# JavaScript

## DOM 2 Events

Par Richard Vigneux

## Table des matières


- Introduction
- Propagation des événements
- Interface EventTarget
- Interface Event
- Interfaces qui héritent de Event
- Interface MouseEvent
- L'interface spéciale DocumentEvent
- Exemple
- Présentation du TP2


# Introduction

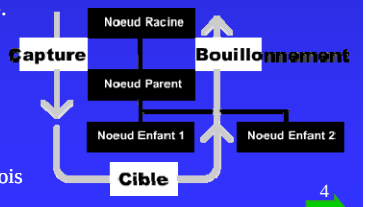
- Le DOM 2 Events est le dernier DOM que nous aborderons. Ce dernier, comme son nom l'indique, nous permet de gérer les événements de façon interactive.
- Le DOM 2 Events est l'un des plus simples puisque ses possibilités sont limitées.
- Le Dom 2 Events se compose de deux interfaces principales.
  - La première « EventTarget », est une extension de l'interface Node et permet principalement d'ajouter ou de supprimer des écouteurs d'événements.
  - La deuxième, « Event » qui est en fait l'interface principale du DOM 2 Events, permet de récupérer les informations sur les événements et de modifier leur comportement. Presque toutes les autres interfaces héritent de celle-ci.
- Références: http://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113/

3

# Propagation des événements

Pour bien travailler avec les événements, vous devez comprendre comment ils se propagent.

- Tout d'abord, vous devez vous rappeler qu'un document HTML a une structure hiérarchique. Ceci veut dire que toutes les balises, hormis la balise <html >, se retrouvent dans d'autres balises.
- Lorsqu'on regarde une page HTML dans un navigateur, cette hiérarchie se présente comme des couches (layer). Par exemple, si on regarde une cellule de tableau, on doit se rappeler que cette cellule est dans une ligne, que cette ligne est dans un tableau, mais aussi que ce tableau est minimalement dans le corps du document (body) qui lui-même est dans la balise <html>. Ceci veux dire que lorsqu'on clique sur une cellule de tableau, l'événement a préalablement lieu dans l'ordre sur les balises <html>, <body>, <table> et <tr>. Ce processus est appelé « phase de capture ».
- Vous devez savoir aussi que lorsque l'événement a atteint la cellule (cible), il refait le parcours en sens inverse. C'est ce qu'on appelle « phase de bouillonnement ». Ce double mouvement fait qu'une capture d'événement « click » placé sur une ligne de tableau <tr> risque d'être appelée 2 fois lorsqu'on clique sur une cellule <td>.




4

# Interface EventTarget (1/2)

Cette interface permet de gérer les événements d'un nœud. Elle offre trois méthodes pour y arriver.

- La première, « addEventListener », permet d'ajouter un écouteur d'événement sur un nœud. Ceci veut dire qu'elle permet de faire exécuter une fonction lorsqu'un certain événement se produit. La fonction associée sera appelée « fonction événementielle » ou « fonction écouteur » (EventListener).
  - addEventListener (typeEvent, EventListener , useCapture)
    - typeEvent → le type d'événement sur lequel on désire réagir (STRING).
    - EventListener → le nom de la fonction qui sera appelée (sans les parenthèses).
    - useCapture → Si « true », indique que « EventListener » sera appelé avant le nœud sur lequel il est lié (BOOLEAN).
  - Exemple d'utilisation:
    - Admettons que nous désirerions que la fonction « GereClic » soit appelée lorsqu'un événement « click » se produit sur une image.

      var_Obj_Image.addEventListener ("click", GereClic, false);

    - Notez que la déclaration de « GereClic » doit avoir un et un seul paramètre pour recevoir l'objet « Event » (voir l'interface EventListener ).

# Interface EventTarget (2/2)

- La deuxième, « removeEventListener », permet d'enlever un écouteur d'événement sur un nœud. Les paramètres de cette méthode sont les mêmes que ceux d' « addEventListener ». D'ailleurs, pour pouvoir retirer le lien entre un nœud et un événement, vous devez impérativement mettre les mêmes paramètres que vous aviez indiqués lors de l'ajout.
  - Exemple d'utilisation (enlever l'écouteur précédent):

    var_Obj_Image.removeEventListener ("click", GereClic, false);
- La dernière, « dispatchEvent », permet de provoquer l'événement comme si l'internaute l'avait fait lui-même. Son seul paramètre est l'événement à provoquer.
  - Exemple d'utilisation (simuler un clic sur une image):

    var_LActionNormaleEtaitDesactivee = var_Obj_Image.dispatchEvent("click");
    - Notez que cette méthode retourne une valeur booléenne qui indique si une des fonctions écouteurs a empêché l'action par défaut de se produire sur le nœud (par exemple un clic sur un lien doit normalement ouvrir le lien).

6

# Interface Event

L'interface Event représente l'objet reçu en paramètre par une fonction événementielle (EventListener). Elle permet de recueillir différentes informations sur l'événement et le nœud lié ainsi que d'effectuer certaines actions en relation avec l'événement.

