

Le Document Object Model : DOM

Formation

Développeur Web et Web Mobile

Module : 03

Développer la partie front-end d'une application web

Séquence : 03

Développer une interface utilisateur web dynamique

Séance : 03

Développer des scripts clients dans une page web

| | |
|-----------------------|------------------|
| Libellé réduit : | Le DOM |
| Type de document : | Support de cours |
| Version : | 2 |
| Auteur : | Didier Bonneau |
| Centre afpa : | Créteil |
| Date de création : | 01/01/2014 |
| Date de mise à jour : | 19/10/2021 |

Sommaire

Contenu

| | | |
|------|--|---|
| I. | Introduction | 2 |
| A. | le DOM et ses niveaux | 2 |
| B. | Importance du DOM..... | 3 |
| II. | La structure du DOM | 3 |
| III. | Les principales propriétés et méthodes | 4 |
| A. | Connaitre les types de noeuds..... | 4 |
| B. | Accéder directement aux nœuds | 5 |
| C. | Connaitre les liens de parenté | 6 |
| D. | Agir sur le DOM | 7 |
| IV. | Annexes | 8 |
| A. | Précisions sur le DOM | 8 |
| B. | Détails des propriétés et méthodes..... | 9 |

I. INTRODUCTION

A. LE DOM ET SES NIVEAUX

Le **DOM**, *Document Object Model*, Modèle Objet du Document, permet une modélisation cohérente et ordonnée d'un document XML, donc en particulier d'une page HTML.

Il fournit :

- une représentation structurée du document, sous forme d'arborescence d'objets.
- un ensemble de fonctions pour accéder, parcourir ou modifier cette arborescence.

Le DOM niveau 0 correspond à l'ancien modèle de gestion interne d'un document WEB. Il permet d'agir par Javascript seulement sur certains éléments (par exemple les manipulations d'éléments de formulaire, la couleur de fond du document (`document.bgcolor`), l'url d'un lien (`document.links[0].href ..`), mais on ne peut pas agir sur la structure du document.

Ce qui est possible (de façon limitée) avec l'ancien modèle DOM niveau 0 :

- le nommage des éléments (propriété `name`)
- le classement de certains types de balises dans des tableaux : `forms[]`, `images[]`, `links[]` ..
- les gestionnaires d'événements associés à certaines balises (par exemple `onmouseover`, `onclick` sur une balise `link`, `onclick` sur un bouton, etc...
- on peut aussi utiliser directement les valeurs de la propriété « `name` » pour atteindre les éléments sans passer par les tableaux (en prenant garde de respecter l'unicité de nom)

Exemple de syntaxe : affecter la valeur du champ nommé « `nom` » dans le formulaire nommé « `inscription` » :

```
<form name="inscription">
  <input type="text" name="nom" />
</form>

document.forms["inscription"].nom.value='Dupond';
document.forms.inscription.nom.value='Dupond';
document.forms[0].nom.value='Dupond';
document.inscription.nom.value='Dupond';
```

Les DOM "intermédiaires" datent de la période de la "guerre" de Microsoft contre Netscape. Ils ont répondu différemment à la demande d'intégrer la gestion des feuilles de style dans les pages html, ce qui a donné naissance au "DHTML", avec 2 DOM inconciliables `document.layers` (Netscape 4) et `document.all` (Microsoft 4).

Le premier standard décrit par le W3C, le DOM niveau 1, date de 1998 et a été rapidement suivi de la version 2 actuelle à partir de 2000. Il a été implémenté dans les navigateurs Mozilla et IE. Ce standard devrait normalement être pris en charge par tous les navigateurs.

Pour tester la prise en charge du DOM 1 par le navigateur, il suffit de tester l'existence de certaine méthode dans l'objet « `document` » :

```
<script type="text/javascript">
  if (document.getElementById && document.createElement)
    alert("le navigateur gère le DOM 1") ;
</script>
```

B. IMPORTANCE DU DOM

Grâce à cette représentation DOM et les fonctions associées, la page WEB est complètement accessible, et donc modifiable dynamiquement des points de vue :

- **structure** : par exemple, permuter 2 paragraphes, ajouter un titre, etc...
- **style** : changement de couleur, de propriétés d'un tableau, de visibilité d'un paragraphe.
- **contenu** : mise à jour du texte d'un paragraphe, d'une image, nouveau paragraphe.

