

WIM - M4103C

DOM et api javascript

Denis Monnerat

`monnerat@u-pec.fr` 

29 janvier 2022

IUT de Fontainebleau

Le DOM : introduction

Structure du DOM

Manipulation du DOM

DOM HTML

DOM et css

DOM Events

Le DOM : introduction

Pourquoi ?

Chaque navigateur implantait ses propres méthodes de manipulation du contenu html. D'où la nécessité :

- Uniformiser et abstraire (indépendance de tout langage) la représentation d'un document html (et xml)
- Spécifier une api de manipulation de cette représentation.

Le Dom **Document Object Model**

Dom

- une représentation objet normalisée des documents html et xml, sous forme arborescente.
- une api qui permet d'accéder au document et de manipuler son contenu, sa structure et ses styles.
- permet ainsi d'interfacer un document avec un langage, comme javascript, mais aussi python, php, java, etc....

Plusieurs niveaux de spécification ont vu le jour :

- DOM 1 (1998) : manipulation d'un document html ou xml.
- DOM 2 (2001) : dernière version finalisée : ajout de méthodes de parcours de l'arbre, gestion des événements et des feuilles de styles, vues filtrées.
- DOM 3/4 (2004/2014) : interface de chargement et de sauvegarde de documents xml, événement clavier, XPath...
- DOM Living Standard : n'est plus versionné.

Tous les navigateurs supportent le DOM 2/3.

Quelques références

- <https://developer.mozilla.org/fr/docs/DOM>
- <http://www.w3.org/DOM>

Structure du DOM

Un arbre

Structure arborescente **de noeuds** (node).

- Chaque noeud est un objet, avec des méthodes et des attributs.
- Cette interface est implanté pour plusieurs langages, comme javascript, php, java, python, perl, activeX.
- Les noms des interfaces, classes, méthodes et propriétés sont indépendantes du langages.
- Interface DOM pour d'autres documents (xml) du WEB : MathML, SVG, X3D, etc.

Un exemple

HTML \Rightarrow DOM \Rightarrow Vue



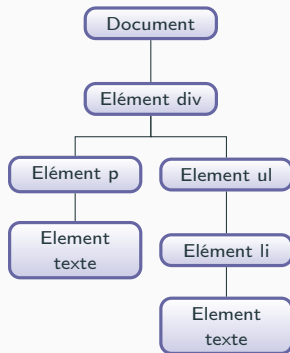
Navigateur = **parseur HTML** + **moteur graphique**

Parseur HTML : construit l'arbre DOM en mémoire

Moteur graphique : construit une représentation de l'arbre DOM, suivant les règles données dans les CSS

Représentation hiérarchique sous forme d'arbre.

```
<div id="exemple">
  <p>un paragraphe</p>
  <ul>
    <li>un element de liste</li>
  </ul>
</div>
```



Un exemple

HTML \Rightarrow DOM \Rightarrow Vue



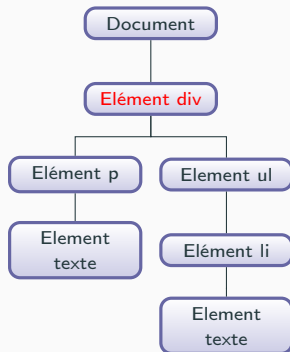
Navigateur = **parseur HTML** + **moteur graphique**

Parseur HTML : construit l'arbre DOM en mémoire

Moteur graphique : construit une représentation de l'arbre DOM, suivant les règles données dans les CSS

Représentation hiérarchique sous forme d'arbre.

```
<div id="exemple">  
  <p>un paragraphe</p>  
  <ul>  
    <li>un element de liste</li>  
  </ul>  
</div>
```



Un exemple

HTML \Rightarrow DOM \Rightarrow Vue



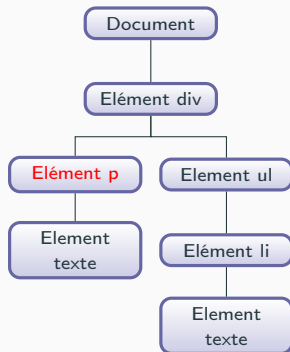
Navigateur = **parseur HTML** + **moteur graphique**

Parseur HTML : construit l'arbre DOM en mémoire

Moteur graphique : construit une représentation de l'arbre DOM, suivant les règles données dans les CSS

