Chapitre 4 : Programmation DOM

Objectifs : Manipuler la structure du DOM en JavaScript,

Prérequis : La connaissance du langage HTML, CSS et des notions de base en javascript (Les variables, structures de contrôle, fonctions et objets)

Avant la standardisation du DOM, chaque navigateur Web implémentait ses propres méthodes afin de manipuler les éléments des pages HTML. Cela se traduisait par autant d'API qu'il y avait de navigateurs. Le code JavaScript était dès lors très difficilement portable sur l'ensemble des navigateurs.

Afin d'harmoniser ces différences, un groupe de travail du W3C s'est donné pour mission de spécifier une API de manipulation de l'arbre DOM.

L'API du DOM (Document Object Model) permet d'accéder à une page Web et de manipuler son contenu, sa structure ainsi que ses styles. Le DOM fournit pour cela une représentation objet normalisée des documents, dont le contenu est arborescent. Ces derniers peuvent être des pages HTML ou des documents XML.

Les spécifications du DOM sont indépendantes de tout langage de programmation. Elles comportent différentes versions, appelées niveaux. Dans le cadre de notre ouvrage, nous détaillons son utilisation avec le langage JavaScript.

1. Structure du DOM

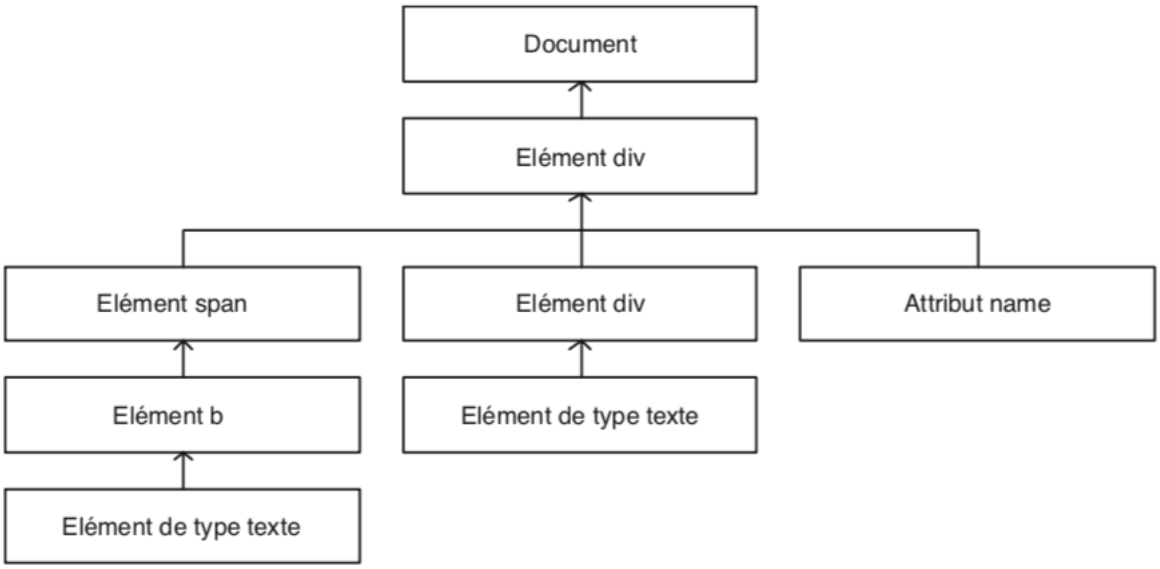
Tout langage de balises possède une structure logique arborescente dont la représentation objet correspond au DOM. Cette dernière fournit des méthodes permettant de parcourir cet arbre et

éventuellement de le modifier.

Afin de détailler la structure du DOM, nous recourons tout au long des sections suivantes à un exemple simple de fragment HTML, qui comprend une balise racine div contenant deux blocs HTML, l'un délimité par la balise **span** et l'autre par une autre balise div :

```
<div id="maZone">
        <span>
                <b>Un texte</b>
        </span>
        <div id="monAutreZone">
                Un autre texte
        </div>
</div>
```

Ce fragment de code HTML correspond à la structure hiérarchique illustrée à la figure suivante. L'objet document en est la racine. Les balises HTML sont reliées entre elles par des relations parent ou enfant.



Avec le DOM, chacune des balises HTML possède une représentation objet correspondant à son nom. Dans l'exemple ci-dessus, la balise div est représentée par un objet de type **HTMLDivElement** et la balise **span** par **HTMLSpanElement.**

Nous pouvons remarquer que les nœuds peuvent être de différents types. Dans notre exemple, le deuxième nœud représente une balise et le cinquième un attribut de cette dernière.