Le DOM version 2 actuelle est un standard du consortium World Wide Web (<http://www.w3.org/TR/DOM-Level-2-Core/core.html>). Cela devrait garantir que tous les navigateurs récents implémentent ses fonctionnalités. Les versions plus récentes des principaux navigateurs donnent accès au DOM en JavaScript (encore imparfaitement) et bientôt le DOM devrait devenir transparent au programmeur de scripts.

Au point de vue informatique, le DOM est une représentation arborescente entièrement orientée objet de la page Web. Ce n'est pas un langage de programmation, mais une structure arborescente accompagnée de spécifications (propriétés et fonctions ou méthodes), sortes de directives (rassemblées sous forme d'interfaces abstraites).

Ces interfaces peuvent être implémentées dans tous les langages de programmation, et pas seulement en JavaScript. Bien sûr ce langage reste privilégié car il est nativement interprété par tous les navigateurs.

II. LA STRUCTURE DU DOM

- Le DOM d'un document (XML ou XHTML) est d'abord une structure de données qui représente ce document comme une arborescence de nœuds (tel un arbre généalogique, ou semblable à une arborescence de répertoires et de fichiers). La racine symbolique de cet arbre est un nœud d'un type tout-à-fait spécial, nommé « **document** », qui correspond à peu près à la balise <html>.
- Chaque nœud du document (sauf document) n'a qu'un seul "parent" et se trouve être un descendant (« enfant » direct ou non) de ce nœud « ancêtre » et peut avoir lui-même plusieurs nœuds « enfants ».
- De façon générale, les balises HTML de la page WEB correspondent à des nœuds de l'arbre DOM et en constituent la structure. Les différentes parties du contenu, c'est-à-dire le texte de la page, sont inclus dans l'arbre en tant que nœuds particuliers. Chaque nœud textuel est toujours un « enfant » du nœud représentant la balise à laquelle il est accroché dans le code HTML.
- Par contre les attributs des balises (par exemple : src pour , href pour <a>, width et border pour <table>, etc ..) ne sont pas représentés dans l'arbre comme des nœuds enfants mais par des propriétés des nœuds des balises.
- Voir plus loin des précisions sur les différents types de nœuds.
- Le DOM est aussi une structure de programmation orientée-objet, qui se compose d'un ensemble d'interfaces (et non de classes, comme c'est le cas pour un véritable

langage objet). Ceci a comme conséquence pour le programmeur de ne pas utiliser de constructeurs d'objets comme `var t = new Text("nouveau nœud texte");`. A la place, il invoque une méthode de création définie dans l'une des interfaces, par exemple pour créer un nœud de type texte:

```
var t = document.createTextNode("nouveau nœud texte");
```

- L'interface fondamentale est « **Node** ». Les autres en sont des sous-interfaces, correspondant à des types de nœuds particuliers (essentiellement **Document**, **Element**, **Text**). De plus, chaque objet Node a un type particulier déterminé par une valeur de la propriété **nodeValue** (voir plus loin).
- Le DOM définit une série de méthodes et de propriétés qui permettent d'interroger l'état de l'arbre, le parcourir et le modifier. A partir de chaque nœud pris comme base de départ, on peut explorer complètement l'arborescence du document, et le modifier dynamiquement, en interaction avec des déclenchements d'événements.

III. LES PRINCIPALES PROPRIETES ET METHODES

Voici les principales propriétés et méthodes qui permettent d'accéder et de parcourir le DOM

A. CONNAITRE LES TYPES DE NOEUDS

Chaque nœud a quelques propriétés et méthodes qui donnent des informations à son sujet.