Représentation hiérarchique sous forme d'arbre.

```
<div id="exemple">
  <p>un paragraphe</p>
  <ul>
    <li>un element de liste</li>
  </ul>
</div>
```



Un exemple

HTML \Rightarrow DOM \Rightarrow Vue



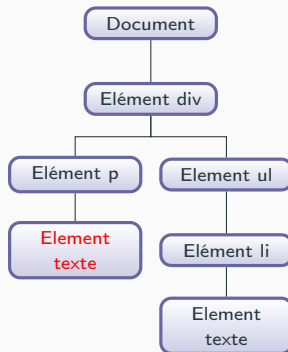
Navigateur = **parseur HTML** + **moteur graphique**

Parseur HTML : construit l'arbre DOM en mémoire

Moteur graphique : construit une représentation de l'arbre DOM, suivant les règles données dans les CSS

Représentation hiérarchique sous forme d'arbre.

```
<div id="exemple">  
  <p>un paragraphe</p>  
  <ul>  
    <li>un element de liste</li>  
  </ul>  
</div>
```



Un exemple

HTML \Rightarrow DOM \Rightarrow Vue



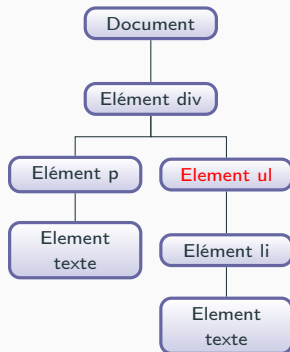
Navigateur = **parseur HTML** + **moteur graphique**

Parseur HTML : construit l'arbre DOM en mémoire

Moteur graphique : construit une représentation de l'arbre DOM, suivant les règles données dans les CSS

Représentation hiérarchique sous forme d'arbre.

```
<div id="exemple">
  <p>un paragraphe</p>
  <ul>
    <li>un element de liste</li>
  </ul>
</div>
```



Un exemple

HTML \Rightarrow DOM \Rightarrow Vue



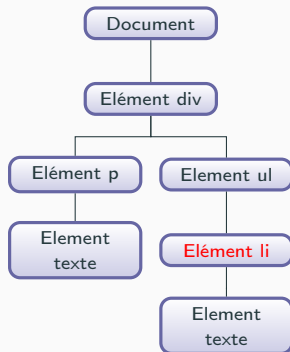
Navigateur = **parseur HTML** + **moteur graphique**

Parseur HTML : construit l'arbre DOM en mémoire

Moteur graphique : construit une représentation de l'arbre DOM, suivant les règles données dans les CSS

Représentation hiérarchique sous forme d'arbre.

```
<div id="exemple">
  <p>un paragraphe</p>
  <ul>
    <li>un element de liste</li>
  </ul>
</div>
```



Un exemple

HTML \Rightarrow DOM \Rightarrow Vue



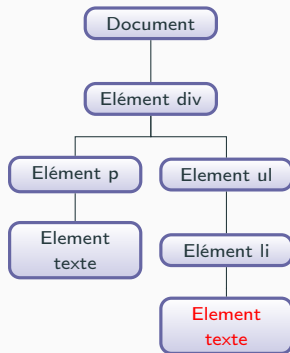
Navigateur = **parseur HTML** + **moteur graphique**

Parseur HTML : construit l'arbre DOM en mémoire


Moteur graphique : construit une représentation de l'arbre DOM, suivant les règles données dans les CSS

Représentation hiérarchique sous forme d'arbre.

```
<div id="exemple">
  <p>un paragraphe</p>
  <ul>
    <li>un element de liste</li>
  </ul>
</div>
```



DOM Living Standard

<https://dom.spec.whatwg.org/> 

~> Pour chaque classe, il existe des méthodes (noms explicites en camelCase) et propriétés pour accéder aux données et les modifier.

~> DOM HTML spécialise le DOM CORE.

Il y a une interface `Node`, dont dérive les noeuds de type :

- `Element` : représente un élément HTML.
- `Text` : représente du texte (forcément une feuille de l'arbre).

Un noeud de type `Element` peut avoir des fils éléments et/ou textes.

Principaux objets

Voir la référence sur MDN

https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model 

- **Node** : les noeuds, qui peuvent être de différents types :
 - **Document** : le document (racine) duquel on a construit le DOM
 - **Element** : nœuds éléments HTML (ou XML), contiennent d'autres nœuds (de type Element, Comment...)
 - **CharacterData** : noeuds de texte, contiennent du texte (objet Text)
- **Event** : les événements
- etc.