2. La classe Node

La classe centrale représentant un nœud DOM est **Node**. Elle définit différentes propriétés afin de donner accès aux informations relatives au nœud, telles que son type et son nom.

Le tableau suivant récapitule les principales propriétés de la classe **Node**.

Propriété	Description
attributes	Attributs du nœud
nodeName	Nom de la balise du nœud
nodeType	Type du nœud (voir la section suivante)
nodeValue	Valeur de la balise. Cette valeur peut être nulle lorsque aucune valeur n'est associée à la balise.
ownerDocument	Référence le document représentant l'arbre DOM dans lequel se trouve le nœud.

Dans notre exemple de fragment HTML, la balise identifiée par **maZone** est une div. La valeur de sa propriété **nodeType** est 1.

Cette classe va de pair avec les classes **NodeList** et **NamedNodeMap**, qui définissent des collections de nœuds. Ces dernières sont utilisées lorsque des méthodes de manipulation de l'arbre DOM renvoient plusieurs nœuds ou attributs.

Différentes syntaxes permettent de parcourir une liste de nœuds, dont la plus simple consiste à utiliser la liste à la manière d'un tableau, comme dans l'exemple suivant :

```
var elements = (...)
for( var cpt=0; cpt<elements.length; cpt++ ){
        var element = elements[cpt];
        (...)
}
```

Une autre solution, plus orientée objet, consiste à utiliser la méthode item sur l'objet **NodeList** de la manière suivante :

```
var elements = (...)
for( var cpt=0; cpt<elements.length; cpt++ ){
        var element = elements.item(cpt);
        (...)
}
```

La classe **NamedNodeMap** est similaire à **NodeList** en ce qu'elle représente une liste de nœuds. Les éléments de cette dernière sont toutefois stockés par clé. Le code suivant en donne un exemple d'utilisation avec des attributs :

```
var attributs = (...)
for( var cpt=0; cpt<attributs.length; cpt++ ){
        var attribut = attributs.item(cpt);
        var nomAttribut = attribut.name;
        var valeurAttribut = attribut.value;
        (...)
}
```

Dans la suite nous verrons les différentes techniques permettant de récupérer une liste de nœuds et d'attributs.

La classe **Node** définit en outre des propriétés référençant d'autres nœuds de l'arbre.

3. Types des nœuds

Un arbre DOM regroupe un ensemble de nœuds représentant différentes entités. Comme la classe **Node** est générique, elle contient une propriété **nodeType** dont la valeur est un entier.

Afin de faciliter l'utilisation de la propriété **nodeType**, la classe **Node** définit les constantes récapitulées au tableau suivant.

Constante	Valeur	Description
ATTRIBUT_NODE	2	Attribut d'une balise de l'arbre
COMMENT_NODE	8	Balise de commentaire
DOCUMENT_FRAGMENT_NODE	11	Nœud racine d'un fragment d'arbre
DOCUMENT_NODE	9	Nœud racine de l'arbre. Il s'agit du type de l'objet document.
ELEMENT_NODE	1	Balise de l'arbre
TEXT_NODE	3	Nœud texte de l'arbre

DOCUMENT_NODE est le type du nœud racine de l'arbre DOM d'une page HTML. Cet élément est représenté par l'objet document de la page.

Dans notre exemple, la balise d'identifiant **maZone** est de type **ELEMENT_NODE**, et son attribut id de type **ATTRIBUT_NODE**. Le texte « un autre texte » est de type **TEXT_NODE**.

4. Manipulation des éléments

Nous allons à présent détailler les différentes API permettant de manipuler les éléments d'un arbre DOM.

4.1. Accès direct aux éléments

Pour accéder au nœud d'un document afin de pouvoir le manipuler par la suite, une méthode simple consiste à identifier ce nœud par un attribut id. La méthode **getElementById()** de l'objet document peut ensuite se fonder sur ce dernier pour récupérer l'instance correspondante.

La balise div de notre bloc HTML exemple peut être référencée de la façon suivante :

```
var zone = document.getElementById("maZone");
```

Puisque l'identifiant d'une balise doit être unique dans une page HTML, la méthode **getElementById()** renvoie un unique élément.

D'autres méthodes permettent de récupérer un ensemble de références sur des nœuds en se fondant soit sur l'objet **document**, soit sur un objet représentant un nœud.

La méthode **getElementsByTagName()**, permet de récupérer une liste de nœuds en se fondant sur le nom des balises, lequel correspond à la valeur de la propriété **nodeName** de la classe **Node** décrite précédemment.

