveniments de teclat.

Els esdeveniments que es produeixen quan l'usuari prem una tecla al teclat, pertanyen a l'objecte **KeyboardEvent**.

Esdeveniment	Descripció
`keydown`	Es produeix quan es prem qualsevol tecla del teclat. També es produeix de forma contínua si es manté clicada la tecla
`keypress`	Es produeix quan es prem una tecla corresponent a un caràcter alfanumèric (no es tenen en compte tecles com a `SHIFT` , `ALT` , etc.). També es produeix de forma contínua si es manté clicada la tecla
`keyup`	Es produeix quan es deixa anar qualsevol tecla premuda

Tipus d’esdeveniments. Esdeveniments de teclat.

L'objecte `event` conté les següents propietats per als esdeveniments de teclat:

La propietat `keyCode` per obtenir el valor Unicode de la tecla premuda

La propietat `key` (per obtenir el botó del teclat que s’ha clicat)

La propietat `target` (per obtenir una referència a l'objecte sobre el que s’ha produït l’esdeveniment)

Les propietats `shiftKey` , `ctrlKey` , `altKey` i `metaKey` (per saber si s’ha clicat la tecla «SHIFT», «CTRL», «ALT» o «META» respectivament)

Tipus d'esdeveniments. Esdeveniments de teclat.

Quan es prem una tecla corresponent a un caràcter alfanumèric, es produeix la següent seqüència d'esdeveniments: `keydown` , `keypress` , `keyup` .

- `Keypress` no detecta totes les tecles (alt, ctrl, shift, esc, ...).
- `Keydown` sí que es llaça per a qualsevol tecla.

Quan es prem un altre tipus de tecla, es produeix la següent seqüència d'esdeveniments: `keydown` , `keyup` .

Si es manté premuda la tecla, en el primer cas es repeteixen de forma contínua els esdeveniments `keydown` i `keypress` i en el segon cas, es repeteix l'esdeveniment `keydown` de forma contínua.

Tipus d’esdeveniments. Esdeveniments HTML.

Esdeveniment	Descripció
`load`	Es produeix en l'objecte `window` quan la pàgina es carrega del tot. En l'element `<img>` quan es carrega per complet la imatge. En l'element `<object>` quan es carrega l'objecte
`unload`	Es produeix en l'objecte `window` quan la pàgina desapareix per complet (en tancar la finestra del navegador per exemple). En l'element `<object>` quan desapareix l'objecte.
`abort`	Es produeix en un element `<object>` quan l'usuari deté la descàrrega de l'element abans que hagi acabat
`error`	Es produeix en l'objecte `window` quan es produeix un error de JavaScript. En l'element `<img>` quan la imatge no s'ha pogut carregar per complet i en l'element `<object>` quan l'element no es carrega correctament
`select`	Es produeix quan se seleccionen diversos caràcters d'un quadre de text (`<input>` i `<textarea>`)

Tipus d’esdeveniments. Esdeveniments HTML.

Esdeveniment	Descripció
`change`	Es produeix quan un quadre de text (`<input>` i `<textarea>`) perd el focus i el seu contingut ha variat. També es produeix quan varia el valor d'un element `<select>`
`submit`	Es produeix quan es prem sobre un botó de tipus submit (`<input type="submit">`)
`reset`	Es produeix quan es prem sobre un botó de tipus reset (`<input type="reset">`)
`resize`	Es produeix en l'objecte `window` quan es redimensiona la finestra del navegador
`scroll`	Es produeix en qualsevol element que tingui una barra de desplaçament, quan l'usuari la fa servir. L'element `<body>` conté la barra de desplaçament de la pàgina completa
`focus`	Es produeix en qualsevol element (inclòs l'objecte `window`) quan l'element obté el focus
`blur`	Es produeix en qualsevol element (inclòs l'objecte `window`) quan l'element perd el focus

Tipus d’esdeveniments. Esdeveniments HTML.

Un dels esdeveniments més utilitzats és l'esdeveniment `load` , ja que totes les manipulacions que es realitzen mitjançant DOM requereixen que la pàgina estigui carregada del tot i per tant, l'arbre DOM s'hagi construït completament.

Tots els elements del DOM tenen les propietats `scrollLeft` i `scrollTop` que es poden emprar juntament amb l'esdeveniment `scroll`.

Tipus d’esdeveniments. Esdeveniments DOM.

Les diferències existents entre els navegadors disponibles en l'actualitat poden complicar el desenvolupament d'aplicacions compatibles amb tots els navegadors, anomenades aplicacions ***"cross browser"*** en anglès.

Tipus d’esdeveniments. Esdeveniments DOM.

jQuery, **Prototype** i **MooTools**, per exemple, són biblioteques de Javascript utilitzades per molts de desenvolupadors que ja han solucionat nombrosos problemes de consistència Cross-Browser. Així que pot ser una bona idea utilitzar algun d'aquests *frameworks* per a desenvolupar les vostres aplicacions Web.