Javascript : les évènements

Contenu

Définition

Les évènements correspondent à des actions effectuées soit par un utilisateur, soit par le navigateur lui-même. Ce sont les évènements qui permettent la grande interactivité du Javascript.

Vous connaissez déjà des évènements sans les connaître : cliquer sur un lien est un évènement, charger une page web l'est aussi ou encore envoyer un formulaire en cliquant sur le bouton de type submit.

Liste des événements

Il existe des évènements dits *standard*, c'est-à-dire validés par la norme ECMA, et des évènements dits non standards non reconnus par la norme ECMA, ces évènements sont spécifiques aux navigateurs.

Voici la liste des principaux évènements Javascript :

Evènement	Description
blur	Lorsqu'un élément de formulaire perd le focus.
change	Lorsque la valeur d'un champ de formulaire est modifiée
click	Lorsque l'utilisateur clique sur un bouton, un lien ou tout autre élément
focus	Lorsqu'un élément de formulaire prend le focus
load	Lorsque la page est chargée par le navigateur
mouseover	Lorsque l'utilisateur place le pointeur de la souris sur un lien ou tout autre élément
mouseout	Lorsque le pointeur de la souris quitte un lien ou tout autre élément
select	Lorsque l'utilisateur sélectionne un champ dans un élément de formulaire
submit	Lorsque l'utilisateur clique sur le bouton de soumission d'un formulaire
unload	Lorsque l'utilisateur quitte la page

⇨ Liste complète des évènements

> **N'appelez jamais la fonction `document.write();` depuis un évènement car cela remplacerait tout le contenu de la page !**

Les événements en pratique

Lorsqu'un évènement a lieu, Javascript va le capter et exécuter du code (situé généralement dans une fonction) pour exécuter des actions consécutives à l'évènement (par exemple modifier une couleur ou une dimension, contrôler la saisie dans un champ de formulaire, mettre à jour du contenu HTML ou des données).

Gestionnaire d'évènement

Les évènements Javascript sont interceptés via un gestionnaire d'évènement :

Le nom de l'évènement doit être écrit en *CamelCase* préfixé par « **on** », par exemple :

Evènement	**Attribut HTML**
click	onClick
mouseover	onMouseOver

```
onEvenement="fonction()";
```

Par exemple, `<p onClick="alert('OK')">Clique ici</p>.`

Dans cet exemple, au clic de la souris, une boîte d'alerte avec le message indiqué s'ouvre.

Javascript « non intrusif »

Vous lirez souvent sur le web des exemples Javascript similaires à celui du paragraphe précédent où le code de l'évènement se situe directement dans la balise HTML (onClick, onBlur etc.).

CETTE PRATIQUE EST A OUBLIER COMPLETEMENT

Depuis la sortie du HTML 5 (mais aussi déjà avant, en HTML 4), la bonne pratique est de séparer (voire d'externaliser complètement dans un fichier) le Javascript. On appelle ça le javascript **« non intrusif ».** Outre une meilleure visibilité du code de la page web (le javascript étant alors centralisé à un seul endroit), cela permet de diminuer le temps de chargement de la page.

Par conséquent, il est nécessaire d'utiliser les évènements d'une autre manière, en les associant à des écouteurs d'évènements via les méthodes `addEventListener/getElementById` ou encore `QuerySelector`.

Méthodes `addEventListener` et `getElementById`

En Javascript non intrusif, on peut intercepter un évènement grâce aux 2 méthodes `getElement-ById` qui va cibler l'identifiant de la balise HTML et `AddEventListener` qui va créer un « gestionnaire d'écoute » sur l'objet ciblé, c'est-à-dire créer connecter cet objet à l'évènement concerné.

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
  </head>
  <body>
    <p id="button1">Clique-moi</p>
  </body>
</html>
<script>
var element = document.getElementById("button1");

element.addEventListener("click", function()
{
   alert("OK");
});
</script>
```

Notez que dans ce cas le nom de l'évènement n'est pas préfixé par « **on** ».

Exercice

Testez l'exemple ci-dessus.

Méthodes `querySelector`

La méthode `querySelector` présente une autre syntaxe mais effectue la même chose que le couple `getElementById/AddEventListener` :

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
  </head>
  <body>
    <p id="button1">Clique-moi</p>
  </body>
</html>
```

```
<script>
document.querySelector('#lien').onclick = function()
{
    alert('Vous avez cliqué !');
}
</script>
```

Autres méthodes

D'autres méthodes permettent de récupérer des éléments :

- `getElementsByClassName(nom_classe)` : cible les éléments sur lesquels une (ou plusieurs) classe CSS spécifique est appliquée et retourne un tableau de ces éléments.

