La gestion des événements DOM Niveau 2

par Philippe Beaucart

Date de publication : 12 avril 2010

Dernière mise à jour :

Avant d'aborder les événements du DOM (Document Object Model), vous devez comprendre la construction arborescente d'un document HTML, avec les notions inhérentes de noeuds, de noeud parent et de noeud enfant. Idéalement, vous pouvez acquérir préalablement la notion d'arbre XML qui est le fondement de la construction arborescente des documents HTML.

1 - Rappel de l'historique du DOM W3C:

1-A - Apparition du W3C DOM

Une des utilisations les plus répandues de Javascript est le DHTML (Dynamic HyperText Markup Language). Cependant, DHTML est toujours spécifique au navigateur (client) et requiert donc un code spécifique lui aussi. Ce qui pose le problème évident de portabilité du code d'un navigateur à l'autre.

Les préconisations W3C DOM sont apparues avec l'intégration du modèle Netscape 2.0 dans le DOM Level 0. Le DOM Level 1 est né en Octobre 1998 afin de définir plus précisément la manière de représenter un document (en particulier un document **XML**) sous la forme d'un arbre. La dernière édition W3C **Document Object Model (DOM) Level 3 Core Specification** date du 7 avril 2004. Depuis cette date, des groupes de travail ont fait évoluer les recommandations W3C avec notamment, l'avènement officiel du groupe de travail **W3C Web Applications Working Group** depuis le 6 Janvier 2009.

L'objectif majeur des recommandations W3C DOM est donc de fournir une interface neutre vis-à-vis des plateformes et langages des navigateurs client, afin de permettre l'accès et la modification du contenu, structure et style des documents. Le document peut être ainsi modifié et ré-intégré dans la page soumise au client.

1-B - W3C DOM Event

Pour résumer, dès le départ de la gestion d'événement, Netscape et Microsoft ont développé chacun leur modèle spécifique. Devant cette diversité quasiment ingérable pour les développeurs, DOM W3C a sorti un troisième modèle de gestion d'événements (**DOM Level 2 Events Specification**), restant pour beaucoup basé sur le modèle Netscape, et qui fait référence aujourd'hui.

Son objectif est double :

- la conception d'un système d'événements génériques qui permet d'enregistrer des gestionnaires d'événement, de décrire le flux d'événement à travers une structure d'arbre et de fournir des données de contexte de base pour chaque événement
- fournir un sous-ensemble commun aux systèmes d'événements actuels employés dans les navigateurs compatibles DOM niveau 0. L'intention est d'assurer l'interopérabilité des scripts et contenus existants.

Nous verrons ci-dessous que les choses ne sont pas aussi tranchées que nous pourrions l'espérer.

2 - Compatibilité des navigateurs avec le DOM W3C

2-A - Compatibilité DOM

Historiquement, Le DOM Level 1 a été disponible dans sa plus grande partie dès les premières versions d'Internet Explorer 5 et de Netscape 6.

Aujourd'hui, les navigateurs de 5ème génération supportent pour la plupart le DOM W3C Level 3. Pour s'en convaincre, vous pouvez vous rendre sur la page de synthèse **compatibilité des navigateurs** de quirksmode. A noter que W3C fournit également une table de compatibilité mais qui ne prend en compte la compatibilité DOM du navigateur utilisé pour l'accès à la page **html** ou à la page **xhtml** présentée par le site W3C. Bien sûr, le monde n'est pas monochrome ici et l'on doit faire face à une diversité dans le mode d'accès aux éléments DOM selon le navigateur. Mais la convergence vers le modèle W3C progresse. On verra dans le préambule à la gestion des événements DOM comment traiter quelques unes des divergences.

2-B - Compatibilité DOM event

La plupart des navigateurs actuels offrent une relativement bonne comptabilité avec les évènements DOM W3C comme l'atteste la **table** de quirksmode. Mais on ne peut pas dire que le DOM Event soit largement transversal ou Cross-Browser (X-browser), c'est-à-dire implémenté systématiquement sur au moins les deux navigateurs majoritaires sur le marché. Par exemple pour Internet Explorer, qui ne supporte aucun des évènements du standard W3C, essentiellement par choix de stratégique, puisque IE implémente son propre mode de gestion pour ces cas de figures, sans toutefois réfuter le bienfondé de ces événements. Pour la version Windows de IE, il existe cependant des solutions alternatives que nous verrons plus loin dans l'article. (Nota: pas de solutions connues pour la version Mac de IE, qui ne constitue toutefois pas une priorité du fait de l'utilisation plus fréquente de Safari et Opéra dans le cas des plateformes MAC OS X)

3 - Pourquoi utiliser la gestion des évènements DOM ?

Tout d'abord, il convient de noter que DOM et Javascript ne constituent pas un couple unique comme on l'aura compris dans les recommandations de W3C DOM. Ainsi DOM est accessible depuis d'autres langages orientés client que Javascript, comme **ActionScript** d'Abobe, VBA, C++ , **Java** (Sun) ou **Qt** par exemple.

Au-delà des problèmes liés aux spécificités de chaque navigateur, la gestion dite « traditionnelle » des évènements avec les modèles *« HTML inline »* montre plusieurs limites (a fortiori mises en exergue dans les contextes Ajax) :

Que dit en substance la spécification W3C DOM Events Level 2 à ce **sujet** :

En HTML 4.0, les gestionnaires d'événement étaient définis comme des attributs de l'élément. Ainsi l'ajout d'un second événement de même nature remplacerait le premier gestionnaire (exemple de l'événement *click* sur deux éléments distincts d'une page HTML).

A contrario, le modèle W3C DOM Events permet l'enregistrement de plusieurs gestionnaires d'événement sur un même élément cible. Mais pour parvenir à cela, les gestionnaires d'événement ne sont plus définis comme attributs des éléments mais comme objets.

Considérons l'exemple suivant avec le modèle classique *« HTML inline »* (et donc fonctionnel mais, quelque part, obsolète puisque non conforme à la spécification DOM Event) :

```html
<div id="ma_balise" onclick="go_page('pageA.asp')" > ...</div>
```

De façon interne, le navigateur interprètera cette demande comme l'affectation d'une fonction anonyme à l'évènement *onclick*.

Cela pourrait donc aussi s'écrire sous la forme :

```javascript
ref_vers_ma_balise.onclick = function () { go_page('pageA.asp'); };
```

Cette approche présente cependant au moins trois limitations :

- En renvoyant « *false* » depuis toute fonction gérant cette évènement (ici go_page), celle-ci empêcherait le déroulement correct de l'action attendue, comme par exemple le navigateur qui ne pourrait pas suivre un lien s'il détecte l'évènement *click*. (entrée en conflit)

Si une gestion de cet évènement existe déjà mais que l'on souhaite lui ajouter une fonctionnalité par le script, il faudrait recoder la fonction avec cette nouvelle fonctionnalité. Même problème si l'on souhaitait soustraire des fonctionnalités. On voit donc venir la contrainte de devoir écrire sa propre fonction de gestionnaire d'évènements, avec un gestionnaire

d'évènements exécutant tour à tour chacune des fonctions souhaitées : la complexité du code et les erreurs risquent alors de rapidement s'inviter à la fête. Rendant alors le débogage et la maintenance tout aussi complexe. C'est un premier point de passage vers le modèle objet.

- Ce modèle « *HTML inline* » présente aussi l'inconvénient que deux éléments détectent le même évènement. Imaginons un lien hypertexte <a> à l'intérieur d'une balise , pointant tous deux vers l'évènement click : que se passe-t-il alors lors du click ? Le lien <a> doit-il détecter l'évènement ou bien la balise ou bien les deux ? Et dans ce cas, dans quel ordre doit s'effectuer la détection ?

Et bien ces questions nous permettent d'introduire la notion de propagation des événements détaillée plus loin.

- Le modèle « *HTML inline* » impose de mêler le code HTML et le code Javascript. Pour donner une image, cela conduit plutôt vers une programmation « Mikado » que vers une programmation « Lego », même si j'apprécie les deux jeux: le code est alors difficilement maintenable si l'on ne sépare pas clairement deux familles qui doivent se parler sans pour autant se mélanger. Préférez donc le mode « Lego » pour la programmation. (Ref vers design patterns MVC)

4 - L'accès aux éléments DOM

Seront abordées ici en Javascript quelques notions essentielles à la compréhension de la suite de l'article plus axé sur la création d'évènements que sur leur accès.

Les exemples illustratifs ont été testés sous Windows XP avec :

- Mozilla Firefox 3.5.8,
- Internet Explorer 8 (8.0.6001),
- Safari 4.0.2 (530.19.1)
- Opera 10.50 (3296)

4-A - Exemple à partir d'un document HTML:

Comme évoqué, un évènement est lié à un ou plusieurs éléments de la page HTML ou X-HTML. Notons au passage que l'on peut étendre ces notions aux documents XML. Il faut donc pourvoir accéder à un élément du document HTML fourni en exemple simple ci-dessous.

