Les événements

Que sont les événements ?

Les événements permettent de déclencher une fonction selon qu'une action s'est produite ou non. Par exemple, on peut faire apparaître une fenêtre alert() lorsque l'utilisateur survole une zone d'une page Web.

« Zone » est un terme un peu vague, il vaut mieux parler d'élément (HTML dans la plupart des cas). Ainsi, vous pouvez très bien ajouter un événement à un élément de votre page Web (par exemple, une balise) pour faire en sorte de déclencher un code JavaScript lorsque l'utilisateur fera une action sur l'élément en question.

La théorie

Liste des événements

Il existe de nombreux événements, tous plus ou moins utiles. Parmi les évènements utiles que nous n'aborderons pas, sachez qu'il en existe des spécifiques pour les plateformes mobiles (smartphones, tablettes, etc...).

Voici la liste des événements principaux, ainsi que les actions à effectuer pour qu'ils se déclenchent :

Nom de l'événement	Action pour le déclencher
click	Cliquer (appuyer puis relâcher) sur l'élément
dblclick	Double-cliquer sur l'élément
mouseover	Faire entrer le curseur sur l'élément
mouseout	Faire sortir le curseur de l'élément
mousedown	Appuyer (sans relâcher) sur le bouton gauche de la souris sur l'élément
mouseup	Relâcher le bouton gauche de la souris sur l'élément
mousemove	Faire déplacer le curseur sur l'élément
keydown	Appuyer (sans relâcher) sur une touche de clavier sur l'élément

Nom de l'événement	Action pour le déclencher
keyup	Relâcher une touche de clavier sur l'élément
keypress	Frapper (appuyer puis relâcher) une touche de clavier sur l'élément
focus	« Cibler » l'élément
blur	Annuler le « ciblage » de l'élément
change	Changer la valeur d'un élément spécifique aux formulaires (input,checkbox, etc.)
input	Taper un caractère dans un champ de texte (son support n'est pas complet sur tous les navigateurs)
select	Sélectionner le contenu d'un champ de texte (input, textarea, etc.)

Il a été dit précédemment que les événements mousedown et mouseup se déclenchaient avec le bouton gauche de la souris. Ceci n'est pas tout à fait exact, ces deux événements peuvent se déclencher avec d'autres boutons de la souris comme le clic de la molette ou le bouton droit. Cependant, cela ne fonctionne pas avec tous les navigateurs comme Firefox qui a choisi de bloquer cette possibilité. L'utilisation de ces deux événements se limite donc généralement au bouton gauche de la souris.

Il existe aussi deux événements spécifiques à l'élément <form>, que voici :

Nom de l'événement	Action pour le déclencher
submit	Envoyer le formulaire
reset	Réinitialiser le formulaire

Retour sur le focus

Le focus définit ce qui peut être appelé le « ciblage » d'un élément. Lorsqu'un élément est ciblé, il va recevoir tous les événements de votre clavier. Un exemple simple est d'utiliser un <input> de type

text: si vous cliquez dessus alors l'input possède le focus. Autrement dit : il est ciblé, et si vous tapez des caractères sur votre clavier vous allez les voir s'afficher *dans* l'input en question.

Le focus peut s'appliquer à de nombreux éléments, ainsi, si vous tapez sur la touche `Tabulation` de votre clavier alors que vous êtes sur une page Web, vous allez avoir un élément de ciblé ou de sélectionné qui recevra alors tout ce que vous tapez sur votre clavier. Par exemple, si vous avez un lien de ciblé et que vous tapez sur la touche `Entrée` de votre clavier alors vous serez redirigé vers l'URL contenue dans ce lien.

La pratique

Utiliser les événements

Nous allons donc voir comment les utiliser sans le DOM, ce qui est considérablement plus limité.

Commençons par l'événement click sur un simple:

```
<span onclick="alert('Hello !');">Cliquez-moi !</span>
```

Il suffit de cliquer sur le texte pour que la boîte de dialogue s'affiche. Afin d'obtenir ce résultat nous avons ajouté à notre un attribut contenant les deux lettres « on » et le nom de notre événement « click » ; nous obtenons donc « onclick ».

Cet attribut possède une valeur qui est un code JavaScript, vous pouvez y écrire quasiment tout ce que vous souhaitez, mais tout doit tenir entre les guillemets de l'attribut.

