Leçon 4-1 – JavaScript : les événements

s1 ----------

Dans cette partie, nous allons voir comment introduire de l'interactivité dans une page web. Nous introduirons la notion d'événement JavaScript et nous préciserons comment il est traité par le navigateur. Nous verrons ensuite comment mettre en oeuvre des événements usuels tels que la fin du chargement de la page HTML, le clic souris, un événement clavier.

s2 ----------

La notion d'interactivité signifie que l'animation va réagir à des événements déclenchés par un utilisateur, tels que des actions sur la souris ou le clavier.

Qu'est-ce qu'un événement ? La notion d'événement est intimement liée à la structure du document HTML (c'est-à-dire l'ensemble des éléments qui constituent ce document : ce sont par exemple, comme sur la figure ci-dessous, des conteneurs div, des éléments span à l'intérieur d'un paragraphe p, des cellules de tableau td à l'intérieur de lignes tr, elles même à l'intérieur d'une table. La notion d'événement s'appuie donc sur le DOM (Document Object Model dont nous avons déjà parlé), ensemble de méthodes qui permettent de manipuler la structure d'un document HTML.
En outre, la notion d'événement nous oblige à introduire la notion de gestionnaire d'événement.

La notion d'événement n'est pas implémentée exactement de la même façon dans tous les navigateurs. Cependant ce qui va être présenté ici leur est commun.

s3 ----------

Pour bien comprendre ce qu'est un événement et ce qu'il implique, nous pouvons voir le document comme un ensemble de bâtiments, constitués d'étages, chaque étage contenant différentes pièces. Certains niveaux (pièces, étages ou immeuble) sont placés sous la responsabilité d'un gestionnaire. D'autres pas. Ces gestionnaires ont pour tâche de réagir à un certain type de bruit. Un gestionnaire principal s'occupe de cet ensemble d'immeubles.

Un bruit de chaise se produit dans une des pièces. Pour le document HTML, un objet JavaScript est créé et contient des données telles que le type d'événement (par exemple le fait d'appuyer sur la partie gauche de la souris) et la cible (en anglais target) (c'est-à-dire le dernier élément recevant le clic, ici l'élément appelé element3). Cet événement implique plusieurs phases distinctes.

s4 ----------

La première phase est appelée phase de capture. Le gestionnaire principal va traverser toute la structure jusqu'à l'endroit où s'est produit l'événement. Sur son chemin, il va pouvoir rencontrer des gestionnaires de bâtiment, d'étage ou de pièce dont la tâche est de réagir à un événement d'un type donné. Si le gestionnaire principal a été autorisé à le faire lors de cette phase, il pourra leur transmettre l'information concernant l'événement. Si le gestionnaire rencontré est supposé réagir à tout événement de type bruit de chaise, alors sa réaction sera déclenchée, par exemple envoyer un courrier à la direction. Pour HTML, l'élément traversé est appelé cible courante (en anglais currentTarget).

s5 ----------

Seconde phase. Le gestionnaire principal a atteint la pièce où s'est produit le bruit. Il va refaire le chemin inverse en commençant par cette pièce, la cible. Avec une différence : si lors de la phase de capture il était autorisé à transmettre les informations, il ne va pas la retransmettre une nouvelle fois. En revanche, s'il n'était pas autorisé à le faire lors de la phase de capture, il va le faire à ce moment-là.
Les actions des gestionnaires ne sont donc exécutées qu'une fois.

Si elle a été placée sous la responsabilité d'un gestionnaire et que ce gestionnaire est supposé réagir à tout événement de type bruit de chaise, alors l'action sera déclenchée. Si un gestionnaire de pièce est présent et supposé réagir à un tout autre type de bruit, rien ne se passera ; de même si aucun gestionnaire n'est présent, aucune action nouvelle ne sera déclenchée.

Pour le document HTML, toute cette phase depuis la cible jusqu'à l'élément racine est appelée phase de bouillonnement (en anglais bubbling).

Voici pour le principe général. L'événement est donc un objet JavaScript qui va pouvoir être transporté au niveau de tous les parents de l'élément qui a reçu le clic. On dit que cet objet se propage dans la structure du document HTML. Nous n'entrerons pas plus en détail dans ce mécanisme. Ceux que ça intéresse trouveront à la fin de cette leçon un lien qui détaille les interactions entre le DOM (Document Object Model) et le concept d'événement et un autre qui précise les différences de comportement entre navigateurs.