Exemple de page HTML :

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" >
   <head>
       <title>La gestion des évènements DOM</title>
       <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
   </head>

   <body>
        <div id="div_id" name="div_name">
            <p>Voila comment récupérer les objets DOM du document HTML</p>
        </div>
   </body>

   <!-- inclusion du code javascript en fin de page -->
    <script type="text/javascript" src="tuto_DOM_events.js"></script>

</html>
```

Nous verrons trois méthodes d'accès, à utiliser de préférence dans l'ordre présenté :

4-B - Accès par l'identifiant id (document.getElementById)

Cette méthode est **universelle** X-browser et retourne un élément unique, puisque les identifiants sont supposés uniques.

Code tuto_DOM_events.js (placé dans le même répertoire que la page page.html)

```javascript
var id    = "div_id";
var id_balise_div = document.getElementById(id) ;
alert('getElementById retourne: ' + id_balise_div);
```

Remarque concernant IE :
*Pour les versions IE8 (en mode IE8), getElementById réalise une analyse tenant compte de la casse seulement sur l'identifiant id. Par contre, IE8 en mode IE7 ainsi que sur les versions précédentes d'IE, l'analyse porte sur les deux attributs name et id, et sans tenir compte de la casse. Ceci peut donner des résultats inattendus si l'on n'y prend garde. Pour plus de détails, veuillez vous référer à l'article suivant : **Defining Document Compatibility**.*

4-C - Accès par le nom de la balise html, ou tag (document.getElementsByTagName)

Cette méthode est **universelle** X-browser et retourne la collection des éléments contenus dans l'arbre DOM de la balise .

Contrairement au cas des identifiants *id*, les tags n'étant pas uniques, cette méthode retournera la collection d'objets dont le tag vaut « DIV » (c'est-à-dire que seront retournés des objets contenant l'ensemble des balises HTML) dans notre exemple.

Code tuto_DOM_events.js (placé dans le même répertoire que page.html)

```javascript
var tag = "DIV" ; // correspond aux balises HTML <div>
var tag_balise_div = document.getElementsByTagName(tag) ;
alert('getElementsByTagName retourne: ' + tag_balise_div );
```

4-D - Accès par l'attribut name (document.getElementsByName)

Contrairement aux deux méthodes précédentes, on notera que cette méthode n'est pas aussi universelle dans le sens où IE ne retourne pas la même chose que les autres navigateurs. (Voir **msdn**)

Code tuto_DOM_events.js (placé dans le même répertoire que page.html)

```javascript
var name_div= "div_name";
var name_balise_div = document.getElementsByName(name_div) ;
alert('getElementsByName retourne: ' + name_balise_div);
```

Cas spécifique de IE :
Cette méthode employée avec IE retourne les éléments dont la valeur de l'attribut name est «name_div» ou dont la valeur de l'identifiant id vaut « name_div », d'où certaines confusions possibles.
En plus de cette première différence comportementale, il faut aussi savoir que cette méthode, quand elle est employée sous IE, ne retourne pas les éléments des tags SPAN et DIV.

Voyons quatre exemples simples pour s'en convaincre,

On utilisera le code javascript unique suivant pour ces quatre exemples HTML

Code tuto_DOM_events.js

```javascript
var name_div= "div_name";
var name_balise_div = document.getElementsByName(name_div) ;
alert('Nombre de noeuds détectés par IE: ' + name_balise_div.length);
```

Exemple 1 : balise

```html
<body>
    <div name="div_name">
        <p>Voila comment récupérer les objets DOM du document</p>
        <h1>Comment IE réagit-il </h1>
    </div>
</body>
```

IE retourne 0 alors que les quatre autres navigateurs testés retournent tous la valeur 1.

La balise DIV n'est pas comptabilisée par IE.

Exemple 2 : balise

```html
<body>
    <div>
        <p>Voila comment récupérer les objets DOM du document</p>
        <span name="div_name"><h1>Comment IE réagit-il </h1></span>
    </div>
</body>
```

IE retourne 0 alors que les quatre autres navigateurs testés retournent tous la valeur 1.

La balise SPAN n'est pas comptabilisée par IE.

Exemple 3: Attribut *name* dans la balise <h1>

```html
<body>
    <div>
        <p>Voila comment récupérer les objets DOM du document</p>
        <h1 name="div_name">Comment IE réagit-il </h1>
    </div>
</body>
```

Là encore IE retourne 0 sans plus d'explications (**bug ?**)

Exemple 4: Attribut *id* dans la balise <h1>

```html
<body>
    <div>
        <p>Voila comment récupérer les objets DOM du document</p>
        <h1 id="div_name">Comment IE réagit-il </h1>
    </div>
</body>
```

IE retourne la valeur 1. IE a donc récupéré la balise <h1> avec son identifiant *id* avec la méthode *getElementsByName()*.

Quelle(s) parade(s) pour harmoniser le comportement de votre code X-browser ?

On a vu dans les exemples ci-dessus que finalement, le meilleur choix dans le cas d'IE est d'éviter d'utiliser la méthode *getElementsByName()* et de privilégier la méthode *getElementsByTagName()* car elle est peut-être la moins aléatoire des trois possibilités.

Une première parade peut consister dans l'utilisation combinée d'attribut *id* et de récupérer les éléments par la méthode des tags comme illustré dans l'exemple ci-dessous :

html
```
<body>
    <div id="div_id">
        <p>Voila comment récupérer les objets DOM du document</p>
        <h1 id="h1_name0">1ere balise</h1>
        <h1 id="h1_name1">2eme balise</h1>
        <h1 id="h1_name2">3eme balise</h1>
    </div>
</body>
```

javascript
```
function getElementsByRegExpOnId(search_reg, search_element, search_tagName) {
// Rechercher dans les tags d'une page (X)HTML, les IDentifiants correspondant à une expression régulière new Reg
    search_element = (search_element === undefined) ? document : search_element;
    search_tagName= (search_tagName === undefined) ? '*' : search_tagName;
    var id_return = new Array;
    for(var i = 0, i_length = search_element.getElementsByTagName(search_tagName).length; i < i_length;
 i++) {
        if (search_element.getElementsByTagName(search_tagName).item(i).id &&
 search_element.getElementsByTagName(search_tagName).item(i).id.match(search_reg)) {
            id_return.push(search_element.getElementsByTagName(search_tagName).item(i).id) ;
        }
    }
    return id_return; // array
}

var balises_h1 = getElementsByRegExpOnId(/^h1_name.+/, document.body , 'h1');
var h1_array = new Array();
for (i=0; i< balises_h1.length; i++ ) {
    h1_array.push(document.getElementById(balises_h1[i]).id) ;
}

alert('Identifiants des balises H1:\n' + h1_array.join(',\n') );
```

Une seconde parade peut consister dans la réécriture de fonctions prototype. Nous ne fournirons qu'un seul exemple ci-dessous pour la méthode *getElementsByName()* ; sachez donc que c'est possible et malgré un investissement-temps initial, c'est une solution qui peut s'avérer efficace. Vous trouverez quelques informations utiles sur cette **page** du site msdn.

Selon ce principe, une variante de la fonction précédente *getElementsByRegExpOnId ()* pourrait alors s'écrire de la façon suivante (attention, en comparant avec la première alternative, on utilise ici l'attribut *name* et non plus l'attribut *id*):

html
```
<body>
    <div id="div_id">
        <p>Voila comment récupérer les objets DOM du document</p>
        <h1 name="h1_name0">1ere balise</h1>
        <h1 name="h1_name1">2eme balise</h1>
        <h1 name="h1_name2">3eme balise</h1>
    </div>
</body>
```

Code Javascript: réécriture de la méthode *getElementsByName()*

```javascript
// Réécriture de la méthode
if(typeof(window.external) != 'undefined'){
    document.getElementsByName = function(reg_name, tag) {
        if (!tag){
            tag = '*';
        }
        var elems = document.getElementsByTagName(tag);
        var result = [] ;
        for(var i=0 ; i<elems.length ; i++) {
            attrib = elems[i].getAttribute('name');
            if (undefined !== attrib) { // au cas où l'on aurait confondu name et id dans la regex
                if ( attrib.match(reg_name) ) {
                    result.push(elems[i]);
                }
            }
        }
        return result;
    }
}