- **n.nodeType** indique le type du nœud « n »

| type de nœud/interface | valeur de <i>nodeType</i> |
|------------------------|---------------------------|
| element | 1 |
| attribut | 2 |
| text | 3 |
| comment | 8 |
| document | 9 |

- **n.nodeName** donne le nom du nœud n. S'il s'agit d'un élément HTML, c'est le nom de la balise (**mis en majuscules**) ou de l'attribut.
Pour un nœud de type texte, le nom générique est toujours « **#text** ».
Le nœud racine du document s'appelle **#document**.
- **n.nodeValue** donne ou force la valeur ou le contenu d'un nœud. Pour les nœuds de type text (nodeType=3) c'est le texte, pour les nœuds « attribute » (nodeType=2), c'est la valeur affectée à l'attribut, sinon pour les nœuds « element » (balises) cette propriété a la valeur « null ».
- **n.data** Cette propriété n'a de sens que pour les nœuds de type « text » et donne ou affecte leur contenu actuel.
- **n.getAttribute(attr)** Cette méthode retourne la valeur un attribut dont le nom est passé en paramètre « attr ».

- **n.attributes** Cette propriété renvoie une collection de nœuds attributs.
- **n.setAttribute(attr,val)** Cette méthode affecte la valeur du paramètre « val » à l'attribut dont le nom est passé en paramètre « attr ».
- **n.style** Cette propriété permet de modifier les différents styles du nœud n.

B. ACCEDER DIRECTEMENT AUX NŒUDS

Deux méthodes permettent de rechercher un nœud unique et d'y accéder directement sans parcourir l'arbre. Pour cela, il faut fournir une caractéristique de l'élément recherché : son identifiant (attribut id) ou un sélecteur css.

```
// id est l'identifiant -unique- porté par une balise.
noeud = document.getElementById(id) ;
// sélecteur css (retourne le premier nœud matchant)
noeud = document.querySelector(sélecteur css) ;
```

- document.getElementById(id)

Interroge l'objet document et retourner la référence du nœud unique qui possède cette valeur d'identifiant (et false s'il n'est pas trouvé). Exemple d'utilisation pour placer une phrase à l'emplacement d'un nœud <p> identifié par l'id "ici" :

```
remplacer le contenu d'un paragraphe
// "ici" est l'identifiant d'un paragraphe
// <p id="ici">Que vais-je dire ?</p>
n= document.getElementById("ici");
texte = "Bonjour à tous !";
n.firstChild.data = texte;
```

- document.querySelector(sélecteur css)

```
// pour le même exemple
n= document.querySelector("#ici");
```

Quatre autres fonctions permettent de retourner un ensemble de nœuds appelé « nodeList » qui se base sur une caractéristique non unique des balises

```
// type_balise est un nom de balise (par exemple h3, li, p, div ..)
liste_noeuds = document.getElementsByTagName(type_balise);
// nom de la propriété name des balises (méthode devenue obsolète)
liste_noeuds = document.getElementsByName(propriété name)
// nom de classe css des balises
liste_noeuds = document.getElementsByClassName(nom de classe css)
// sélecteur css (retourne tous les nœuds matchant)
noeud = document.querySelectorAll(sélecteur css) ;
```

- node.getElementsByTagName(balise)

Cette méthode parcourt la sous-arborescence du DOM présente sous « node », à la recherche de toutes les balises du type spécifié « balise ». Si aucune balise n'est rencontrée il renvoie false, sinon il donne un objet de type « **nodeList** », qui est un tableau JS de références vers les divers nœuds trouvés.

Les éléments d'un objet « nodeList » sont accessibles par un index de deux manières différentes : **liste.item(n)** ou **liste[n]**, où n est une valeur d'indice. La première syntaxe item() est une méthode de l'objet nodeList, tandis que liste[n] utilise la syntaxe habituelle d'un tableau pour accéder à un élément de la liste. Il y a 2 propriétés spéciales de document qui donnent directement les références dans le DOM de la balise html (donc la "racine") et de la balise body :

document.documentElement et document.body

Exemple de syntaxe

```
// référence vers la balise body, identique à document.body
body=document.getElementsByTagName("body")[0];
// chercher tous les noeuds des balises <h3> du document
listeH3 = document.getElementsByTagName("h3");
// nombre de h3 dans le document
nombreH3 = listeH3.length;
// parcourir cette liste
for (var i=0; i< listeH3.length; i++)
    alert (i + listeH3[i].nodeName) ;
```

C. CONNAITRE LES LIENS DE PARENTE

Différentes propriétés et méthodes s'appliquent à un nœud et permettent de connaître ses parents proches.

