# L'objet Event

## Généralités sur l'objet Event

Tout d'abord, à quoi sert cet objet ? À vous fournir une multitude d'informations sur l'événement actuellement déclenché. Par exemple, vous pouvez récupérer quelles sont les touches actuellement enfoncées, les coordonnées du curseur, l'élément qui a déclenché l'événement… Les possibilités sont nombreuses !

Cet objet est bien particulier dans le sens où il n'est accessible que lorsqu'un événement est déclenché. Son accès ne peut se faire que dans une fonction exécutée par un événement, cela se fait de la manière suivante avec le DOM-0 :

```
element.onclick = function(e) { // L'argument « e » va
récupérer une référence vers l'objet « Event »
    alert(e.type); // Ceci affiche le type de l'événement
(click, mouseover, etc.)
};
```

Et de cette façon là avec le DOM-2 :

```
element.addEventListener('click', function(e) { // L'argument
« e » va récupérer une référence vers l'objet « Event »
    alert(e.type); // Ceci affiche le type de l'événement
(click, mouseover, etc.)
}, false);
```

## Les fonctionnalités de l'objet Event

Vous avez déjà découvert la propriété `type` qui permet de savoir quel type d'événement s'est déclenché. Passons maintenant à la découverte des autres propriétés et méthodes que possède cet objet (attention, tout n'est pas présenté, seulement l'essentiel) :

## Récupérer l'élément de l'événement actuellement déclenché

Une des plus importantes propriétés de notre objet se nomme `target`. Celle-ci permet de récupérer une référence vers l'élément dont l'événement a été déclenché (exactement comme le `this` pour les événements sans le DOM ou avec DOM-0), ainsi vous pouvez très bien modifier le contenu d'un élément qui a été cliqué :

```
<span id="clickme">Cliquez-moi !</span>

<script>
    var clickme = document.getElementById('clickme');

    clickme.addEventListener('click', function(e) {
        e.target.innerHTML = 'Vous avez cliqué !';
    }, false);
</script>
```

Comme il y a toujours un problème quelque part, voilà qu'Internet Explorer (versions antérieures à la version 9) ne supporte pas cette propriété. Ou plutôt, il la supporte à sa manière avec la propriété `srcElement`. Voici le même code que précédemment mais compatible avec tous les navigateurs (dont IE) :

```
<span id="clickme">Cliquez-moi !</span>

<script>
    function addEvent(element, event, func) { // On réutilise notre fonction de compatibilité pour les événements DOM-2
        if (element.addEventListener) {
            element.addEventListener(event, func, false);
        } else {
            element.attachEvent('on' + event, func);
        }
    }

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

    addEvent(clickme, 'click', function(e) {
        var target = e.target || e.srcElement; // Si vous avez oublié cette spécificité de l'opérateur OU, allez voir le chapitre des conditions
        target.innerHTML = 'Vous avez cliqué !';
    });
</script>
```

À noter qu'ici nous avons fait un code compatible pour Internet Explorer parce que le but était de gérer la compatibilité, mais pour le reste des autres codes nous ne le ferons que si cela s'avère nécessaire. Il vaut donc mieux avoir un navigateur à jour pour pouvoir tester tous les codes de ce cours.

## Récupérer l'élément à l'origine du déclenchement de l'événement

*Hum, ce n'est pas un peu la même chose ?*

Eh bien non ! Pour expliquer cela de façon simple, certains événements appliqués à un élément parent peuvent se propager d'eux-mêmes aux éléments enfants ; c'est le cas des événements `mouseover`, `mouseout`, `mousemove`, `click`… ainsi que d'autres événements moins utilisés. Regardez donc cet exemple pour mieux comprendre :

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

<div id="parent1">
  Parent
  <div id="child1">Enfant N°1</div>
  <div id="child2">Enfant N°2</div>
</div>

<script>
    var parent1 = document.getElementById('parent1'),
        result = document.getElementById('result');

    parent1.addEventListener('mouseover', function(e) {
        result.innerHTML = "L'élément déclencheur de
l'événement \"mouseover\" possède l'ID : " + e.target.id;
    }, false);
</script>
```

En testant cet exemple, vous avez sûrement remarqué que la propriété `target` renvoyait toujours l'élément déclencheur de l'événement, or nous souhaitons obtenir l'élément sur lequel a été appliqué l'événement. Autrement dit, on veut connaître l'élément à l'origine de cet événement, et non pas ses enfants.
La solution est simple : utiliser la propriété `currentTarget` au lieu de `target`. Essayez donc par vous-mêmes après modification de cette seule ligne, l'ID affiché ne changera jamais :