// Exemple d'utilisation
var search_name  = /h1_name.+/ ;
var search_balise = "h1" ;
var balises_h1 = document.getElementsByName( search_name , search_balise ) ;
alert('Liste des Objets avec la balise h1 et l\'attribut name selon pattern search_name:\n' +
 balises_h1.join(',\n'));
```

4-E - Accès aux attributs des éléments du document :

Section à compléter

4-F - Navigation dans l'arbre DOM

Section à compléter

5 - La gestion des évènements DOM :

5-A - Les 4 interfaces d'événements implémentées par W3C DOM Events

5-A-1 - Événements HTML

Le module (ou interface) d'événements HTML se compose des événements listés dans HTML 4.0 et des autres événements reconnus par les navigateurs DOM niveau 0 (1):

Événement(2)	Élément(s) concerné(s)	Bubbling(3)
load	Document, FRAMESET, OBJECT	Non
unload	BODY et FRAMESET	Non
error	OBJECT, BODY et FRAMESET.	Non
abort	BODY et FRAMESET	Oui
select	INPUT et TEXTAREA	Oui
change	INPUT, SELECT et TEXTAREA.	Oui
submit	FORM	Oui
reset	FORM	Oui
focus	LABEL, INPUT, SELECT, TEXTAREA et BUTTON.	Non
blur	LABEL, INPUT, SELECT, TEXTAREA et BUTTON.	Non
resize	Document	Oui
scroll	Document	Oui

(1): Se rapporter au chapitre sur la compatibilité DOM Event des navigateurs.

(2): Aucun de ces événements ne comporte de données de contexte et n'est annulable.

(3): Se rapporter au chapitre sur le flux des événements.

5-A-2 - Événements de l'interface Utilisateur (UIEvent)

Le module d'événements de l'interface utilisateur se compose des événements listés dans HTML 4.0 et d'autres événements reconnus par les navigateurs DOM niveau 0.

Schématiquement ce sont les événements similaires aux événements HTML mais avec une portée plus large.

Événement(1)	Élément(s) concerné(s)	Bubbling(2)
DOMFocusIn	applicable à tout objet *EventTarget* capable de recevoir l'attention, et pas seulement aux commandes de formulaire d'un élément FORM.	Oui
DOMFocusOut	À la différence de l'événement HTML *blur*, l'événement DOMFocusOut est applicable à tout objet *EventTarget* capable de recevoir l'attention, et pas seulement aux commandes de formulaire d'un élément FORM.	Oui
DOMActivate	se produit quand un élément est activé, par exemple, au travers d'un clic de souris ou de l'appui d'une touche. Un argument numérique est	Oui

	fourni pour indiquer le type d'activation qui se produit.	

1 : Seul DOMActivate est un événement annulable
2 : Se rapporter au chapitre sur le flux des événements.

5-A-3 - Événements de souris (MouseEvent)

Événement(1)	Élément(s) concerné(s)	Bubbling(2)
click	L'événement *click* advient lorsqu'on clique un bouton du dispositif de pointage sur un élément. Un clic est défini comme étant un **cycle d'événements** *mousedown-mouseup* au même endroit de l'écran. La séquence de ces événements est : *mousedown* => *mouseup* => *click*	Oui
mousedown	lorsqu'on presse un bouton du dispositif de pointage sur un élément	Oui
mouseup	lorsqu'on relâche un bouton du dispositif de pointage sur un élément	Oui
mouseover	lorsqu'on déplace le dispositif de pointage sur un élément	Oui
mousemove	lorsqu'on déplace le dispositif de pointage alors qu'il se trouve sur un élément	Oui
mouseout	lorsqu'on déplace le dispositif de pointage hors d'un élément	Oui

(1): Tous ces événements sont de type « bubbling » (bouillonnants), annulables et s'appliquent à la plupart des éléments. Les données de contexte communes à tous ces événements sont les suivantes : screenX, screenY, clientX, clientY, altKey, ctrlKey, shiftKey, metaKey. Les événements « *click* », « *mousedown* » et « *mouseup* » possèdent en plus la donnée de contexte « button »,

5-A-4 - Événements de mutation (MutationEvent):

Non implémentés pour l'instant par DOM Niveau 2 mais le sujet est placé en perspective.

Le module ou interface d'événements de mutation est conçu(e) pour permettre la notification de tout changement intervenant dans la structure d'un document, y compris les modifications des objets Attr et Text.

Événement(1)	Élément(s) concerné(s)	Bubbling(2)
DOMSubtreeModified	événement général pour la notification de tous les	Oui

	changements intervenus sur le document	
DOMNodeInserted	Lancé quand un nœud a été ajouté comme enfant d'un autre nœud	Oui
DOMNodeRemoved	Lancé quand un nœud est retiré de son nœud parent.	Oui
DOMNodeRemovedFromDocument	Lancé quand un nœud est retiré d'un document, qu'il s'agisse d'un retrait direct du nœud ou bien du retrait d'un sous-arbre qui le contient	Non
DOMNodeInsertedIntoDocument	Lancé quand un nœud est inséré dans un document, qu'il s'agisse d'une insertion directe du nœud ou bien de l'insertion d'un sous-arbre qui le contient.	Non
DOMAttrModified	Lancé après qu'un objet Attr a été modifié sur un nœud	Oui
DOMCharacterDataModified	Lancé après que l'objet CharacterData dans un nœud a été modifié, sans que le nœud en question n'ait été inséré ni retiré	Oui

(1): Aucun de ces événements n'est annulable,

(2): Se rapporter au chapitre sur le flux des événements.

Remarques:

- L'interface d'événements de touches clavier n'est pas implémentées par le DOM Niveau 2

- L'interface MouseEvents est une sous-classe héritant de UIEvents.

5-B - Les notions de flux et de propagation d'un événement

5-B-1 - Les 3 phases du flux d'un événement selon les 3 modèles de gestionnaire

5-B-1-A - Rappel de la terminologie W3C DOM Events :

La capture: Le processus selon lequel un événement est manipulé par l'un des ancêtres de la cible d'événement avant que celle-ci ne traite l'événement.

Le bouillonnement (bubbling): Le processus selon lequel un événement se propage en amont vers les ancêtres de la cible d'événement après que celle-ci a traité l'événement.

Annulable : Une caractéristique des événements qui indique au client, lors de la manipulation d'un événement, qu'il peut empêcher la mise en œuvre DOM de poursuivre l'action implicite associée à cet événement.

Je vous encourage à lire les **définitions détaillées** de ces termes.

Quelques Définitions

A la dimension « physique » des nœuds du document HTML s'ajoute une dimension « temps » avec la notion d' , à rapprocher de la notion de *parent* : « Un nœud ancêtre d'un nœud A correspond à tout nœud en amont de A dans le modèle en arbre d'un document, où « en amont » veut dire « en se rapprochant de la racine » Mais un *ancêtre* n'est pas forcément un *parent* au sens DOM du terme.

Même principe pour les *descendants* et les *enfants* (*childNodes*)

Autrement dit, le flux d'événements se décompose donc en trois phases distinctes :

- phase de capture (capturing): comprend tous les nœuds depuis le nœud d'origine (inclus) jusqu'au parent du nœud cible (exclus).
- phase cible (Event target): comprend uniquement le nœud cible.
- phase de bouillonnement (bubbling) : c'est la propagation retour vers le haut de l'événement. Elle comprend les nœuds rencontrés lors du cheminement du parent du nœud cible (exclus) jusqu'au nœud d'origine (inclus pour boucler la boucle). C'est le mode de détection le plus commun.

5-B-1-B - Exemple de flux

Exemple HTML

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" >
   <head>
       <title>La gestion des évènements DOM</title>
       <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
   </head>

<body>
    <div>
    <table border="2">
        <tbody>
        <tr><td>Hello</td><td>World</td></tr>
        <tr><td>VALIDER</td><td>ANNULER</td></tr>
        </tbody>
    </table>

    </div>
</body>
```

Imaginons maintenant que nous attachions un événement clic à la cellule TD correspondante au nœud texte VALIDER par une méthode que nous verrons plus loin.

Schéma DOM et flux des évènements

Nota : le modèle W3C permet quant à lui d'utiliser au choix l'un des deux modes : Capture ou bouillonnement

5-B-2 - La propagation

La progression du guetteur d'évènement d'un élément à l'autre depuis le sommet de l'arbre vers la cible d'événement est appelé la propagation.

Il faut noter ici la différence entre les 3 modèles pour le mode de propagation de l'évènement :

Le modèle Netscape est basé sur le flux de capture (depuis le sommet vers la cible)

Le modèle Microsoft est basé sur le flux de bouillonnement (depuis la cible vers le sommet)

Le modèle W3C DOM Niveau 2 donne le choix entre ces deux modes.

5-C - les gestionnaires d'événements

Le gestionnaire d'événement sert à pouvoir intercepter un événement pendant sa propagation dans l'arbre DOM et lui affecter une fonction de gestion.

Comme évoqué précédemment, il existe 3 modèles DOM différents :

- *Le W3C DOM niveau 0 (héritage Netscape)*
- *(non supporté par Internet Explorer jusqu'à la version 8 incluse).*
- *Le* **modèle Microsoft** *: Internet Explorer fournit en réalité les fonctionnalités du DOM niveau 2, mais avec des dénominations différentes.*

5-C-1 - Le modèle W3C DOM Niveau 2 Event

Ce modèle permet d'attacher plusieurs gestionnaires d'événement à un même événement (*click* par exemple) mais également de spécifier dans de quelle façon (capture, bouillonnement) ces gestionnaires doivent voir l'événement.

A noter que dans le cas de plusieurs attachements du même événement au même élément avec des fonctions de gestion différentes (gestionnaires), l'ordre de détection ne peut pas être spécifié.