On peut ainsi connaître les nœuds parents et naviguer dans le document tout en suivant ces liens de parenté. Voici les propriétés s'appliquant à chaque nœud :

- **node.parentNode** : donne le nœud parent
- **node.firstChild** : donne le premier enfant
- **node.lastChild** : donne le dernier enfant
- **node.nextSibling** : donne le frère suivant
- **node.previousSibling** : donne le frère précédent
- **node.childNodes** : donne un objet de type nodeList, liste des nœuds enfants de node
- **hasChildNodes()** : méthode testant l'existence d'un nœud

Exemple de code pour connaître ses "enfants"

```
// version simple
chaîne = "";
b = document.body; // b est le nœud body
var enfants = b.childNodes; // enfants est un NodeList
```

```

    for (var i = 0; i < enfants.length; i++)
        chaine += enfants[i].nodeName + "--" ;
    alert(chaine);
// version améliorée
chaine = "";
b = document.body;
for (var m = b.firstChild; m != null; m = b.nextSibling) {
    chaine += m.nodeName ;
    if (m.nodeType == 3)
        chaine += "(" + m.nodeValue + ") --- ";
    else
        chaine += " -- ";
}
alert(chaine);

```

D. AGIR SUR LE DOM

Pour modifier un document il faut agir sur son arbre. Les opérations usuelles sont : créer un nœud balise ou texte, l'ajouter à l'arbre, le retirer ou modifier son contenu textuel. Tout ceci est réalisable en utilisant différentes méthodes :

- `document.createElement(type_element)`

Permet de créer un nœud du type spécifié par « type_element » qui doit correspondre à un nom de balise. Ce nœud sera ensuite ajouté comme fils d'un autre nœud (voir `appendChild()`).

```

// pour créer un nouveau nœud paragraphe
nodePara = document.createElement("p");

```

- `document.createTextNode(text)`

Ceci créer un nœud de type texte.

```

// pour créer un nouveau nœud texte
textDePara = document.createTextNode("Introduction au DOM 2");

```

- `parentNode.appendChild(node)`

Cette méthode permet d'insérer un nœud dans un document XHTML mais pas n'importe où. Elle nécessite que vous indiquiez le nœud qui servira de parentNode au nœud que vous voulez insérer. L'insertion se fait de manière à ce que votre nœud devienne le lastChild de son nouveau parentNode.

```

// ajouter le texte dans le nœud nodePara
nodePara.appendChild(textDePara);
// ajouter le nouveau nœud "nodePara" dans body
body = document.getElementsByTagName("body")[0];
//plus simple : body = document.body
body.appendChild(nodePara);

```

- `textNode.appendData(text)`

Ajoute du texte à un nœud texte

- `childNodes.insertBefore(node)`

Pour insérer dans un arbre, un nœud comme "ainée" d'un nœud à préciser

- `parentNode.removeChild(node)`

On peut enlever un nœud du document en utilisant la méthode « `removeChild` », qui s'adresse au parent du nœud que l'on veut enlever et à laquelle on passe en paramètre le nœud à enlever.

```
// enlève le premier nœud-enfant du nœud liste  
liste.removeChild(liste.firstChild);
```

- `parentNode.replaceChild(newChildNode, oldChildNode)`

```
// remplace le premier enfant du noeud liste par le noeud nv  
liste.replaceChild(nv, liste.firstChild);
```

On peut aussi récupérer ou modifier les attributs d'un nœud par les méthodes :

- `node.getAttribute(nom_attr)`
- `node.setAttribute(nom_attr, valeur)`

Ou encore aux différentes propriétés de style CSS d'une balise par :

- `node.style`

IV. ANNEXES


A. PRECISIONS SUR LE DOM

Le DOM est la description structurée d'un document XML donc en particulier (X)HTML, sous forme d'une arborescence d'objets. Ces objets sont rangés dans des interfaces qui précisent les propriétés et les fonctions de ces objets.