- Cette interface offre 7 propriétés qui offrent de l'information sur l'événement. Les deux plus importants sont « target » et « currentTarget ».
  - target → représente le nœud sur lequel l'événement a été initialement provoqué.
  - currentTarget → représente le nœud sur lequel l'écouteur a été installé.
- Cette interface offre 3 méthodes.
  - La première, « stopPropagation », arrête le processus de propagation. Donc à partir du moment où l'on utilise cette méthode, l'événement courant ne sera pas transmis aux enfants du nœud courant et il n'y aura pas de bouillonnement.
  - La deuxième, « preventDefault », permet d'éviter que l'événement ne provoque l'action normale qu'il devait provoquer sur le nœud courant (clic sur bouton « submit »).
  - La dernière, « initEvent », permet d'initialiser un objet Event créé à l'aide de l'interface DocumentEvent.

7

# Interfaces qui héritent de Event

Trois interfaces hérite directement de l'interface Event.

- La première, « HTMLEvent », n'offre que des événements standard qui pourront être utilisés avec l'interface EventTarget et la méthode « initEvent » de l'interface Event.
  - load, unload, abort, error, select, change, submit, reset, focus, blur, resize, scroll.
- La deuxième, « MutationEvent », offre une série d'événements liés à la modification de la structure du document (DOM Core). Elle offre aussi 5 propriétés pour obtenir des informations sur les éléments impliqués dans la modification. Enfin, elle offre une méthode particulière pour initialiser ce type d'événement. Ces événements doivent être créés par l'entremise de l'interface DocumentEvent.
  - DOMSubtreeModified, DOMNodeInserted, DOMNodeRemoved, DOMNodeRemovedFromDocument, DOMNodeInsertedIntoDocument, DOMAttrModified, DOMCharacterDataModified
- La dernière, « UIEvent », offre une série d'événements liés au DOM 2 Views. Ce dernier s'intéresse à l'apparence de la page à un moment donné. Cette interface offre elle aussi une méthode particulière pour initialiser ce type d'événement.
  - DOMFocusIn, DOMFocusOut, DOMActivate

8

# Interface MouseEvent

Cette interface hérite de l'interface « UIEvent ». Cette interface est particulièrement intéressante puisqu'elle nous offre, entre autres, des informations sur la position du pointeur de la souris. Cette interface offre elle aussi une méthode particulière pour initialiser ce type d'événement.

- Cette interface offre 6 événements:
  - click, mousedown, mouseup, mouseover, mousemove, mouseout
- Cette interface offre 10 propriétés:
  - screenX → Position horizontale de la souris par rapport à l'écran.
  - screenY → Position verticale de la souris par rapport à l'écran.
  - clientX → Position horizontale de la souris par rapport à la fenêtre du navigateur.
  - clientY → Position verticale de la souris par rapport à la fenêtre du navigateur.
  - ctrlKey → Booléen indiquant si la touche CTRL était enfoncée lors de l'événement.
  - shiftKey → Booléen indiquant si la touche SHIFT était enfoncée lors de l'événement.
  - altKey → Booléen indiquant si la touche ALT était enfoncée lors de l'événement.
  - metaKey → Booléen indiquant si la touche spéciale était enfoncée lors de l'événement.
  - button → [0=Gauche, 1=Centre, 2=Droit]
  - relatedTarget → Utilisé pour connaître l'objet précédent ou suivant (over, out).

9

# L'interface spéciale DocumentEvent

L'interface DocumentEvent étend l'interface Document. Elle n'a qu'une seule méthode et celle-ci a pour objectif de créer un événement. La création d'événement est utile lorsqu'on veut paramétrer un événement à l'aide de méthodes « init… » ou lorsqu'on veut appeler directement une fonction événementielle.

- Exemple d'utilisation:

```
var_Obj_Event = document.creatEvent("MouseEvent");
var_Obj_Event.initMouseEvent("click", true, true, window, 0, 0, 0, 0, 0, false, false, false, false, 0, null);
GereClick(var_Obj_Event );
```

10

# Exemple

Cet exemple consiste à ajouter un écouteur de « click » sur un tableau et à empêcher que l'événement ne soit transmis aux lignes et aux cellules de celui-ci. De plus, lorsqu'un clic aura lieu sur le tableau, nous afficherons son ID.

```
function  GestionClick_Tableau (evenement)
{
   var  var_Obj_Tableau = evenement.currentTarget;

   alert (var_Obj_Tableau .getAttribute("id"));

   evenement.stopPropagation( );
}

var  var_Obj_MonTab = document.getElementByTagName("table")[0];

var_Obj_MonTab .addEventListener ("click", GestionClick_Tableau, false);
```

11

# Présentation du TP2

- Ce TP est une synthèse du cours. Il vous demandera d'utiliser JavaScript , les différents DOM étudiés et des expressions rationnelles. Bien entendu, pour réaliser toutes les manipulations demandées, vous devrez aussi bien connaître le XHTML et les CSS.
- Vous avez trois cours pour réaliser ce travail et vous ne devriez pas en avoir de trop. Comme de raison, le temps en classe ne suffira pas. Pour passer au travers, vous devrez y consacrer au moins autant de temps en dehors des heures de cours. Par contre, si vous constituez une équipe de trois personnes et que tous les coéquipiers travaillent bien, vous devriez vous en sortir vivant.
- Conseil: Mettez les bouchés doubles dès le départ. Plus tard il sera peut-être trop tard!
- Résultat

12


JavaScript
FIN
DOM 2 Events
Par Richard Vigneux