Le DOM se réfère au(x) fonction(s) du gestionnaire d'événement par les écouteurs d'événement de la façon suivante :

```javascript
referenceVersUnElement.addEventListener('typeEvenement', referenceVersFonction, phase);
referenceVersUnElement.removeEventListener('typeEvenement', referenceVersFonction, phase);
```

Descriptif :

referenceVersUnElement :

l'objet DOM auquel on veut attacher un gestionnaire d'événement

addEventListener ou *removeEventListener* :

Cette méthode permet d'enregistrer ou supprimer des guetteurs d'événement sur la cible d'événement.

typeEvenement :

le type d'événement à guetter (écouter) parmi les types disponibles (par exemple: *click*).

Remarque: *typeEvenement* est à écrire en minuscule

referenceVersFonction :

La fonction qui sera exécutée lorsque la cible sera atteinte.

Selon la spécification, *referenceVersFonction* doit pointer vers un objet gérant la méthode *handleEvent* . En d'autres termes, le premier paramètre de la fonction sera un objet de type *event* .

Remarque: contrairement au modèle traditionnel, on peut pas lister les fonctions du gestionnaire (par *referenceVersUnElement.ontypeEvenement*) car W3C DOM Event Niveau 2 ne fournit cette possibilité. Cela devrait être corrigé avec W3C DOM Event Niveau 3 qui a prévu d'offrir cette possibilité.

phase :

le mode de détection (capture si la phase est égale à *true* / bouillonnement si la phase est égale à *false*). Par défaut et par continuité du modèle traditionnel, ce mode est positionné à false (bouillonnement). En effet, la détection par capture est particulière : en se référant à la spécification W3C, le gestionnaire d'événement n'est lancé que si l'événement se produit sur l'ancêtre de la cible.

5-C-2 - Le modèle Microsoft

IE met a disposition des méthodes similaires à celles de *W3C DOM Event Level 2* sauf que :

- la phase n'est pas précisée et représente est obligatoirement le mode de détection par bouillonnement
- le type d'événement *typeEvenement* doit être précédé de la préposition « ***on*** » (par exemple *onclick* au lieu de *click*)
- Les événements supportés sont de type classiques comme les événements HTML ou de type « souris » par exemple. Par contre, IE ne respectant pas le modèle W3C DOM Event, les événements de type Mutation (**MutationEvent**) ou Utilisateur (**UIEvent**) ne sont pas supportés.

javascript
```
referenceVersUnElement.attachEvent('ontypeEvenement',referenceVersFonction);
referenceVersUnElement.detachEvent('ontypeEvenement',referenceVersFonction);
```

5-C-3 - Exemples illustratifs

code HTML commun aux exemples suivants :

html
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" >
   <head>
       <title>La gestion des évènements DOM</title>
       <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
   </head>

<body>
    <div id="ma_div">
        <a href="/lib/tuto_dvp/page.html" id="mon_lien">test Events par capture ou bouillonnement</a>
    </div>
</body>

   <!-- inclusion du code javascript en fin de page -->
    <script type="text/javascript" src="tuto_DOM_events.js"></script>

</html>
```

5-C-3-A - Exemple 1: Modes de détection capture ou bouillonnement selon le modèle W3C

code Javascript

javascript
```
// Recupération des éléments
    var oDiv    = document.getElementById('ma_div');

    var oLien= document.getElementById('mon_lien');
```

javascript
```
// Ajout des listeners avec le même type événement "click"

    // selon W3C DOM Niv 2

        oDiv.addEventListener('click',function (event) {
          alert('1- Capture de la DIV');
        },true);

        oLien.addEventListener('click',function (event) {
          alert('2- Capture du lien A (non conformité W3C)');
        },true);

        oLien.addEventListener('click',function (event) {
          alert('3- Bouillonnement du lien A');
        },false);

        oDiv.addEventListener('click',function (event) {
          alert('4- Bouillonnement de la DIV');
        },false);
```

Explications et remarques :

1 Cet exemple est donc applicable dans le contexte des navigateurs compatibles W3C DOM Event Level 2. Dans notre cas, l'exemple est donc testé avec les 3 navigateurs Mozilla Firefox 3.5.8, Internet Explorer 8 (8.0.6001), Safari 4.0.2 (530.19.1) et Opera 10.50 (3296).
2 Notons une différence comportementale entre la spécification W3C et le comportement effectif des navigateurs :

Selon la spécification, le flux devrait respecter cette chronologie :

1- Capture de l'écouteur d'événement sur la cible DIV

2- Bouillonnement de l'écouteur d'événement sur la cible « lien A »

3- Bouillonnement de l'écouteur d'événement sur la cible « DIV »

Or on observe que le lien A est capturé alors que l'attribution d'un gestionnaire d'événement sur le lien A devrait, selon la spécification, n'être lancé que si l'événement se produit sur l'ancêtre (DIV) de la cible (A).

Il vaut donc mieux éviter de se baser sur la phase de capture d'autant plus que la spécification W3C DOM Event Niveau 3 corrigera ce point. C'est d'ailleurs également le choix du modèle Microsoft.

5-C-3-B - Exemple 2: Mode de détection selon le modèle Microsoft (bouillonnement)

javascript
```
// Recupération des éléments
    var oDiv    = document.getElementById('ma_div');

    var oLien= document.getElementById('mon_lien');

// Ajout des listeners avec le même type événement on "click"

    // selon le modèle Microsoft

        oLien.attachEvent('onclick', function (event) { alert('1- Bouillonnement du lien A'); } );

        oDiv.attachEvent('onclick', function (event) { alert('2- Bouillonnement de la DIV');} );
```

Remarque:

L'ordre dans lequel on affecte les gestionnaires dans le code n'a pas d'importance.

5-C-3-C - Exemple 3: Transversalité du code aux deux modèles W3C et Microsoft

javascript
```
// Recupération des éléments
    var oDiv    = document.getElementById('ma_div');

    var oLien= document.getElementById('mon_lien');

// Ajout des listeners uniquement avec une phase Bouillonnement et avec le même type evenement "onclick"

    // selon le modèle Microsoft
        if (oLien.attachEvent)
            oLien.attachEvent('onclick', function (event) {
 alert('Modele MS: Bouillonnement du lien A'); } );

    // Modele W3C DOM Event Level 2
        if (oLien.addEventListener)
            oLien.addEventListener('click',function (event) {
 alert('Modele W3C: Bouillonnement du lien A');}, false);
```

Remarque :

Notre version d'Opera (10.50) implémente ces deux modèles de gestionnaire. Donc pour une transversalité plus complète par rapport aux navigateurs du marché, il faudrait tenir compte de son user-agent.

5-D - L'objet Event

Les propriétés et méthodes de l'objet Event dépendront du type d'événement sur lequel porte le gestionnaire.

Dans un premier temps, concentrons-nous spécifiquement sur les méthodes et propriétés de l'interface Event du modèle W3C DOM 2 Event et de leur alter-ego du modèle Microsoft quand ils existent :

5-D-1 - Propriétés et Méthodes W3C

Propriétés W3C (objet event)	Description W3C	Équivalent Microsoft (objet window.event)
bubbles	Sert à indiquer s'il s'agit ou non d'un événement de bouillonnement. Si l'événement peut bouillonner, la valeur est true, sinon false.	non
cancelable	Sert à indiquer si l'action implicite d'un événement peut être empêchée, ou non. Si c'est le cas, la valeur est true, sinon false.	non
currentTarget	Sert à désigner la cible **EventTarget** dont le traitement des guetteurs **EventListener** est en cours. Cela est particulièrement	non

	utile pendant la capture et le bouillonnement.	
eventPhase	Sert à indiquer quelle phase du flux d'événement est en cours d'évaluation.	non
target	Sert à désigner la cible **EventTarget** vers laquelle l'événement a été distribué à l'origine.	srcElement
timeStamp	Sert à indiquer l'heure (en millisecondes par rapport à l'époque) à laquelle l'événement a été créé. Comme certains systèmes ne fournissent pas cette information, la valeur de l'attribut timeStamp n'est pas forcément disponible pour tous les événements. A considérer avec précaution.	non
type	Le nom de l'événement (insensible à la casse). Ce doit être un .	type

Méthodes W3C (objet event)	**Description W3C**	**Équivalent Microsoft (objet window.event)**
initEvent	La méthode *initEvent* sert à initialiser la valeur d'un objet Event créé au travers de l'interface **DocumentEvent**. On ne peut appeler cette méthode qu'avant la distribution de Event via la méthode *dispatchEvent*, quoiqu'on puisse l'appeler plusieurs fois pendant cette phase si nécessaire. Si on l'appelle plusieurs fois, la dernière invocation prime. Si on l'appelle à partir d'une sous-classe de Event, seules les valeurs définies dans la méthode *initEvent* sont modifiées, et tous les autres attributs restent inchangés.	non
preventDefault	Si un événement est annulable, on utilise la méthode *preventDefault* pour indiquer que l'événement doit être annulé, c'est-à-dire que toute action implicite entreprise normalement par la mise en œuvre en conséquence de l'événement ne devra pas se produire. Si, pendant une étape du flux d'événement, on appelle	returnValue c'est une propriété à placer à false pour stopper tout traitement par défaut de l'événement.

	la méthode *preventDefault*, l'événement est annulé. Toute action implicite associée à l'événement ne se produira pas. Pour un événement non annulable, l'appel de cette méthode n'a aucun effet. Une fois la méthode *preventDefault* appelée, elle restera active pour le restant de la propagation de l'événement. On peut appeler cette méthode à n'importe quelle étape du flux d'événement.	
stopPropagation	La méthode *stopPropagation* sert à empêcher la propagation ultérieure d'un événement pendant le flux d'événement. Si un guetteur **EventListener** appelle cette méthode, l'événement cessera de se propager dans l'arbre. La distribution de l'événement à tous les guetteurs se terminera sur l'objet **EventTarget** courant, avant l'interruption du flux d'événement. Cette méthode peut servir à toute étape du flux d'événement.	cancelBubble Propriété à mettre à true pour stopper la remontée de l'événement.

5-D-1-A - Exemple 0 : interfaces d'événement(s) supportées (W3C: hasFeature)

Écrivons un petit code Javascript afin de déterminer les interfaces d'événements supportées par le navigateur qui exécute le code.

Utilisons la méthode suivante :

document.implementation.hasFeature(feature, version)

La méthode retourne true ou false selon que l'interface testée sur le navigateur est supportée ou non.

avec les arguments :

feature (argument obligatoire):

Le nom de l'interface que l'on veut tester, à sélectionner parmi la liste de conformité W3C DOM 2.

version (argument facultatif) :

La version DOM à tester. Si l'argument est vide, la méthode retourne true si l'interface est supportée par tous les niveaux DOM

exemple :

var interface_supportee = document.implementation.hasFeature('Events', '2.0');

javascript
```
var DOMEventInterfaceArray = new Array(
  /* DOM1 */    '_DOM1', 'HTML', 'XML',
  /* DOM2 */
  '_DOM2', 'Core', 'Views', 'StyleSheets', 'CSS', 'CSS2', 'Events', 'UIEvents', 'MouseEvents', 'HTMLEvents', 'Ran
  /* DOM3 */    '_DOM3', 'TextEvents', 'MutationEvents', 'LS-Load', 'LS-Load-async', 'LS-Save', 'VAL-
DOC', 'XPath',
  /* SVG */    '_SVG', 'http://www.w3.org/TR/SVG11/feature#CoreAttribute' /* à compléter */
   );