La méthode **getElementsByTagName()** peut s'appliquer sur l'objet **document** ou sur un objet **nœud**, comme dans le code suivant :

```
var balisesDiv = document.getElementsByTagName("div");
```

Ce code permet de récupérer l'ensemble des balises div d'un document sous la forme d'une liste de nœuds **(NodeList).** La valeur ***** peut être utilisée comme argument afin de récupérer l'ensemble des nœuds du document, comme dans le code suivant :

```
var balises = document.getElementsByTagName("*");
```

La méthode **getElementsByName()** retourne une liste de nœuds dont l'attribut **name** est égal à la valeur spécifiée en paramètre. Cette méthode est particulièrement appropriée lors de la manipulation de cases d'option *(radio button)* d'un formulaire HTML, comme dans l'exemple suivant :

```
<form method="post" id="monFormulaire">
        <input type="radio" name="couleur" value="rouge"/>Rouge<br/>
        <input type="radio" name="couleur" value="bleu"/>Bleu<br/>
        <input type="radio" name="couleur" value="jaune"/>Jaune<br/>
</form>
```

Le code JavaScript suivant permet de référencer directement les trois balises **input** avec le support de cette méthode :

```
var elementsInput = document.getElementsByName("couleur");
```

Les méthodes **getElementsByTagName** et **getElementsByName** permettent de récupérer une

liste de nœuds dans une hiérarchie complète. Ces méthodes parcourent l'ensemble de l'arbre à partir d'un nœud de manière récursive.

4.2. Accès aux éléments à partir d'un nœud

La classe Node permet de parcourir un arbre DOM relativement à un nœud précis. Elle possède à cet effet des propriétés spécifiques qui permettent de référencer les nœuds autour du nœud courant. Le tableau suivant récapitule ces propriétés.

Propriété	Description
childNodes	Représente la liste des nœuds enfants sous forme d'objet NodeList. La méthode hasChildNodes permet de déterminer si le nœud a des nœuds enfants. La propriété childNodes ne contient pas les attributs du nœud.
firstChild	Référence le premier nœud enfant, correspondant au premier nœud de la liste childNodes.
lastChild	Référence le dernier nœud enfant, correspondant au dernier nœud de la liste childNodes.
nextSibling	Pointe sur l'enfant suivant dont le parent est identique. Elle contient null si le nœud est le dernier enfant.
parentNode	Référence le nœud parent.
previousSibling	Pointe sur l'enfant précédent dont le parent est identique. Elle contient null si le nœud est le premier enfant.

Les propriétés **firstChild, lastChild, previousSibling** et **nextSibling** permettent de s'abstraire de cette navigation en référençant directement certains nœuds.

Le code suivant donne un exemple de la façon d'accéder à ces nœuds :

```
var zone = document.getElementById("maZone");
var span = zone.firstChild;
var div = zone.lastChild;
```

L'utilisation des propriétés **previousSibling** et **nextSibling** offre la possibilité de parcourir un arbre DOM, comme dans le code suivant :

```
var zone = document.getElementById("maZone");
var element = zone.firstChild;
while( element!=null ){
        (...)
        element = element.nextSibling;
}
```

La propriété **childNodes** permet de référencer les différents enfants directs d'un nœud. Le code suivant indique comment récupérer l'ensemble des enfants du nœud d'identifiant **maZone** de notre fragment HTML :

```
var zone = document.getElementById("maZone");
var enfants = zone.childNodes;
```

Dans cet exemple, la variable enfants contient les nœuds relatifs aux balises **span** et div contenues dans la balise div d'identifiant **maZone**.

4.3. Manipulation des nœuds

DOM définit des méthodes permettant de créer, d'ajouter, de remplacer ou de supprimer des nœuds de tout type. Ces méthodes sont fournies par l'objet document, qui décline les méthodes de création pour tous les types de nœuds.

Le code suivant fournit un exemple de création d'une balise, d'un nœud de type texte et d'un commentaire HTML :

```
var element = document.createElement("label");
var elementTexte = document.createTextNode("mon texte");
var commentaire = document.createComment("mon commentaire");
```

Une fois le nœud créé, il ne reste plus qu'à l'attacher à un nœud existant en utilisant les méthodes **appendChild()** ou **insertBefore()**, qui ajoutent un nœud enfant sous une balise.

Le code suivant se fonde sur notre code HTML exemple pour ajouter dynamiquement une balise label une fois la page chargée :

```
var zone = document.getElementById("maZone");
var label = document.createElement("label");
var texte = document.createTextNode("mon label");
label.appendChild(texte);
zone.appendChild(label);
```

Ce code permet de modifier l'arbre DOM en mémoire de la page afin d'obtenir le résultat suivant :

```
<div id="maZone">
        <span><b>Un texte</b></span>
        <div id="monAutreZone">
                Un autre texte
        </div>
        <label>mon label</label>
 </div>
```

Notons que la méthode **appendChild** ne permet pas de spécifier à quel endroit le nœud enfant est ajouté. Ce dernier est toujours ajouté à la fin de la liste des enfants.