```
result.innerHTML = "L'élément déclencheur de l'événement
\"mouseover\" possède l'ID : " + e.currentTarget.id;
```

Bien que cette propriété semble intéressante pour certains cas d'application, il est assez difficile de la mettre en pratique car Internet Explorer (versions antérieures à la version 9) ne la supporte pas et il n'y a pas d'autre possibilité correcte (mis à part avec l'utilisation du mot-clé `this` avec le DOM-0 ou sans le DOM).

## Récupérer la position du curseur

La position du curseur est une information très importante, beaucoup de monde s'en sert pour de nombreux scripts comme le *drag & drop*. Généralement, on récupère la position du curseur par rapport au coin supérieur gauche de la page Web, cela dit il est aussi possible de récupérer sa position par rapport au coin supérieur gauche de l'écran. Toutefois, dans ce tutoriel, nous allons nous limiter à la page Web. Regardez la documentation de l'objet `Event` si vous souhaitez en apprendre plus.

Pour récupérer la position de notre curseur, il existe deux propriétés : `clientX` pour la position horizontale et `clientY` pour la position verticale. Étant donné que la position du curseur change à chaque déplacement de la souris, il est donc logique de dire que l'événement le plus adapté à la majorité des cas est `mousemove`.

Il est très fortement déconseillé d'essayer d'exécuter la fonction `alert()` dans un événement `mousemove` ou bien vous allez rapidement être submergés de fenêtres !
Comme d'habitude, voici un petit exemple pour que vous compreniez bien :

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

<script>
    var position = document.getElementById('position');

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

Pas très compliqué, n'est-ce pas ? Bon, il est possible que vous trouviez l'intérêt de ce code assez limité, mais quand vous saurez manipuler les propriétés CSS des éléments vous pourrez, par exemple, faire en sorte que des éléments HTML suivent votre curseur. Ce sera déjà bien plus sympathique !

## Récupérer l'élément en relation avec un événement de souris

Cette fois nous allons étudier une propriété un peu plus « exotique » assez peu utilisée mais qui peut pourtant se révéler très utile ! Il s'agit de `relatedTarget` et elle ne s'utilise qu'avec les événements `mouseover` et `mouseout`.
Cette propriété remplit deux fonctions différentes selon l'événement utilisé. Avec l'événement `mouseout`, elle vous fournira l'objet de l'élément sur lequel le curseur vient d'entrer ; avec l'événement `mouseover`, elle vous fournira l'objet de l'élément dont le curseur vient de sortir.
Voici un exemple qui illustre son fonctionnement :

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

```
<div id="parent1">
  Parent N°1<br />
  Mouseover sur l'enfant
  <div id="child1">Enfant N°1</div>
</div>

<div id="parent2">
  Parent N°2<br />
  Mouseout sur l'enfant
  <div id="child2">Enfant N°2</div>
</div>

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

    child1.addEventListener('mouseover', function(e) {
        result.innerHTML = "L'élément quitté juste avant que
le curseur n'entre sur l'enfant n°1 est :
" + e.relatedTarget.id;
    }, false);

    child2.addEventListener('mouseout', function(e) {
        result.innerHTML = "L'élément survolé juste après que
le curseur ait quitté l'enfant n°2 est :
" + e.relatedTarget.id;
    }, false);
</script>
```

Concernant Internet Explorer (toutes versions antérieures à la neuvième), il vous faut utiliser les propriétés`fromElement` et `toElement` de la façon suivante :

```
child1.attachEvent('onmouseover', function(e) {
    result.innerHTML = "L'élément quitté juste avant que le
curseur n'entre sur l'enfant n°1 est : " + e.fromElement.id;
});