var DOMLevel = new Array( '', '1.0', '1.1', '1.2', '2.0', '3.0', '4.0' );
var minireg = /_DOM|_SVG/g;

function tbody(content)    { return '<table border="1" cellspacing="1">\n'+content+'</table>\n'; }
function th(content,_class){ return '<th '+_class+'>'+content+'</th>'; }
function td(content,_class){ return '<td '+_class+'>'+content+'</td>'; }
function tr(content)    {
    var style= content.match(minireg)?' style="background:gray";>':'>' ;
    return '<tr' + style + content +'</tr>\n';
    }

function vers(f){
  var i,s=th(f,'class="none"');
  for(i=0; i<DOMLevel.length; i++){
    var r=(document.implementation.hasFeature)? document.implementation.hasFeature(f,DOMLevel[i]):
 window.f;//implementation.hasFeature(f,DOMLevel[i]);
    s+=!f.match(minireg)?td(r,r?'style="color:blue"':'style="color:red"'):td('','');
  }
  return tr(s);
}

function DOMTable(){
  var s=tr(th('','class="c"')+ th('version', 'colspan="'+DOMLevel.length+'"'));
  var c=th('DOMEventInterfaceArray','class="none"');
  for(var i=0; i<DOMLevel.length; i++){ c+=th(DOMLevel[i]==''?'n/d' : DOMLevel[i],''); }
  s+=tr(c);
  for(var i=0; i<DOMEventInterfaceArray.length; i++){ s+=vers(DOMEventInterfaceArray[i]); }
  return tbody(s);
}
document.write(document.implementation?  DOMTable(): '<p>Navigateur non supporté<\/p>\n'  );
```

Avertissements :
Comme précisé à cette ***adresse*** *, la méthode hasFeature (W3C DOM Core) comporte au moins trois limitations importantes :*
1) Il est plus fiable de tester la compatibilité avec un type d'événement précis plutot qu'avec son interface. Par exemple, il vaut mieux tester 'Events.click' que seulement son interface 'Events'.
2) Comme la spécification DOM n'impose pas de compatibilité ascendante des noms d'interfaces, la méthode hasFeature peut retourner false pour des versions antérieures de DOM et true pour la version courante. Par exemple, la méthode testée pour l'interface 'Events ' sur un navigateur compatible DOM Niveau 2 indiquera qu'il ne l'est pas sur cette interface au Niveau 1
3) Du fait des deux points précédents, la méthode est fiable qualitativement si et seulement si elle retourne true.

5-D-1-B - Exemple 1 : propriétés W3C

Cherchons à récupérer les propriétés W3C en tant que valeurs ou objets de premier niveau de l'objet *event*

html
```
<body>
    <div id="ma_div">
```

html
```
        <p id="noeud_texte">Ceci est un n□ud texte au sein de la DIV - cliquez dessus</p>
    </div>
</body>
```

javascript
```
// Récupération des éléments
    var oText    = document.getElementById('noeud_texte');

// Ajout des listeners avec une phase Bouillonnement (phase==false sur modèle W3C)

    // Modèle Microsoft
        if (oText.attachEvent) {
            oText.attachEvent('onclick', ShowCoreProperties );
        }

    // Modèle W3C DOM Event Level 2
        if (oText.addEventListener) {
            oText.addEventListener('click', ShowCoreProperties, false);
        }

Array.prototype.in_array = function(searched_val) {
    for(var i = 0, l = this.length; i < l; i++) {
        if(this[i] == searched_val) {
            return true;
        }
    }
    return false;
}

function EpochToHuman(timestamp) {
    var theDate = new Date(timestamp * 1000);
    return theDate.toLocaleString();
}

function ShowCoreProperties(event) {
    var W3CCoreProperties
  = new Array('target', 'type', 'bubbles', 'cancelable', 'currentTarget', 'eventPhase', 'timeStamp' );
    var tmp = new Array();
    for (var prop in event) {
        if (W3CCoreProperties.in_array(prop)) {
                if (prop == 'timeStamp')
                    var that = EpochToHuman(event[prop]);
                else {
                    var that = event[prop];
                }
                tmp.push(prop + ' = ' + that );
        }
    }
    alert('Event Core Properties\n'+ tmp.join(', \n'));


}
```

Si l'on teste ce bout de code, on se rend compte que ces navigateurs renvoient les valeurs suivantes :

Propriété testée	Valeur retournée par Mozilla	Valeur retournée par IE (non compatible W3C)

	(compatible W3C)	
target	[object HTMLParagraphElement],	-
cancelable	true,	-
currentTarget	[object HTMLParagraphElement],	-
timeStamp	13/04/82 18:57	-
bubbles	true	-
type	click	click
eventPhase	2	-

Explications

La fonction *EpochToHuman* vient agrémenter l'affichage du timestamp unix pour afficher une date lisible,

La méthode in_ *array* vient surcharger l'objet Array afin de pouvoir tester si la propriété « prop » retournée par l'événement « event » fait partie des propriétés W3C stockées dans le tableau « W3CCoreProperties ».

La fonction ShowCoreProperties linéarise (seulement le premier niveau) de l'objet event dans un tableau temporaire tmp que l'on affiche par la fonction alert.

Interprétons les résultats retournés :

target et *currentTarget*

Le retour de la cible nous permettra d'accéder à l'élément déclencheur de l'événement par la méthode *event.target.*

cancelable

l'événement est annulable: c'est bien le cas de cet evenement de type click selon la specification W3C

bubbles

l'événement est bouillonnant. C'est normal puisqu'on l'a demandé lors de l'affectation de l'événement au nœud texte par la valeur false de la phase.

type

Le gestionnaire a été déclenché par un événement de type *click*

eventPhase

La phase vaut 0 pour un événement créé manuellement mais encore inactif

La phase vaut 1 pour la phase de capture

La phase vaut 2 pour la phase de bouillonnement sur l'élément cible (le nœud texte)

La phase vaut 3 pour la phase de bouillonnement sur les ancêtres de la cible

On est bien dans le cas de *eventPhase* =2

5-D-1-C - Exemple 2 : Référencement de l'événement déclencheur du gestionnaire

Conservons le même code HTML que précédemment et cherchons à identifier la source de déclenchement (l'élément DOM) du gestionnaire d'événement (ici notre fonction *ShowEventSource*), avec les deux modèles W3C et Microsoft :

javascript
```
// Récupération des éléments
    var oText    = document.getElementById('noeud_texte');

// Ajout des listeners avec une phase Bouillonnement (phase==false)

    // Modèle Microsoft
        if (oText.attachEvent) {
            oText.attachEvent('onclick', ShowEventSource );
        }

    // Modèle W3C DOM Event Level 2
        if (oText.addEventListener) {
            oText.addEventListener('click', ShowEventSource, false);
        }

function ShowEventSource(event) {
    var elmt;
    var event = event || window.event;            // W3C ou MS
    var la_cible = event.target || event.srcElement;    // W3C ou MS
    if (la_cible.nodeType == 3)            // déjouer le bug Safari
        elmt = la_cible.parentNode;
    else
        elmt = la_cible.tagName;

    alert('Cible de l\'événement: ' + elmt);