Deux objets représentent des collections :

- **NodeList** : liste de noeuds (par exemple la liste des fils d'un noeud)
- **HTMLCollection** : liste d'éléments

Une collection peut être statique ou "vivante" (live), c'est-à-dire que les changements du DOM y sont reflétés en permanence.

Manipulation du DOM

Manipulation du DOM

Les noeuds

Accéder à un noeud

Directement

↪ avec un nom d'élément :

```
NodeList getElementsByTagName(in DOMString tagname);
```

avec l'interface correspondante à une liste de noeud :

```
interface NodeList {  
    Node item(in unsigned long index);  
    readonly attribute unsigned long length;  
};
```

↪ avec un identifiant :

```
Element getElementById(in DOMString elementId);
```

Accéder à un noeud

Avec un selecteur css (<http://www.w3.org/TR/selectors-api/> )

- Selector Api du DOM : reprend le mécanisme de selections de jquery qui utilise des selecteurs css.
- Un selecteur css : une règle css (qui selectionne donc un ensemble de noeuds).

```
let el = document.querySelector(".myclass");  
// renvoie le premier noeud correspondant  
let special = document.querySelectorAll( "p.warning, p.note" );  
// renvoie tous les noeuds sous forme d'une NodeList  
let cells = document.querySelectorAll("#score>tr>td:nth-of-type(2)");  
// si aucune correspondance, renvoie null  
// Les pseudo-classes css ne sont pas supportées
```

Accéder à un noeud



en parcourant l'arbre avec les propriétés de la classe
node

<code>readonly attribute Node</code>	<code>parentNode;</code>
<code>readonly attribute NodeList</code>	<code>childNodes;</code>
<code>readonly attribute Node</code>	<code>firstChild;</code>
<code>readonly attribute Node</code>	<code>lastChild;</code>
<code>readonly attribute Node</code>	<code>previousSibling;</code>
<code>readonly attribute Node</code>	<code>nextSibling;</code>
<code>readonly attribute NamedNodeMap</code>	<code>attributes;</code>



Firefox stocke l'ensemble des espaces du document html sous forme de noeuds texte. il faut les prendre en compte lors des traitements.

Créer un noeud

Dans la classe Document

↪ un noeud élément :

```
Element                createElement(in DOMString tagName)  
    raises(DOMException);
```

↪ un noeud texte :

```
Text                    createTextNode(in DOMString data);
```

↪ un noeud attribut :

```
Attr                    createAttribute(in DOMString name)  
    raises(DOMException);
```

On peut également dupliquer un noeud avec la méthode `cloneNode` de la classe `Node` :

```
let elem=document.getElementById("mon_div");  
let sous_arbre=elem.cloneNode(true);  
let div=elem.cloneNode(false);
```

Méthode de la classe node :

```
Node insertBefore(  
    in Node newChild,  
    in Node refChild)  
raises(DOMException);
```

```
Node replaceChild(  
    in Node newChild,  
    in Node oldChild)  
raises(DOMException);
```

```
Node removeChild(in Node oldChild)  
    raises(DOMException);
```

```
Node appendChild(in Node newChild)  
    raises(DOMException);
```

1. Arbre initial

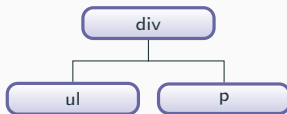
```
<div id="mon_div">
  <ul><li>toto</li></ul>
</div>
```

2. Exécution du code

```
let zone=document.getElementById("mon_div");
let p=document.createElement("p");
let texte=document.createTextNode("blablabla ...");
p.appendChild(texte);
zone.appendChild(p);
```

3. Nouvel arbre

```
<div id="mon_div">
  <ul><li>toto</li></ul>
  <p>blablablabla ...</p>
</div>
```



1. Arbre initial

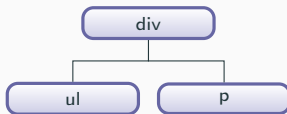
```
<div id="mon_div">
  <ul><li>toto</li></ul>
</div>
```

2. Exécution du code

```
let zone=document.getElementById("mon_div");
let p=document.createElement("p");
let texte=document.createTextNode("blablabla ...");
p.appendChild(texte);
zone.appendChild(p);
```