child2.attachEvent('onmouseout', function(e) {
    result.innerHTML = "L'élément survolé juste après que le
curseur ait quitté l'enfant n°2 est : " + e.toElement.id;
});
```

## Récupérer les touches frappées par l'utilisateur

La récupération des touches frappées se fait par le biais de trois événements différents. Dit comme ça, cela laisse sur un sentiment de complexité, mais vous allez voir qu'au final tout est beaucoup plus simple qu'il n'y paraît.
Les événements `keyup` et `keydown` sont conçus pour capter toutes les frappes de touches. Ainsi, il est parfaitement possible de détecter l'appui sur la touche A voire même sur la touche Ctrl. La différence entre ces deux événements se situe dans l'ordre de déclenchement : `keyup` se déclenche lorsque vous relâchez une touche, tandis que `keydown` se déclenche au moment de l'appui sur la touche (comme `mousedown`).

Cependant, faites bien attention avec ces deux événements : toutes les touches retournant un caractère retournerons un caractère *majuscule*, que la touche Maj soit pressée ou non.

L'événement `keypress`, lui, est d'une toute autre utilité : il sert uniquement à capter les touches qui écrivent un caractère, oubliez donc les Ctrl, Alt et autres touches de ce genre qui n'affichent pas de caractère. Alors, forcément, vous vous demandez probablement à quoi peut bien servir cet événement au final ? Eh bien son avantage réside dans sa capacité à détecter les combinaisons de touches ! Ainsi, si vous faites la combinaison Maj + A, l'événement `keypress` détectera bien un A majuscule là où les événements `keyup` et `keydown` se déclencheront deux fois, une fois pour la touche Maj et une deuxième fois pour la touche A.

*Et j'utilise quelle propriété pour récupérer mon caractère, du coup ?*

Si nous devions énumérer toutes les propriétés capables de vous fournir une valeur, il y en aurait trois :`keyCode`, `charCode` et `which`. Ces propriétés renvoient chacune un code ASCII correspondant à la touche pressée.

Cependant, la propriété `keyCode` est amplement suffisante dans tous les cas, comme vous pouvez le constater dans l'exemple qui suit :

```
<p>
    <input id="field" type="text" />
</p>

<table>
    <tr>
        <td>keydown</td>
        <td id="down"></td>
    </tr>
    <tr>
        <td>keypress</td>
        <td id="press"></td>
    </tr>
    <tr>
        <td>keyup</td>
        <td id="up"></td>
    </tr>
</table>
```

```
<script>
    var field = document.getElementById('field'),
        down = document.getElementById('down'),
        press  = document.getElementById('press'),
        up  = document.getElementById('up');

    document.addEventListener('keydown', function(e) {
        down.innerHTML = e.keyCode;
    }, false);

    document.addEventListener('keypress', function(e) {
        press.innerHTML = e.keyCode;
    }, false);

    document.addEventListener('keyup', function(e) {
        up.innerHTML = e.keyCode;
    }, false);
</script>
```

*Je ne veux pas obtenir un code, mais le caractère !*

Dans ce cas, il n'existe qu'une seule solution : la méthode `fromCharCode()`. Elle prend en paramètre une infinité d'arguments. Cependant, pour des raisons un peu particulières qui ne seront abordées que plus tard dans ce cours, sachez que cette méthode s'utilise avec le préfixe `String.`, comme suit :

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

Cette méthode est donc conçue pour convertir les valeurs ASCII vers des caractères lisibles. Faites donc bien attention à n'utiliser cette méthode qu'avec un événement `keypress` afin d'éviter d'afficher, par exemple, le caractère d'un code correspondant à la touche Ctrl, cela ne fonctionnera pas !

Pour terminer, voici un court exemple :

```
String.fromCharCode(84, 101, 115, 116); // Affiche : Test
```

## Bloquer l'action par défaut de certains événements

Eh oui, nous y revenons ! Nous avons vu qu'il est possible de bloquer l'action par défaut de certains événements, comme la redirection d'un lien vers une page Web. Sans le DOM-2, cette opération était très simple vu qu'il suffisait d'écrire `return false;`. Avec l'objet `Event`, c'est quasiment tout aussi simple vu qu'il suffit juste d'appeler la méthode `preventDefault()` !

Reprenons l'exemple que nous avions utilisé pour les événements sans le DOM et utilisons donc cette méthode :

```
<a id="link" href="http://www.siteduzero.com">Cliquez-moi
!</a>

<script>
    var link = document.getElementById('link');

    link.addEventListener('click', function(e) {
        e.preventDefault(); // On bloque l'action par défaut
de cet événement
        alert('Vous avez cliqué !');
    }, false);
</script>
```

C'est simple comme bonjour et ça fonctionne sans problème avec tous les navigateurs, que demander de plus ? Enfin, tous les navigateurs sauf les versions d'Internet Explorer antérieures à la neuvième (c'est pénible, hein ?). Pour IE, il va vous falloir utiliser la propriété `returnValue` et lui attribuer la valeur `false`pour bloquer l'action par défaut :

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

Pour avoir un code compatible avec tous les navigateurs, utilisez donc ceci :

```
e.returnValue = false;

if (e.preventDefault) {
    e.preventDefault();
}
```