}
```

5-D-1-D - Exemple 3 : Référencement d'un événement sur une cible avec n□uds enfants selon les deux modèles W3C et Microsoft

Cherchons à référencer un nœud cible de type texte (balise <p>) qui comporte lui-même des enfants (balises <b>, <i> et)

html
```
<body>
    <div id="ma_div">
        <p id="noeud_texte">Ceci <b>est</b> un <i>noeud texte</i> <span>au sein de la DIV - cliquez
 dessus</span></p>
    </div>
</body>
```

javascript
```
// Récupération des éléments
    var oText    = document.getElementById('noeud_texte');

// Ajout des listeners avec une phase Bouillonnement (phase==false)

    // Modèle Microsoft
        if (oText.attachEvent) {
            oText.attachEvent('onclick', ShowEventSource );
        }

    // Modèle W3C DOM Event Level 2
        if (oText.addEventListener) {
            oText.addEventListener('click', ShowEventSource, false);
        }
```

```javascript
function ShowEventSource(event) {
    var elmt;
    var event = event || window.event  ;                // W3C ou MS
    var Elt_Clic = event.currentTarget || event.srcElement.parentNode;     // W3C ou MS
    var la_cible = event.target || event.srcElement;            // W3C ou MS
    elmt = la_cible.tagName;
    var Parent_cible = Elt_Clic.parentNode.tagName || Elt_Clic.tagName ;// W3C ou MS
    alert('Type d événement: ' + event.type + '\nCible: ' + Elt_Clic.tagName + '\nElément cliqué ' +
 elmt + '\nParent de la cible: ' + Parent_cible );
}
```

Explications :

Le code est donc fonctionnel à la fois sur les navigateurs compatibles W3C et sur IE grâce aux tests *OU* (Alt Gr + 6)sur chaque objet à récupérer: *var C = A ou B* donne A s'il existe ou donne B s'il existe, (ou donne A si A et B existent tous les deux).

- Que se passe-t-il pour le modèle W3C

- On attache le gestionnaire *ShowEventSource* déclenché sur l'événement *click* bouillonnant à la balise <p> récupérée par son identifiant *id*
- Le gestionnaire récupère l'événement *event* à la source de son activation (objet *event*)
- On utilise ensuite simplement les propriétés et méthodes W3C DOM. Vous noterez la différence entre les propriétés target et currentTarget.

- Que se passe-t-il pour le modèle Microsoft

Un exemple illustratif complémentaire avec des layers et des balises imbriquées :

Remarque: l'équivalent de la propriété W3C relatedTarget est découpé en deux propriétés Microsoft: fromElement et toElement .

5-D-1-E - Exemple 4 : Inhiber un lien (W3C: preventDefault, IE: returnValue)

Considérons un exemple simple pour inhiber le clic sur un lien hypertexte.

```html
<body>
    <div id="une_div">
        <a href="/" id="mon_lien">test</a>
    </div>
<body>
```

```javascript
var oLien = document.getElementById('mon_lien');
oLien.addEventListener('click', function (e) {
  alert('Le click sur le lien est rendu inactif. La propriété W3C Cancellable= '+e.cancellable);
  e.preventDefault();
},false);
```

L'adaptation au modèle Microsoft est simple en utilisant la propriété adéquate (***returnValue***).

5-D-1-F - Exemple 5 : Inhiber un bouton SUBMIT (W3C: preventDefault, IE: returnValue)

On va prendre l'exemple de ce qu'il ne faut pas faire car il permet de bien illustrer la propriété W3C et au passage de montrer les choses à proscrire absolument, bien qu'elles foisonnent sur Internet.

On peut imaginer dans le cas d'un formulaire « mal rempli » d'inhiber l'action par défaut du bouton SUBMIT. Prenons l'exemple de l'utilisateur qui rentre des balises HTML ou bien des instructions SQL.

A noter que :

- ces contrôles sont à réaliser (ou à doubler) impérativement côté serveur (PHP par exemple), et non côté client (Javascript par exemple), c'est donc un mode de réalisation « côté client » uniquement à des fins d'illustration de la propriété W3C *preventDefault.*
- on trouve souvent le cas des formulaires « validés » suite à un événement *click* sur le bouton SUBMIT (quand ce n'est pas sur un élément texte ou un lien): c'est une méthode également à proscrire, car l'utilisateur peut très bien appuyer sur la touche ENTREE au lieu de cliquer sur le bouton SUBMIT, auquel cas, le gestionnaire n'est pas déclenché. Donc un contrôle sur chaque champ INPUT est a priori plus adapté, tel qu'implémenté dans cet exemple illustratif.

html
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" >
   <head>
       <title>La gestion des évènements DOM</title>
       <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
   </head>

<body>
    <div id="ma_div">
        <form id="my_form" action="/lib/tuto_dvp/page.html" method="POST">

  <input id="input_1" value="premiere valeur par defaut" size="50" maxlength="30" tabindex="10"/><br/>

  <input id="input_2" value="seconde valeur par defaut" size="50" maxlength="30" tabindex="20"/><br/>

  <input id="submit" type="submit" value="ENVOYER"tabindex="30"/><input id="reset" type="reset" value="ANNULER" t
        </form>
    </div>
</body>

   <!-- inclusion du code javascript en fin de page -->
    <script type="text/javascript" src="tuto_DOM_events.js"></script>

</html>
```

javascript
```
function AddEvent(id, evt_type, ma_fonction, phase) {
    var oElt = document.getElementById(id);
    // modèle W3C mode bubbling
    if( oElt.addEventListener ) {
        oElt.addEventListener(evt_type, ma_fonction, phase);
    // modèle MSIE
    } else if( oElt.attachEvent ) {
        oElt.attachEvent('on'+evt_type, ma_fonction);
    }
    record_listeners.push(oElt); // enregistreur d'event

    return false;
}

function DelEvent(id, evt_type, ma_fonction, phase) {
    var oElt = document.getElementById(id);
    // modèle W3C mode bubbling
    if( oElt.removeEventListener ) {
        oElt.removeEventListener(evt_type, ma_fonction, phase);
    // modèle MSIE
    } else if( oElt.detachEvent ) {
        oElt.detachEvent('on'+evt_type, ma_fonction);
    }
    record_listeners.splice(oElt); // enregistreur d'event
```

javascript

```
    return false;
}


// Affectation du gestionnaire ControlData aux champs inputs sur un événement "change"
    record_listeners= new Array();
    var status_submit = true;
    mes_inputs = new Array(); // var globale
    mes_inputs = getElementsByRegExpOnId(/^input_.+/, document , 'input') ;
    if (mes_inputs) {
        for (i=0; i< mes_inputs.length; i++ ) {
            var chaque_input = document.getElementById( mes_inputs[i] ) ;
            AddEvent(chaque_input.id, 'change', ControlData, false);
        }
    }

// Affectation du gestionnaire ControlData aux champs inputs sur un événement "submit"
    AddEvent('submit', 'click', ControlData, false);


function ControlData(event) {
 // NOTA1: contrôle non fiable coté client, et donc à réaliser ou à doubler du côté serveur (PHP)
 // NOTA2 DOM-0: "this" represente 'event'.
 // Ainsi, "this.id" et "event.target.id" sont équivalents sous Firefox
 // (mais pas sous IE qui ne connait pas la propriété target mais srcElement)
    var obj    = event.srcElement || event.target; // IE ou W3C
    var flag_probleme=false;

    // Contrôle individuel à la volée
    if (event.type == 'change') {
        var input_value    = obj.value ;
        var checked_value = htmlspecialchars(input_value);
        if ( input_value == null ) alert('Champ vide');
        if ( input_value != checked_value || input_value == null) {
            obj.value = checked_value;
            flag_probleme= true;
        }
    }
    // Contrôle collectif
    else if (event.type == 'click') {
        for (i=0, nb_inputs=mes_inputs.length ; i< nb_inputs; i++ ) {
            var chaque_input = document.getElementById( mes_inputs[i] ) ;
            var checked_value = htmlspecialchars(chaque_input.value);
            if ( chaque_input.value == null ) alert('Champ vide');
            if ( chaque_input.value != checked_value || chaque_input.value == null ) {
                chaque_input.value = checked_value;
                flag_probleme= true;
            }
        }
    }

    if (flag_probleme && status_submit) {
        alert('Balises HTML, Cmdes SQL non admises => BLOCAGE SUBMIT');
        AddEvent('submit', 'click',
 StopSubmit, false); // on inhibe le submit une seule fois même si plusieurs problemes
        AddEvent('reset', 'click', ReactiverSubmit, false);
    }

    return false;
}

function StopSubmit(evt) {
// Inhiber l'action par défaut du SUBMIT
// (nota: pour un submit, un simple return false aurait suffi :-)

    if( evt.preventDefault ) { evt.preventDefault(); }
    evt.returnValue = false;

    DelEvent('submit', 'click', ControlData, false);
```

javascript
```
    status_submit = false;

    alert('le bouton est inhibé\n\ncliquez sur ANNULER pour le réactiver');

    return false;
}


function ReactiverSubmit(evt) {

    alert('réactivation du click sur le bouton SUBMIT');

    // On supprime tt simplement le gestionnaire ajouté lors de la désactivation du bouton SUBMIT
    DelEvent('submit', 'click', StopSubmit, false) ;

    // On supprime l'action de réactivation possible par le bouton RESET
    DelEvent('reset', 'click', ReactiverSubmit, false) ;

    status_submit =true ;

    return false;
}

function htmlspecialchars(ch) {
// redondance locale js (à doubler en PHP)

    // carac html
    ch = ch.replace(/&/g,"&amp;") ;
    ch = ch.replace(/\"/g,"&quot;") ;
    ch = ch.replace(/\'/g,"&#039;") ;
    ch = ch.replace(/</g,"&lt;") ;
    ch = ch.replace(/>/g,"&gt;") ;
    ch = ch.replace(/=/g,"&eq;") ;

    // requetes SQL
    ch = ch.replace(/INSERT/gi, "--");
    ch = ch.replace(/DELETE/gi, "--");
    ch = ch.replace(/UPDATE/gi, "--");

   return ch;
}
```