3. Nouvel arbre

```
<div id="mon_div">
  <ul><li>toto</li></ul>
  <p>blablablabla ...</p>
</div>
```



1. Arbre initial

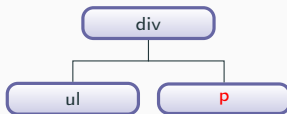
```
<div id="mon_div">
  <ul><li>toto</li></ul>
</div>
```

2. Exécution du code

```
let zone=document.getElementById("mon_div");
let p=document.createElement("p");
let texte=document.createTextNode("blablabla ...");
p.appendChild(texte);
zone.appendChild(p);
```

3. Nouvel arbre

```
<div id="mon_div">
  <ul><li>toto</li></ul>
  <p>blablablabla ...</p>
</div>
```



Manipulation du DOM

Les attributs

Attributs

Tout élément (node de type `NODE_ELEMENT`) peut contenir des attributs, c'est à dire une paire (clé,valeur) rajoutant des informations.

```
interface Element : Node {
    readonly attribute DOMString      tagName;
    DOMString      getAttribute(in DOMString name);
    void           setAttribute(in DOMString name,
                               in DOMString value)
        raises(DOMException);
    void           removeAttribute(in DOMString name)
        raises(DOMException);
    Attr           getAttributeNode(in DOMString name);
    Attr           setAttributeNode(in Attr newAttr)
        raises(DOMException);
    Attr           removeAttributeNode(in Attr oldAttr)
        raises(DOMException);
    boolean        hasAttribute(in DOMString name);
```

La plupart des attributs HTML sont accessibles via des propriétés de l'objet JS représentant un élément :

- `id` : identifiant d'un élément
- `href` : attribut href, pour un lien
- `src` : attribut src, pour une image
- `style` : objet représentant le contenu de l'attribut style (voir plus loin)
- `classList` : objet représentant le contenu de l'attribut class (voir plus loin)
- etc.

Attention, la valeur de ces propriétés n'est pas forcément exactement identique à celle de l'attribut (par exemple, la propriété `src` d'un `HTMLImageElement` contient l'URL absolue vers l'image, telle que résolue par le navigateur)

On peut manipuler des sous-arbres avec l'objet `DocumentFragment`.

On utilise un fragment comme un noeud.

```
let zone=document.getElementById("ma_zone");
let fragment=document.createDocumentFragment();
let jours=["lundi","mardi","mercredi","jeudi","vendredi",
    "samedi","dimanche"];
for(let j of jours) {
    let p=document.createElement("p");
    let texte=document.createTextNode(j);
    p.appendChild(texte);
    fragment.appendChild(p);
}
zone.appendChild(fragment);
```

Quel est le père des paragraphes créés ?

Très utile lorsque l'on aurait été obligé de modifier le dom après chaque insertion de noeud, comme c'est le cas dans l'exemple précédent.
Permet de gagner en performance (pas de **reflow**).
Il est important de savoir comment une librairie qui abstrait le dom gère sa modification.

Manipulation du DOM

Dom et XML

Javascript permet :

- de créer des documents xml, vierges ou à partir d'un fichier xml local.
- de le parser avec l'api DOM.
- de le transformer avec xslt.

DOM HTML

Le DOM HTML est une extension du DOM CORE.

Pourquoi ?

- Spécialiser et ajouter des attributs et fonctionnalités spécifiques aux documents et éléments HTML.
- Assurer la compatibilité avec le DOM 0.
- Ajouter des mécanismes, des traitements utiles, communs et pratiques dans le cas spécifique d'HTML.

- `HTMLDocument` dérive de `Document` du DOM CORE.
- `HTMLElement` dérive de `Element` du DOM CORE.
- Nouvelle Interface pour chaque type d'éléments html qui spécialise `HTMLElement`.