  ➔ documentation

- `getElementsByTagName(nom_balise)` : cible la balise HTML (tag) passée en argument (une seule possible)
  ➔ documentation

Attention : ne pas oublier le 's' dans ces 2 méthodes : `getElement`**`s`**`By…`

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
  </head>
  <body>
    <p id="button1">Clique-moi</p>
  </body>
</html>
<script>
document.querySelector('#lien').onclick = function()
{
    alert('Vous avez cliqué !');
}
</script>
```

Gestionnaires d'événements disponibles en Javascript

Il nous semble utile dans cette partie "avancée" de présenter la liste des objets auxquels correspondent des gestionnaires d'événement bien déterminés.

Objets	Balises	Evenements
Fenetre	<body>	load, unload, focus, blur, error
Ancre	<a>	click,mouseover, mouseout
Formulaire	<form>	reset, submit
Eléments de formulaire	<input> <select> <textarea>	click, focus, blur, change, select (<select> et <textarea>)
Elément de zone de texte	<area>, <map>	click, focus, blur
Image	<img>	load , abort, error

Il y a beaucoup d'autres gestionnaires d'événement non traités dans ce cours ; par exemple les événements liés au clavier : ***keypress, keydown, keyup***.

La syntaxe de MouseOver

Le code du gestionnaire d'événement `mouseover` s'ajoute aux balises de lien :

```
<a href="" onMouseOver="action();">lien</a>
```

Ainsi, lorsque l'utilisateur passe avec sa souris sur le lien, la fonction `action()` est appelée. L'attribut `href` est indispensable. Il peut contenir l'adresse d'une page Web si vous souhaitez que le lien soit actif ou simplement des guillemets si aucun lien actif n'est prévu. Nous reviendrons ci-après sur certains désagréments du codage `href=""`.

Voici un exemple. Par le survol du lien "message important", une fenêtre d'alerte s'ouvre.

Le code est :

```
<body>
   <a href="#" onMouseOver="alert('Coucou');">message important</a>
</body>
```

Ou si vous préférez utiliser les balises <head> :

```
<html>
<head>
</head>
<body>
   <a href="" onMouseOver="message();">message important</a>
   <script>
   function message()
   {
      alert("Coucou");
   }
   </script>
</body>
</html>
```

Problème! Et si on clique quand même...

Vous avez codé votre instruction `mouseover` avec le lien fictif `<a href="". . . >`, vous avez même prévu un petit texte, demandant gentiment à l'utilisateur de ne pas cliquer sur le lien et comme de bien entendu celui-ci clique quand même.

Horreur, le navigateur affiche alors l'entièreté des répertoires de la machine ou du site ou affiche un message d'erreur. Ce qui est un résultat non désiré et pour le moins imprévu.

Pour éviter cela, prenez l'habitude de mettre l'adresse de la page en cours ou plus simplement le signe # (pour un ancrage) entre les guillemets de *href*. Ainsi, si le lecteur clique quand même sur le

lien, au pire, la page en cours sera simplement rechargée et sans perte de temps car elle est déjà dans le cache du navigateur.

Prenez donc l'habitude de mettre le code suivant

```
<a href="#" onMouseOver="action();">lien</a>
```

Changement d'images

Avec le gestionnaire d'événement *mouseover*, on peut prévoir qu'après le survol d'une image, une autre image apparaisse (pour autant qu'elle soit de la même taille).

Le code est relativement simple.

```
<img src="images/bleu_p.jpg" border="0" width="38" height="20"
onMouseOver="this.src='images/bleu_a.jpg'"
onMouseOut="this.src='images/bleu_p.jpg'">
```

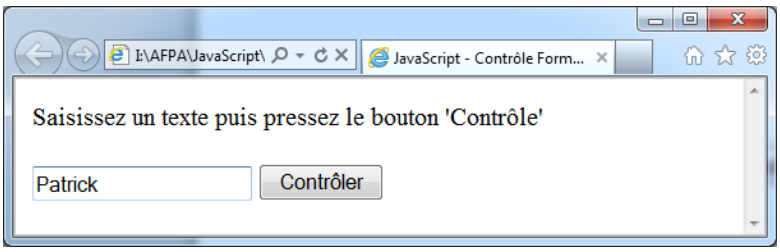
L'image invisible

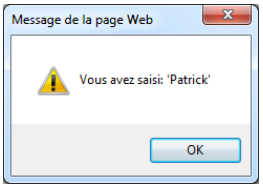
Ce changement d'image ne vous donne-t-il pas des idées ?... Petit futé ! Et oui, on peut prévoir une image invisible de la même couleur que l'arrière-plan (même transparente). On la place avec malice sur le chemin de la souris de l'utilisateur et son survol peut déclencher, à l'insu de l'utilisateur, un feu d'artifice d'actions de votre choix.

Exercice

Le clic sur le bouton *Contrôler* engendre l'appel à la fenêtre d'information.

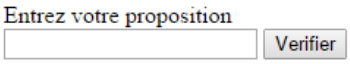
Résultat à obtenir :





Nombre Magique (the Magic Number)

Reprenez l'exercice du nombre magique




```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Nombre magique</title>
  </head>
  <body>
    <div id="label1">Entrez votre proposition</div>
    <input id="textBox1" value="">
    <input type="button" id="button1" value="Verifier" onclick="verif();">
  </body>
</html>
<script>

</script>
```

Votre programme doit générer un nombre aléatoire à l'aide de la fonction `Math.random`.

Ecrivez la fonction *verif* qui doit vérifier si la saisie de l'utilisateur (dans textBox1) correspond au nombre magique, elle affiche des informations (trop grand, trop petit dans le label1.

Quand votre programme fonctionne, modifiez-le pour rendre le javascript non intrusif.