Explications

AddEvent : fonction de création de gestionnaire d'un événement *evt_type* sur un identifiant *id*

DelEvent : fonction de suppression de gestionnaire d'un événement *evt_type* sur un identifiant *id*

htmlspecialchars : fonction de remplacement de balises html par leur code caractère,

On commence par attacher un gestionnaire de contrôle des données sur les champs input et sur le bouton SUBMIT.

On vérifie le format des données des champs input pour lesquels on compare la valeur avant et après transformation par la fonction *htmlspecialchars* . S'il y a une différence, c'est que l'utilisateur a tenté de rentrer des balises HTML dans un des champs INPUT. Dans ce cas, on identifie cette situation comme un problème.

Si un problème est détecté, on inhibe le bouton SUBMIT du formulaire et on ajoute la possibilité de sa réactivation si l'on clique sur le bouton ANNULER.

Lors de la réactivation du bouton SUBMIT, on supprime les gestionnaires que l'on avait ajouté pour l'occasion.

5-D-1-G - Exemple 6 : Supprimer l'appel d'un gestionnaire lors de son exécution (W3C, IE)

Prenons le cas d'une alerte que l'on ne veut afficher qu'une seule fois sur un événement *mouseover* sur un élément texte par exemple.

Utilisons les petites fonctions d'ajout/suppression de gestionnaire d'événements vues dans l'exemple 5, compatibles pour les deux modèles W3C et MSIE.

Voyons deux méthodes différentes, toutes les deux compatibles avec chacun des des deux modèles W3C et MSIE, pour atteindre cet objectif, décrites dans les commentaires du gestionnaire *action_du_gestionnaire* déclenché sur l'événement *mouseover* :

html
```
<body>
    <div id="ma_div">
        <p>Lorem ipsum dolor sit amet, consectetuer adipiscing elit. </p>
        <p id="mon_texte_declencheur">Un <b>premier</b> survol lancera une alerte mais pas
 un <b>second</b> survol</p>
        <p>Etiam pede lectus, facilisis sit amet, varius a, malesuada varius, nisl. Aenean aliquam,
 odio quis semper cursus, nisl lacus rutrum ipsum, a laoreet neque ante vitae tortor. </p>
    </div>
</body>
```

javascript
```
function AddEvent(id, evt_type, ma_fonction, phase) {
    var oElt = document.getElementById(id);
    // modèle W3C mode bubbling
    if( oElt.addEventListener ) {
        oElt.addEventListener(evt_type, ma_fonction, phase);
    // modèle MSIE
    } else if( oElt.attachEvent ) {
        oElt.attachEvent('on'+evt_type, ma_fonction);
    }
    return false;
}

function DelEvent(id, evt_type, ma_fonction, phase) {
    var oElt = document.getElementById(id);
    // modèle W3C mode bubbling
    if( oElt.removeEventListener ) {
        oElt.removeEventListener(evt_type, ma_fonction, phase);
    // modèle MSIE
    } else if( oElt.detachEvent ) {
        oElt.detachEvent('on'+evt_type, ma_fonction);
    }
    return false;
}

    var mon_id = 'mon_texte_declencheur';
    var le_type_evt = "mouseover";
    var flux_evt = false; // bouillonnement
    var action_du_gestionnaire = function(e) {
        alert('evt mouseover sur la balise <p>');

        // 1ère méthode : DOM Lev 2
        // W3C
            if ( e.target )
                e.target.removeEventListener(le_type_evt, arguments.callee, flux_evt);
        // MSIE
            else if ( e.srcElement )
                e.srcElement.detachEvent('on'+le_type_evt, arguments.callee);

        // 2ème méthode DOM Lev2
        // DelEvent(mon_id, le_type_evt, action_du_gestionnaire, flux_evt);
    };
```

javascript
```
    AddEvent(mon_id, le_type_evt, action_du_gestionnaire, flux_evt);
```

Explications

L'avantage de la première méthode (*arguments . callee*) est qu'elle est utilisable sans avoir à connaître forcément le gestionnaire initialement affecté à l'élément. Dans le cas d'une affectation multiple de fonctions anonymes déclenchées par un même événement sur un même élément, cela peut s'avérer pratique sans avoir avoir à gérer spécifiquement l'origine de cette fonction.

Nota :

La méthode *arguments . callee* est dépréciée depuis Javascript 1.4 en tant que propriété de *Function.arguments* , et devient une propriété de la variable locale arguments propre à chaque fonction. Mais l'utilisation de la méthode *arguments.callee.caller* (non standard selon ECMA3 mais non dépréciée et utilisée par la majorité des navigateurs) reste possible car elle fait appel à la propriété objet des fonctions *Function.caller* . (alors que *arguments.callee* ne donnera qu'une référence vers la fonction active).

En résumé :

1 *arguments.caller* est dépréciée en faveur de *Function.caller* et n'est pas implémentée sur Firefox 3 par exemple,
2 *Function.caller* n'est pas un standard ECMA3 mais elle est implémentée chez les navigateurs majoritaires,

Référence page 72: **http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf**

5-D-1-H - Exemple 7 : Empêcher les éléments enfants de réagir (W3C: stopPropagation, IE: cancelBubble)

Reprenons le principe du code HTML de l'exemple 3 et ajoutons lui un lien hypertexte.

Affectons un gestionnaire de l'événement *click* sur chacun des éléments DOM.

Puis bloquons la propagation de cet événement afin d'empêcher la détection de cet événement sur les nœuds enfant du nœud DIV.

A noter que l'on découpe bien distinctement le code Javascript des deux modèles pour la lisibilité dans le contexte pédagogique de l'exemple. Cependant, un code plus dense serait possible en s'inspirant de l'exemple 5.

html
```
<body>
    <div id="ma_div">
        <p id="noeud_texte">Ceci <b>est</b> un <i>noeud texte</i> <span id="mon_span">au sein de la DIV
 - cliquez dessus</span><a id="mon_lien" href="" >, et test du lien</a></p>
    </div>

</body>
```

javascript
```
    var oDiv    = document.getElementById('ma_div');
    var oText    = document.getElementById('noeud_texte');
    var oSpan    = document.getElementById('mon_span');
    var oLink    = document.getElementById('mon_lien');

// Ajout des listeners sur chacun de ces éléments (MSIE) ou seulement sur le nud au sommet du flux (W3C)
// et blocage de la propagation de l'événement vers les éléments enfants
```

javascript
```
// Modèle W3C
if (oDiv.addEventListener) {
    // DIV
    // nota: on pourrait aussi le faire à un stade différent du flux
    oDiv.addEventListener('click',function (e) {
    alert('Capture de la DIV et arrêt de la propagation');
    e.stopPropagation();
    },true);
}

// Modele MS IE
// Nota:  MS IE ne permet pas la capture
else if (oDiv.attachEvent){
    // DIV
    oDiv.attachEvent('onclick',function (e) {
        alert('2- Bouillonnement de la DIV et arrêt de la propagation');
        if (!e) var e = window.event;
        e.cancelBubble = true;
    });

    // TEXTE: l'evenement ne bouillonnera pas car la propagation l'evenement clik a été stoppée
    oText.attachEvent('onclick',function (e) {
        alert('1- Bouillonnement de l element <p>');
        if (!e) var e = window.event;
        e.cancelBubble = true;
    });

    // SPAN: l'evenement ne bouillonnera pas car la propagation l'evenement clik a été stoppée
    oSpan.attachEvent('click',function (e) {
        alert('Bouillonnement de l element <span>');
        if (!e) var e = window.event;
        e.cancelBubble = true;
    });

    // LIEN: l'evenement ne bouillonnera pas car la propagation l'evenement clik a déjà été stoppée
    oLink.attachEvent('onclick',function (e) {
        alert('3- Bouillonnement de l element <a>');
        if (!e) var e = window.event;
        e.cancelBubble = true;
        return false;
    });

}
```

Explications

Dans le cas du modèle W3C, on arrête la propagation immédiatement, car on peut détecter l'événement lors de la capture du nœud sommet du DOM. On peut donc se passer d'attacher les événements sur les descendants de l'élément DIV.