Le mot-clé this

Il s'agit d'une propriété pointant sur l'objet actuellement en cours d'utilisation. Donc, si vous faites appel à ce mot-clé lorsqu'un événement est déclenché, l'objet pointé sera l'élément qui a déclenché l'événement. Exemple :

```
<span onclick="alert('Voici le contenu de l\'élément que vous avez cliqué :\n\n' +
this.innerHTML);">Cliquez-moi !</span>
```

Retour sur le focus

Voici un exemple qui vous montrera ce que ça donne sur un input classique et un lien :

```
<input id="input" type="text" size="50" value="Cliquez ici !" onfocus="this.value='Appuyez
maintenant sur votre touche de tabulation.';" onblur="this.value='Cliquez ici !';"/>
<br /><br/>
```

```
<a href="#" onfocus="document.getElementById('input').value = 'Vous avez maintenant le focus sur le lien !';">Un lien</a>
```

Comme vous pouvez le constater, lorsque vous cliquez sur l'input, celui-ci « possède » le focus : il exécute donc l'événement et affiche alors un texte différent vous demandant d'appuyer sur votre touche de tabulation. L'appui sur la touche de tabulation permet de faire passer le focus à l'élément suivant. En appuyant sur cette touche vous faites perdre le focus à l'input, ce qui déclenche l'événement blur (qui désigne la perte du focus) et fait passer le focus sur le lien qui affiche alors son message grâce à son événement focus.

Bloquer l'action par défaut de certains événements

Que se passe t'il si l'on applique un événement click sur un lien :

```
<a href="http://www.google.com" onclick="alert('Vous avez cliqué !');">Cliquez-moi !</a>
```

La fonction alert() a bien fonctionné, mais après vous avez été redirigé vers le site de Google, or on souhaite bloquer cette redirection. Pour cela, il suffit juste d'ajouter le code return false; dans notre événement click :

```
<a href="http://www.siteduzero.com" onclick="alert('Vous avez cliqué !'); return false;">Cliquez-moi !</a>
```

Ici, le return false; sert juste à bloquer l'action par défaut de l'événement qui le déclenche.

À noter que l'utilisation de return true; permet de faire fonctionner l'événement comme si de rien n'était. En clair, comme si on n'utilisait pas de return false; Cela peut avoir son utilité si vous utilisez, par exemple, la fonction confirm() dans votre événement.

L'utilisation de « javascript: » dans les liens : une technique prohibée

Dans certains cas, vous allez devoir créer des liens juste pour leur attribuer un événement click et non pas pour leur fournir un lien vers lequel rediriger. Dans ce genre de cas, il est courant de voir ce type de code :

```
<a href="javascript: alert('Vous avez cliqué !');">Cliquez-moi !</a>
```

Il est fortement déconseiller de faire cela. Il s'agit en effet d'une vieille méthode qui permet d'insérer du JavaScript directement dans l'attribut href de votre lien juste en ajoutant javascript: au début de l'attribut. Cette technique est maintenant obsolète. Il existe une méthode alternative :

```
<a href="#" onclick="alert('Vous avez cliqué !'); return false;">Cliquez-moi !</a>
```

On remplace javascript: par un dièse (#) puis on a mis notre code JavaScript dans l'événement approprié (click). Par ailleurs, on libère l'attribut href, ce qui nous permet, si besoin, de laisser un lien

pour ceux qui n'activent pas le JavaScript ou bien encore pour ceux qui aiment bien ouvrir leurs liens dans de nouveaux onglets.

Pourquoi un return false; ?

Parce que le dièse redirige tout en haut de la page Web, ce qui n'est pas ce que l'on souhaite. On bloque donc cette redirection avec notre petit bout de code.

L'utilisation d'un lien uniquement pour le déclenchement d'un événement click n'est pas une bonne chose, préférez plutôt l'utilisation d'une balise <button> à laquelle vous aurez retiré le style CSS. La balise <a>, elle, est conçue pour rediriger vers une page Web et non pas pour servir exclusivement de déclencheur !

Les événements au travers du DOM

Le DOM-0

Cette interface est vieille mais n'est pas forcément dénuée d'intérêt. Elle reste très pratique pour créer des événements et peut parfois être préférée au DOM-2.

Commençons par créer un simple code avec un événement click:

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

<script>

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

    element.onclick = function() {
        alert("Vous m'avez cliqué !");
    };

</script>
```

Alors, voyons par étapes ce que nous avons fait dans ce code :

- On récupère tout d'abord l'élément HTML dont l'ID est clickme;
- On accède ensuite à la propriété onclick à laquelle on assigne une fonction anonyme ;
- Dans cette même fonction, on fait un appel à la fonction alert()avec un texte en paramètre.

Comme vous le voyez, on définit maintenant les événements non plus dans le code HTML mais directement en JavaScript. Chaque événement standard possède donc une propriété dont le nom

est, à nouveau, précédé par les deux lettres « on ». Cette propriété ne prend plus pour valeur un code JavaScript brut, mais soit le nom d'une fonction, soit une fonction anonyme.

Concernant la suppression d'un événement avec le DOM-0, il suffit tout simplement de lui attribuer une fonction anonyme vide :

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

Le DOM-2

Le DOM-2, permet la création multiple d'un même événement et gère aussi l'objet Event.

Comme pour les autres interfaces événementielles, voici un exemple avec l'événement click :

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

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

    element.addEventListener('click', function() {
        alert("Vous m'avez cliqué !");
    });
</script>
```

Nous n'utilisons plus une propriété mais une méthode nommée addEventListener(), et qui prend trois paramètres :

- Le nom de l'événement (sans les lettres « on ») ;
- La fonction à exécuter ;
- Un booléen **optionnel** pour spécifier si l'on souhaite utiliser la phase de capture ou bien celle de bubbling.