JS utilise des interfaces (et non des classes), comme c'est le cas pour un véritable langage objet. Ceci a comme conséquence pour le programmeur de ne pouvoir utiliser de constructeurs. A la place il invoque une méthode de création définie dans l'une des interfaces. Par exemple pour créer un nœud de type texte :

```
var t = document.createTextNode("nouveau nœud texte");  
// au lieu d'une syntaxe du genre :  
var t = new Text("nouveau nœud texte");
```

L'interface fondamentale est `Node`. Les autres en sont des sous-interfaces, correspondant à des types de nœuds particuliers.

| | | | | | |
|---|----------------|------------|-----------|------------------|---------------------------|
|  | Auteur | Nom région | Formation | Date Mise à jour | Page 8/ 14 |
| | Didier Bonneau | GRN164 | DWWM | 19/10/2021 | Support_De_Cours-DOM.docx |

Chaque objet Node a son type déterminé par une valeur de la propriété `nodeType`.

Voici les principaux types dérivés de Node.

- Document : sous-interface qui gère le nœud document (`nodeType=9`)
Element : sous-interface qui gère les nœuds qui représentent les éléments html (`nodeType=1`)
Text : sous-interface des nœuds texte (`nodeType=3`)
NodeList : tableaux indicés d'éléments (objets créés notamment par la méthode `document.getElementsByTagName()`)

B. DETAILS DES PROPRIETES ET METHODES

| propriété/méthode du DOM | objet retourné | commentaires/exemples |
|--------------------------------|---|---|
| n.nodeName | Le nom du nœud | // exemples avec divers nœuds document.body.nodeName; document.body.firstChild.nodeName; document.body.attributes[0].nodeName; |
| n.nodeType | donne une valeur entière identifiant le type de nœud | 1 pour une balise, 2 pour un attribut, 3 pour un texte... |
| n.nodeValue | La valeur du nœud | |
| n.data | | b = document.body; // affiche ou change le texte directement mis au début de BODY, alert(b.firstChild.data); b.firstChild.data = "Bonjour !"; |
| enfants = n.childNodes; | enfants est la liste indicée de tous les nœuds dont n est le parent | Tester cet exemple n= document.getElementById("p0"); enfants = n.childNodes; premier_enf=n.childNodes[0]; for (var i=0; i< enfants.length; i++) chaine += enfants[i].nodeName ; |
| b=n.hasChildNodes() | recherche l'existence d'enfants du nœud n et donne en conséquence à b la valeur true ou false | // connaître une partie du texte inclus dans tous les paragraphes listeP = document.getElementsByTagName("p"); for (var i=0; i< listeP.length; i++) { if (listeP[i].hasChildNodes()) { document.write(listeP[i]+ " a les enfants"); enfants=listeP[i].childNodes; |

| | | |
|--|--|--|
| | | <pre> for (var i=0; i < enfants.length; i++) document.write("noeud "+ i +" --> " + enfants[i].data) ; } } </pre> |
| parent=n.parentNode | nœud parent de n | |
| enf=n.firstChild; | le nœud du premier enfant de n (ou null si inexistant) | // connaître le texte du premier enfant du paragraphe d'id "p0" n= document.getElementById("p0"); texte= n.firstChild.data; |
| enf=n.lastChild; | le nœud du dernier enfant de n (ou null si inexistant) | // connaître le texte du dernier paragraphe listeP= document.getElementsByTagName("p"); der = listeP[listeP.length-1]; texte= der.lastChild.data; alert(texte); |
| n_suv=n.nextSibling | renvoie le nœud suivant immédiatement le nœud n, dans la liste des enfants (childNodes) de son nœud parent (ou null s'il est le dernier. | // dans un tableau, parcourir une rangée tr0 = document.getElementById("tr0"); for (td=tr0.firstChild; td != null; ts.nextSibling) ch += "" + ts.firstChild.nodeValue + ""; |
| n_prev=n.previousSibling | renvoie le nœud précédant directement le nœud n dans la liste des enfants de son nœud parent. (autrement dit le précédent dans la "fratrie") | // trouver la cellule suivante de celle identifiée "td2" td2 = document.getElementById("td2"); td1 = td2.previousSibling; |
| chaine=n.innerHTML; n.innerHTML="code html" | récupère ou affecte l'ensemble du balisage et du texte contenu à partir du nœud n (extension du DOM2) | // remplace le texte du 1er paragraphe par une liste html n = document.getElementsByTagName("p")[0]; n.innerHTML="item"; // met body à vide ! document.body.innerHTML = ""; |
| nv = | crée un nouveau | maBaliseH1 = |