Dans le cas du modèle Microsoft, la détection de l'événement pendant la phase capture n'existe pas. Elle est réalisée pendant la phase de bouillonnement. On est donc contraint d'arrêter la propagation sur le dernier descendant si l'on veut inhiber le lien A.

Pour vous rendre compte de l'effet de propagation, il suffit de supprimer l'arrêt de la propagation de l'événement *click* dans l'arbre DOM, vous verrez alors apparaître successivement les alertes liées à la propagation de l'événement sur les descendants du nœud sommet.

Nota: Opera et Safari suivent tous deux le modèle W3C dans ce cas.

5-E - Des événements plus complexes

5-E-1 - Création manuelle d'objet événement

5-E-1-A - Introduction

On a vu précédemment que le modèle W3C DOM Niveau 2 Event permet d'attacher plusieurs gestionnaires à un même élément (ex: DIV) avec le même événement (ex: *click*). Contrairement au modèle traditionnel qui ne permet d'attacher qu'un seul gestionnaire traité comme une méthode d'un élément donné (ex: *ref_element.onclick*()).

Le DOM fournit cependant quelques méthodes pour créer et paramétrer « manuellement » un objet événement fictif et de l'utiliser comme déclencheur d'un gestionnaire sur la cible souhaitée.

L'élément sur lequel on affecte un objet événement fictif de ce type n'a alors pas besoin d'écouter cet événement car le parent de l'élément peut potentiellement écouter également cet événement.

Il faut noter que le déclenchement de cet événement fictif sur un élément ne gère pas l'action par défaut prévisible avec cet événement. Par exemple, générer un objet événement fictif de type FOCUS sur un élément n'engendrera pas le focus sur cet élément. Pour cela, il faudra utiliser la méthode FOCUS directement sur l'élément.

C'est particulièrement important dans le cas de l'interface des événements utilisateur *UIEvent* puisque cela empêche de simuler des actions du navigateur au travers du script. Sans quoi on ferait face à un sérieux problème de sécurité.

5-E-1-B - Principe et syntaxe du fonctionnement en 4 étapes :

1 Sélection de l'élément cible *element_cible*

Reportez-vous au chapitre L'accès aux éléments DOM

1 Création de l'objet événement fictif parmi l'une des 5 interfaces DOM Event : Events, MouseEvents, HTML, UIEvents, MutationEvents

var oEvt = document.createEvent(interfaceDOMDevent);

1 Initialisation de cet objet fictif *oEvt* selon le type d'interface par l'une de ces méthodes :

Interface	Initialisation
Event	initEvent('type_evt', bouillonnant, annulable)
HTML	initEvent('type_evt', bouillonnant, annulable)
UIEvents	initUIEvent('type_evt', bouillonnant, annulable, windowObject, detail)
MouseEvents	initMouseEvent('type_evt', bouillonnant, annulable, windowObject, detail, screenX, screenY, clientX, clientY, ctrlKey, altKey, shiftKey, metaKey, button, relatedTarget)
MutationEvents	initMutationEvent('type_evt', bouillonnant, annulable, relatedNode, prevValue, newValue, attrName, attrChange)

1 Affectation de l'objet fictif à la cible

element_cible.dispatchEvent(oEvt);

5-E-1-C - Exemple 1 : transformer l'événement « click » affecté par défaut à un lien hypertexte <a> en un événement de type « mouseover »

html
```
<body>
    <div id="ma_div">
        <a id="mon_lien" href="/lib/tuto_dvp/page.html" >Essayez de cliquer sur ce lien et vous verrez
 l'effet du mouseover</a>
    </div>
</body>
```

javascript
```
// La petite fonction utilitaire
function AddEvent(id, evt_type, ma_fonction, phase) {
    var oElt = document.getElementById(id);
    // modèle W3C mode bubbling
    if( oElt.addEventListener ) {
        oElt.addEventListener(evt_type, ma_fonction, phase);
    // modèle MSIE
    } else if( oElt.attachEvent ) {
        oElt.attachEvent('on'+evt_type, ma_fonction);
    }
    return oElt;
}

// Le gestionnaire d'evenement fictif selon les modeles W3C ou MSIE
var CreationEvtFictif = function (e) {

            // Creation de l'evenement fictif
            var oEvt = (document.createEvent)? document.createEvent('MouseEvents') :
 document.createEventObject(); // W3C ou MSIE

            // Initialisation de l'evt fictif avec des parametres fixes
            // W3C
            if (oEvt.initMouseEvent)
                oEvt.initMouseEvent(
                    /* type*/             'mouseup',
                    /* bubble*/            true,
                    /* cancel*/            true,
                    /* AbstractView*/      window,
                    /* detail */         10,
                    /* screenX */         20,
                    /* screenY */         30,
                    /* clientX */         40,
                    /* clientY */         50,
                    /* ctrlKey */         false,
                    /* altKey */         false,
                    /* shiftKey */         true,
                    /* metaKey */         false,
                    /* button */         0,
                    /* relatedTarget*/     null ) ;
            // MSIE
            else {
                    var oEvt = document.createEventObject();
                    oEvt.detail = 10;
                    oEvt.screenX = 20;
                    oEvt.screenY = 30;
                    oEvt.clientX = 40;
                    oEvt.clientY = 50;
                    oEvt.ctrlKey = false;
                    oEvt.altKey = false;
                    oEvt.shiftKey = true;
                    oEvt.metaKey = false;
                    oEvt.button = 0;
                    oEvt.relatedTarget = null;
            }
```

javascript

```
            // Redéfinition de la cible si c'est un noeud texte (cf bug Safari)
            var LaCible = e.target || e.srcElement ; // W3C ou MSIE
            if( LaCible && ( LaCible.nodeType == 3 ) )
                LaCible = LaCible.parentNode;

            // Affectation de l'obj evt fictif à la cible
            // MSIE
            if (document.createEventObject )
                LaCible.fireEvent('onmouseup',oEvt) ;
            // W3C
            else
                LaCible.dispatchEvent(oEvt) ;

        }

// Le gestionnaire qui sera déclenché par l'evenement fictif
var GestionEvtFictif = function (evt_fictif) {
        var s = '',
 propriete=',type,target,currentTarget,bubbles,cancelable,windowObject,detail,screenX,screenY,clientX,clientY,ctr
        for(var param in evt_fictif){
            // Si la propriété existe
            if (propriete.indexOf(',' + param + ',') + 1) {
                s+='\n'+param+': '+ ((typeof(evt_fictif[param])=='object')? 'L\'element A ' :
 evt_fictif[param]);
            }
        }
        alert('Déclenchement de l\'evt fictif:\n' + s);
    }

// Création de l'evenement fictif de type "mouseup" que l'on génère quand on passe sur la cible (mouseover)
    AddEvent('mon_lien', 'mouseover', CreationEvtFictif, false) ;


// On va maintenant pouvoir tester notre obj evt fictif
    AddEvent('mon_lien', 'mouseup', GestionEvtFictif, false) ;
```

Explications

Le code est fonctionnel sur les deux modèles W3C et Microsoft (MSIE), et donc sur les quatre navigateurs.

On créé un événement *mouseover* sur l'élément cible constitué par le lien hypertexte de la balise <a>, identifée par son identifiant *id= « mon_lien »*. Lorsque cet événement est déclenché, en passant sur le lien, le gestionnaire crée à son tour un objet événement fictif « *mouseup* ». Puis on affecte cet objet evenement fictif à la cible. Le gestionnaire *GestionEvtFictif* de cet objet événement fictif affiche les paramètres de l'objet événement fictif.

6 - Conclusion

La gestion des événements DOM par le modèle W3C peut paraître un peu plus compliqué d'accès que l'approche dite « traditionnelle ». Cependant, on se rend rapidement compte des limites du modèle traditionnel dès que l'on a à gérer l'interface d'une page plus complexe. Un bouton peut ainsi prendre différentes fonctions successivement ou contextuellement.

On se rend également rapidement compte de la nécessité de suivre un standard afin d'éviter d'écrire un gestionnaire par type de navigateur. D'où l'avènement du modèle W3C DOM Event Niveau2 et celui imminent du Niveau3.

Le cavalier seul de Microsoft sur le sujet est uniquement une question commerciale. Si la majorité des navigateurs et des développeurs suivent le standard, il s'imposera intégralement par lui-même. En outre, la diversité imposée par le modèle MSIE n'est techniquement pas ingérable même s'il apporte inéluctablement une lourdeur supplémentaire du code. Qui de toutes façons existe par sa différence, indépendamment du niveau des standards.

Pour info, Sources d'inspiration et d'infos **[à supprimer]** :

ECMA

Tutoriaux Javascript et Ajax

FAQ spécifiques au DOM et événements

W3C DOM

W3C DOM Event

QuirksMode

Mozilla

Microsoft