Lorsque nous souhaitons insérer dans un tableau une nouvelle ligne entre deux lignes existantes, l'utilisation de **insertBefore** est requise. Dans l'exemple précédent, cette méthode permet d'insérer la balise label directement en dessous de la balise div d'identifiant **maZone**, comme dans le code suivant :

```
var zone = document.getElementById("maZone");
var autreZone = document.getElementById("monAutreZone");
var label = document.createElement("label");
var texte = document.createTextNode("mon label");
label.appendChild(texte);
zone.insertBefore(autreZone, label);
```

Cet exemple permet d'obtenir l'arbre DOM suivant :

```
<div id="maZone">
        <span><b>Un texte</b></span>
        <label>mon label</label>
        <div id="monAutreZone">
                Un autre texte
        </div>
</div>
```

Tous les nouveaux nœuds que nous avons créés jusqu'à présent utilisaient les différentes méthodes **createXXX()**de la classe **Document**. Il est aussi possible d'obtenir un nouveau nœud par clonage d'un nœud existant, en utilisant la méthode **cloneNode()**. Le premier argument de cette méthode est un booléen, qui permet d'indiquer si seul le nœud doit être cloné ou si le nœud ainsi que tous ses enfants doivent l'être.

4.4. Manipulation des attributs

Tout élément du type NODE_ELEMENT peut contenir des attributs permettant de lui ajouter des informations. Un attribut possède une clé et une valeur, la valeur de la clé devant être unique pour l'élément.

Le code suivant met en œuvre les attributs dans notre fragment HTML exemple :

```
<div id="maZone">
        <span><b>Un texte</b></span>
         <div id="monAutreZone">
                Un autre texte
        </div>
</div>
```

id est un attribut de la balise div. La classe Node met à disposition la propriété **attributes** afin de lister ses différents attributs. Cette propriété est du type **NamedNodeMap**, décrit précédemment. La classe Node fournit également l'ensemble de méthodes récapitulées au tableau suivant.

Méthode	Paramètre	Description
getAttribute	Identifiant de l'attribut	Référence un attribut d'un élément en utilisant son identifiant.

hasAttribute	Identifiant de l'attribut	Détermine si un attribut est présent pour un élément.
removeAttribute	Identifiant de l'attribut	Supprime un attribut pour un élément.
setAttribute	Identifiant de l'attribut ainsi que sa valeur	Crée un attribut ou remplace un attribut existant d'un élément.

La méthode **setAttribute** offre la possibilité d'ajouter un attribut à une balise. Le code suivant donne un exemple d'utilisation de cette technique afin d'ajouter un attribut d'identifiant type pour la première balise div de notre fragment HTML :

```
var zone = document.getElementById("maZone");
zone.setAttribute("type", "UnType");
```

La méthode **getAttribute()** donne accès à la valeur d'un attribut. Dans le cas où l'attribut n'existe pas, la méthode renvoie null.

Le code suivant permet de récupérer la valeur de l'attribut type précédemment ajouté :

```
var elt = document.getElementById("d1");
var type = elt.getAttribute("type");
```

Les méthodes **hasAttribute**() et **removeAttribute()** permettent respectivement de vérifier la présence d'un attribut et de supprimer un attribut, comme dans le code suivant :

```
var zone = document.getElementById("maZone");
if( zone.hasAttribute("type") ) {
        zone.removeAttribute("type");
}
```

Les attributs peuvent être utilisés sur des nœuds afin d'ajouter des données supplémentaires accessibles pour les traitements sans impacter le rendu visuel. Ces attributs sont souvent utilisés pour stocker des éléments techniques sur le nœud.

4.5. L'attribut innerHTML

Bien que l'attribut **innerHTML**, présent sur les nœuds d'un arbre DOM, ne soit pas normalisé par le W3C, cet ouvrage se doit de le présenter. Cet attribut permet en effet de remplacer complètement le contenu d'un élément par celui spécifié dans une chaîne de caractères.

Le code ci-dessous donne un exemple d'utilisation de l'attribut **innerHTML** afin de remplacer le contenu de la balise **span** définie en début de chapitre par un ensemble de balises HTML :

```
var zone = document.getElementById("maZone");
var span = zone.getElementsByTagName("span")[0];
span.innerHTML = "<p><b>mon texte</b></p>";
```

Ce code permet de modifier l'arbre DOM en mémoire de la page afin d'obtenir le résultat suivant :

```
var zone = document.getElementById("maZone");
var span = zone.getElementsByTagName("span")[0];
span.innerHTML = "<p><b>mon texte</b></p>";
```