| | | |
|---|---|--|
| document.createElement(type) | nœud nv associé à une balise du type balise HTML | <code>document.createElement("h1");</code> |
| nv = document.createTextNode(chaine) | crée un nouveau nœud nv de type texte, avec le texte chaine (sans le placer) | <code>maBaliseH1 = document.createElement("h1"); monTitre = document.createTextNode("Introduction au DOM 2");</code> |
| parent.appendChild(enf) | place l'objet enf comme un enfant de l'objet parent | <code>maBaliseH1 = document.createElement("h1"); monTitre = document.createTextNode("Introduction au DOM 2"); maBaliseH1.appendChild(monTitre); document.body.appendChild(maBaliseH1);</code> |
| parent.insertBefore(enf, frere) | insère l'objet enf en tant qu'enfant de l'objet parent juste avant l'enfant frere | <code>// insertion d'une image en début de 2è paragraphe p=document.getElementById("p1"); // création du noeud pour l'image im = document.createElement("img"); im.src = "image.gif"; im.align="middle"; // change le texte existant dans le paragraphe p.firstChild.nodeValue ="voici mon image .."; // insertion de l'image avant le texte p.insertBefore(im, p.firstChild);</code> |
| parent.removeChild(enf) | supprime le sous-arbre enf du noeud parent | <code>// supprimer le premier noeud item de la 1ère liste ordonnée Element1Element2Element3 ol=document.getElementsByTagName("ol")[0]; disparu = ol.removeChild(ol.firstChild); alert(disparu.firstChild.data);</code> |
| parent.replaceChild(nvEnf, enf) | remplace le nœud existant enf par le nouveau noeud nvEnf | <code>// remplacer le texte du 3ème item de la 1ère liste ordonnée LundiMardiJeudi texte = document.createTextNode("Mercredi"); noeud =</code> |

| | | |
|---------------------------------|--|---|
| | | <pre>document.getElementsByTagName("ol")[0]; item3 = noeud.childNodes[2]; // remplace le jour erroné et affiche "jeudi" disparu = item3.replaceChild(texte, item3.firstChild); alert(disparu.nodeValue);</pre> |
| attrs=n.attributes; | attrs est la liste indicée de tous les nœuds attributs | <pre>// donne le nombre d'attributs du premier noeud h1 nbAttr=document.getElementByTagName("h1")[0].attributes.length;</pre> |
| n.getAttribute(attr) | recherche la valeur de l'attribut attr du nœud balise n | <pre>body=document.body; alert("La couleur de fond est "+body.getAttribute("bgcolor")); alert("La couleur des liens est "+body.getAttribute("link"));</pre> |
| n.setAttribute(attr,val) | fixe la valeur val à l'attribut attr, en remplaçant l'ancienne valeur le cas échéant | <pre>document.body.setAttribute("bgcolor", "magenta"); document.body.setAttribute("text", "white");</pre> |
| n.style | donne un objet représentant l'attribut style de l'élément n, ce qui permet d'agir sur les valeurs de styles CSS (lecture et modification). Attention l'usage de ces attributs de style est incompatible avec les attributs traditionnels attachés aux balises ! | <pre>// lire et changer quelques styles du 1er paragraphe <p style="text-align:center; font-size:20px;color:blue">Texte</p> var p = document.getElementsByTagName("p")[0]; alert("alignement : " + p.style.textAlign+"\ncouleur : " + p.style.color+"\ntaille : " + p.style.fontSize); p.style.textAlign="right"; p.style.fontSize="30px"; p.style.color="red"; alert("alignement : " + p.style.textAlign+"\ncouleur : " + p.style.color+"\ntaille : " + p.style.fontSize);</pre> |