HTMLDocument

```
interface HTMLDocument : Document {
    attribute DOMString      title;
    readonly attribute DOMString  referrer;
    readonly attribute DOMString  domain;
    readonly attribute DOMString  URL;
    attribute HTMLElement      body;
    readonly attribute HTMLCollection  images;
    readonly attribute HTMLCollection  applets;
    readonly attribute HTMLCollection  links;
    readonly attribute HTMLCollection  forms;
    readonly attribute HTMLCollection  anchors;
    attribute DOMString      cookie;
    // raises(DOMException) on setting

    void      open();
    void      close();
    void      write(in DOMString text);
    void      writeln(in DOMString text);
    NodeList  getElementsByName(in DOMString elementName);
};
```

```
interface HTMLElement : Element {  
    // DOM tree accessors  
    NodeList getElementsByClassName(in DOMString classNames);  
  
    // dynamic markup insertion  
    attribute DOMString innerHTML;  
  
    // metadata attributes  
    attribute DOMString id;  
    attribute DOMString title;  
    attribute DOMString lang;  
    attribute DOMString dir;  
    attribute DOMString className;
```

La plupart des éléments html spécialisent cette classe. Voici deux exemples

HTMLFormElement

```
interface HTMLFormElement : HTMLElement {
    readonly attribute HTMLCollection elements;
    readonly attribute long length;
    attribute DOMString name;
    attribute DOMString acceptCharset;
    attribute DOMString action;
    attribute DOMString enctype;
    attribute DOMString method;
    attribute DOMString target;
    void submit();
    void reset();
};
```

HTMLTableElement

```
interface HTMLTableElement : HTMLElement {
    // Modified in DOM Level 2:
    attribute HTMLTableCaptionElement caption;
    // raises(DOMException) on setting

    // Modified in DOM Level 2:
    attribute HTMLTableSectionElement tHead;
    // raises(DOMException) on setting

    // Modified in DOM Level 2:
    attribute HTMLTableSectionElement tFoot;
    // raises(DOMException) on setting

    readonly attribute HTMLCollection rows;
    readonly attribute HTMLCollection tBodies;
    attribute DOMString align;
    attribute DOMString bgColor;
    attribute DOMString border;
    attribute DOMString cellPadding;
    attribute DOMString cellSpacing;
    attribute DOMString frame;
    attribute DOMString rules;
    attribute DOMString summary;
    attribute DOMString width;
};
```

HTMLTableElement (suite)

```
interface HTMLTableElement : HTMLElement {

    HTMLElement        createTHead();
    void               deleteTHead();
    HTMLElement        createTFoot();
    void               deleteTFoot();
    HTMLElement        createCaption();
    void               deleteCaption();
    // Modified in DOM Level 2:
    HTMLElement        insertRow(in long index)
        raises(DOMException);
    // Modified in DOM Level 2:
    void               deleteRow(in long index)
        raises(DOMException);
};
```

Spécifications complètes

<http://www.w3.org/TR/DOM-Level-2-HTML/> 

La propriété innerHTML permet de récupérer ou fixer le contenu html d'un élément, à partir d'une chaîne de caractères (représentant un fragment d'html)

`document`

```
.getElementById("ma_div")  
.innerHTML="<p>youpi !!!</p>";
```

On a aussi `innerText` et `textContent`.

DOM et css

Attribut style d'un noeud.

```
let toto = document.getElementById('toto');  
toto.style.color="green";  
toto.style.backgroundColor="blue";  
toto.style.display="none";
```

Attribut ClassList d'un noeud.

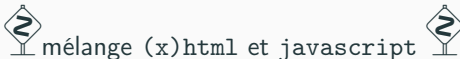
```
div.classList.toggle("visible", i < 10 );  
// ajouter ou supprimer plusieurs classes  
div.classList.add("foo", "bar", "baz");  
div.classList.remove("foo", "bar", "baz");  
const cls = ["foo", "bar"];  
div.classList.add(...cls);  
div.classList.remove(...cls);  
// remplacer la classe "foo" par la classe "bar"  
div.classList.replace("foo", "bar");
```

DOM Events

But ?

Des événements peuvent être associé à des balises html en utilisant certains de leur attribut dont le nom est préfixé par `on` (`onload`, `onclick`, etc...)

- dans l'html directement.
- directement dans la noeud.



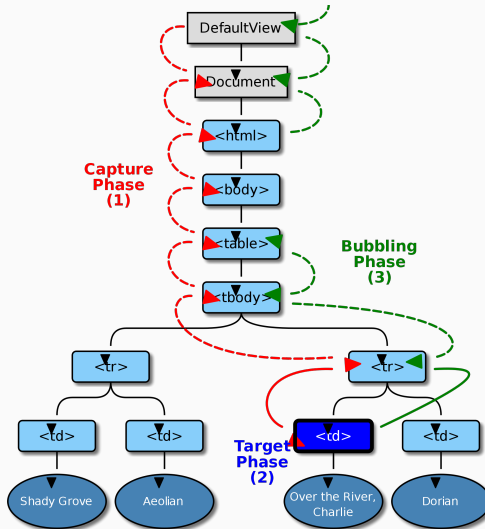
DOM fournit un support à la gestion des événements.