Une première remarque : nous verrons qu'il est possible à chaque instant de stopper le travail de retransmission de l'événement.

s6 ----------

Introduire de l'interactivité au niveau d'un document HTML signifie donc plusieurs choses :

- définir l'élément au niveau duquel installer un gestionnaire d'événement
- pour cet élément, définir quel type d'événement on souhaite détecter
- définir ce qui doit se passer lorsque cet événement a été détecté, généralement en définissant la fonction à exécuter

C'est le langage JavaScript qui va s'occuper de ces aspects

- la détection d'un événement sur un élément se fait par l'intermédiaire d'un écouteur (ou listeneur) qui va écouter spécifiquement un type d'événement
- à un écouteur est associée la fonction qui sera exécutée lorsque l'événement sera détecté

Comment gérer les phases de capture et de bubbling ? En général, on n'exploite pas la phase de capture, mais plutôt celle de bouillonnement. C'est le paramètre useCapture=false qui va le préciser.
Et puis, pour les applications que vous allez programmer, les choses seront plus simples : un gestionnaire unique sera installé au niveau de l'élément qui doit recevoir le clic. Nous n'exploiterons en général pas le fait qu'un événement soit retransmis à tous les parents de cet élément.

Pour autant, il très est important de bien comprendre ce mécanisme pour maîtriser, dans le cas où plusieurs gestionnaires seraient présents, l'ordre dans lequel s'exécutent les actions attachées aux gestionnaires.

s7 ----------

Voyons tout de suite comment se fait la mise en oeuvre.

Dans l'exemple présenté ci-dessous, nous avons un élément de type case à cocher avec son identifiant check1.

À cet élément, nous associons un écouteur d'événement de type clic et on définit la fonction qui sera exécutée. L'argument e de cette fonction est précisément l'objet événement lui-même.
Le dernier argument de cette fonction est l'argument useCapture. "useCapture=false" signifie que les éventuels gestionnaires rencontrés lors de la phase de capture ne sont pas activés. Ce qui n'a ici aucune influence, car un seul gestionnaire est défini.

s8 ----------

Dans l'exemple présenté ici, nous avons deux cases à cocher check1 et check2 à l'intérieur d'un conteneur div appelé element1.

À chacun de ces éléments est associé un écouteur ainsi que la fonction à exécuter.

Je vous propose de tester ce code tel qu'il est écrit. Vous remarquerez que le paramètre useCapture est à true. L'alerte "clic sur element1" se produit bien avant les alertes associées aux cases à cocher.

Ensuite, décommentez la ligne où l'on applique la méthode stopPropagation() à l'événement. Vous constaterez que la propagation est stoppée.

Testez enfin ce code avec le paramètre useCapture à false.

s9 ----------

Cet exemple illustre une autre façon d'installer un gestionnaire d'événement. Ici le gestionnaire est installé directement au niveau de la balise HTML.

s10 ----------

Voyons quelques événements parmi les plus usuels.

L'événement Click se produit lorsque l'utilisateur clique sur l'élément associé à cet événement, par exemple un bouton.

L'événement MouseOver se produit lorsque l'utilisateur positionne le curseur de la souris au-dessus de l'élément associé à cet événement.

L'événement Keydown se produit lorsque l'utilisateur appuie sur une touche de son clavier. Le gestionnaire de cet événement est placé au niveau le plus haut de la structure, celui du document. Attention ici à la version de JavaScript utilisée.

L'événement Load se produit lorsque le navigateur de l'utilisateur charge la page en cours.

Pour une liste complète des événements, merci de consulter le lien proposé. Vous y découvrirez notamment qu'un événement ne peut pas être associé à n'importe quel objet.

s11 ----------

Voyons à présent comment accéder aux informations de l'événement, notamment, souvent le plus important, quel en a été le type ou la cible.
Reprenons l'exemple 1 et modifions la fonction exécutée : l'expression e.target.name signifie que l'on va chercher la propriété name de la cible de l'événement, ici la chaîne option1. Vous pouvez également afficher l'identifiant de la cible e.target.id

Je vous propose de réaliser cet exercice, puis de reprendre l'exercice avec les deux cases à cocher et de modifier les fonctions afin d'afficher l'identifiant de la cible courante e.currentTarget.id.

s12 ----------

Voyons maintenant comment prendre en compte une interaction au clavier. Si on reprend le schéma précédent

- l'élément qui est ici concerné est le document lui-même
- l'événement que l'on va chercher à détecter est l'événement onKeyDown (lorsqu'une touche sera enfoncée)
- comme précédemment il faudra écrire la fonction qui sera lancée comme conséquence de cet événement.

Justement, comment savoir quelle touche a été enfoncée ? Chaque caractère du clavier possède un identifiant prédéfini. En JavaScript, cet identifiant correspond à l'attribut keyCode de l'événement. Par exemple le code de la touche flèche droite est 39.

Je vous propose de reproduire le code proposé ici. On attache au document, un écouteur d'événement de type keydown et on définit la fonction qui sera exécutée, ici une simple alerte qui affiche le code de la touche.

Bon travail et à tout de suite.