Le DOM-2 permet la création multiple d'événements identiques pour un même élément, ainsi, vous pouvez très bien faire ceci :

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

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

    // Premier événement click
    element.addEventListener('click', function() {
        alert("Et de un !");
    });

    // Deuxième événement click
    element.addEventListener('click', function() {
        alert("Et de deux !");
```

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

Si vous exécutez ce code, vous verrez que les événements ne se déclenchent pas forcément dans l'ordre de création.

Pour supprimer des événements on utilise la méthode removeEventListener() :

```
element.addEventListener('click', myFunction); // On crée l'événement

element.removeEventListener('click', myFunction); // On supprime l'événement en lui repassant les mêmes paramètres
```

Toute suppression d'événement avec le DOM-2 se fait avec les mêmes paramètres utilisés lors de sa création ! Cependant, cela ne fonctionne pas aussi facilement avec les fonctions anonymes ! Tout événement DOM-2 créé avec une fonction anonyme est particulièrement complexe à supprimer, car il faut posséder une référence vers la fonction concernée, ce qui n'est généralement pas le cas avec une fonction anonyme.

Attention ! Les versions d'Internet Explorer antérieures à la version 9 ne supportent pas l'ajout d'évènements DOM-2, tout du moins pas de la même manière que la version standardisée.

Les phases de capture et de bubbling

Du côté de la théorie

La première, étape de l'exécution d'un événement est la **capture**, qui s'exécute avant le déclenchement de l'événement, tandis que la deuxième est le **bubbling**, qui s'exécute après que l'événement ait été déclenché. Toutes deux permettent de définir le sens de propagation des événements.

Mais qu'est-ce que la propagation d'un événement ? Pour expliquer cela, prenons un exemple avec ces deux éléments HTML :

```
<div>
    <span>Du texte !</span>
</div>
```

Si nous attribuons une fonction à l'événement click de chacun de ces deux éléments et que l'on clique sur le texte, quel événement va se déclencher en premier ?

Notre réponse se trouve dans les phases de capture et de bubbling. Si vous décidez d'utiliser la capture, alors l'événement du se déclenchera en premier puis viendra ensuite l'événement du . En revanche, si vous utilisez le bubbling, ce sera d'abord l'événement du qui se déclenchera, puis viendra par la suite celui du .

La phase de bubbling est celle définie par défaut et sera probablement celle que vous utiliserez la majorité du temps;
Voici en pratique l'utilisation de ces deux phases :

```
<div id="capt1">
    <span id="capt2">Cliquez-moi pour la phase de capture.</span>
</div>

<div id="boul1">
    <span id="boul2">Cliquez-moi pour la phase de bouillonnement.</span>
</div>

<script>
    var capt1 = document.getElementById('capt1'),
        capt2 = document.getElementById('capt2'),
        boul1 = document.getElementById('boul1'),
        boul2 = document.getElementById('boul2');

    capt1.addEventListener('click', function() {
        alert("L'événement du div vient de se déclencher.");
    }, true);

    capt2.addEventListener('click', function() {
        alert("L'événement du span vient de se déclencher.");
    }, true);

    boul1.addEventListener('click', function() {
        alert("L'événement du div vient de se déclencher.");
    }, false);

    boul2.addEventListener('click', function() {
        alert("L'événement du span vient de se déclencher.");
    }, false);
</script>
```

L'objet Event

Généralités sur l'objetEvent

Cet objet sert à 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…

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.)
});
```

Les fonctionnalités de l'objetEvent

La propriété type permet de savoir quel type d'événement s'est déclenché.

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, et 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 !';
    });
</script>
```

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

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. Voici un 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;
    });
</script>
```

En testant cet exemple, on remarque que la propriété target renvoye 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;
```

Attention ! Cette propriété n'est pas supportée par les versions d'Internet Explorer antérieures à la 9.

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.

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.

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';
    });
</script>
```

Récupérer l'élément en relation avec un événement de souris

La propriété relatedTarget ne s'utilise qu'avec les événements mouseover et mouseout.
Cette propriété remplit deux fonctions différentes selon l'événement utilisé. Avec l'événement mouseover, elle vous fournira l'objet de l'élément sur lequel le curseur vient d'entrer ; avec l'événement mouseout, 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;
    });

    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;
    });
</script>
```

Récupérer les touches frappées par l'utilisateur

La récupération des touches frappées se fait par le biais de trois événements différents.

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 retourneront 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. 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`.
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;
    });

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

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

Pour obtenir un caractère à la place du code, il existe la méthode fromCharCode(). Elle 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 !

Voici un court exemple :

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

Bloquer l'action par défaut de certains événements

Avec l'objet Event, 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.google.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 !');
    });
</script>
```

Attention ! Le blocage de l'action par défaut ne fonctionne pas pour les versions d'Internet Explorer antérieures à la 9. Il est cependant possible de résoudre cela simplement en faisant un simple e.returnValue = false; au sein du code de votre évènement.