- Modèle générique qui permet :
 - l'enregistrement de handlers d'événements.
 - de décrire le mécanisme de propagation des événements dans une structure d'arbre.
 - de décrire une information contextuel pour chaque événement.
- Fournir un sous-ensemble commun pour ce qui est déjà utilisé dans DOM 0. (augmenter la portabilité)
- Consulter les pages

<https://www.w3.org/TR/uievents/> 

DOM Events

Propagation



- Basiquement, chaque événement possède une cible (EventTarget) à qui il est délivré.
- **Tous** les handlers correspondant sont alors exécutés (ordre?).

Il existe deux modes de propagations :

Modes

Ascendant	nommé bubbling, bas niveau vers haut niveau.(I.E au début)
Descendant	nommé capturing, haut niveau vers bas niveau. (Netscape au début)

- capturing

Lors de la délivrance d'un événement à sa cible, l'événement peut être délivré à un noeud parent (en partant de la racine) si le noeud a enregistré une fonction reflexe en autorisant la capture avec le paramètre `useCapture` dans la fonction d'enregistrement (`addEventListener`).

La propagation peut-être interrompu avec la méthode `stopPropagation` de la classe `Event`.

- bubbling

Certains événements, après la phase de descente, sont alors délivrés aux parents de la cible (qui ont enregistré une fonction reflexe), en remontant vers la racine. (ceux qui avaient capturé en sont exclus!)

La propagation peut être interrompue avec la méthode `stopPropagation` de la classe `Event`.

- `Cancellation` : L'implémentation Dom (l'application) possède un traitement par défaut pour certains événements (click et hyperlien par exemple)
Quand l'événement survient, les fonctions reflexes sont exécutées. Le traitement par défaut peut être alors inhibé par la méthode `preventDefault` de la classe `Event`.

DOM Events

Evts

```
function ReponseClick(e){  
    /* traitement*/  
    // e.target : siège de l'événement  
    // e.currentTarget : propagation ou capture  
}  
  
let n=document.getElementById("mon_noeud");  
n.addEventListener("click",ReponseClick,false);  
  
n.removeEventListener("click",ReponseClick,true);
```

Types d'évts

Il existe plusieurs catégories logiques d'événements :

Événement Interface Utilisateur	Event UIEvent	Evenement générique DOMActivate, DOMFocusIn, DOMFocusOut et événements claviers
Evt souris	MouseEvent	click, mousedown, mouseup, mou- seover, mouseout
Evt clavier	WheelEvent	
Evt élément html	KeyboardEvent	input, keydown, keyup, etc.
	HTMLEvent	abort, blur, change, focus, error, load, unload, reset, scroll, select, submit, ...
Evt de mutation	MutationEvent	modification du DOM

Nom et description

<code>click</code>	clic de souris dans l'élément
<code>mousedown</code> , <code>mouseup</code>	bouton enfoncé, relaché
<code>mouseover</code> , <code>mouseout</code>	le pointeur entre ou sort de l'élément
<code>mousemove</code>	déplacement du pointeur

- Tous ces événements sont transmis aux ascendants (bubbling).
- Seul `mousemove` n'a pas de traitement par défaut (cancelable).
- Il existe également une interface `WheelEvent` qui dérive de `MouseEvent`. Le nom de l'événement est `wheel`

- Il n'existe pas dans DOM 2 !
- Introduction dans DOM 3 (depuis 2009)
- L'interface correspondante est `KeyboardEvent`

Nom et description

input

keydown

keypress

keyup

touches qui produisent un caractère

Événements relatifs à l'interface graphique.

Nom et description

focus, blur	un élément gagne ou perd le focus
load	chargement complet d'une page
resize	fenêtre du navigateur redimensionnée
scroll	page scrollée
unload	page déchargée
abort	chargement d'une page stoppée avant le chargement complet d'une image
error	erreur de chargement ou dans un script

Evénements relatifs aux éléments de formulaire

Nom et description	
change	changement dans un élément de formulaire
reset	réinitialisation du formulaire
select	sélection de texte
submit	soumission d'un formulaire