JS : Manipuler des documents HTML

L2 MPCIE - UE Développement Web

David Lesaint

david.lesaint@univ-angers.fr

Janvier 2019

Manipuler des documents HTML

Les parties importantes d'un navigateur

Trois types d'objets JS fondamentaux (alias interfaces)

`navigator`

- L'état et l'identité du navigateur (alias user-agent).
- Pour récupérer des données de géolocalisation, le langage préféré de l'utilisateur, le flux média de la webcam, ...

`window`

- La fenêtre (ou onglet) dans lequel la page est chargée.
- Pour redimensionner la fenêtre, lui associer des gestionnaires d'évènements, stocker des données spécifiques à la page ...

`document`

- Le document chargé.
- Pour obtenir une référence d'élément HTML, en changer le contenu texte, en modifier le style, pour créer/ajouter/supprimer des éléments.

L'objet `window`

Objet global représentant la fenêtre du navigateur

Il n'est pas nécessaire de le spécifier, sauf lorsqu'on manipule des (i)frames.

dom-window.html

```
1  <!DOCTYPE html>
2  <html><head><meta charset="utf-8" />
3  <title>L'objet window</title>
4  <link rel="stylesheet" href="dom.css"></head>
5  <body>
6    <script>
7      // 2 instructions équivalentes
8      alert("Hello world!");
9      window.alert("Hello world!");
10   </script>
11 </body></html>
```

L'objet `window`

Les fonctions globales ne sont pas des méthodes de `window`

`isNaN(), parseInt(), parseFloat()` ...

Une variable déclarée sans `var` est une propriété de `window`.

dom-window-1.html

```
1  <!DOCTYPE html>
2  <html><head><meta charset="utf-8" />
3  <title>L'objet window</title>
4  <link rel="stylesheet" href="dom.css"></head>
5  <body>
6    <script>
7      var texte = 'globale';
8      (function() { // IIFE (Immediately-Invoked Function Expression)
9        var texte = 'locale';
10       glob = "autre globale"; // déconseillé !
11       alert(window.texte); // 'globale'
12       alert(texte); // 'locale'
13     })();
14     alert(texte); // 'globale'
15     alert(glob); // 'autre globale'
16   </script>
17 </body></html>
```

Le DOM

API pour manipuler documents XML et HTML

- Offre une représentation structurée et orientée objet des éléments, de leurs attributs et de leur contenu.
- Permet l'ajout et la suppression d'objets.
- Permet la modification des propriétés de ces objets à l'aide de méthodes.
- Permet de répondre aux évènements utilisateur.

Mise en oeuvre

- API standardisée autour de JS dans tous les navigateurs.
- API dépendante du langage de scripts côté serveur pour la manipulation de documents XML
  - eg. les classes PHP `DOMDocument`, `DOMElement` ... .

Les normes du DOM

Recommandations du W3C

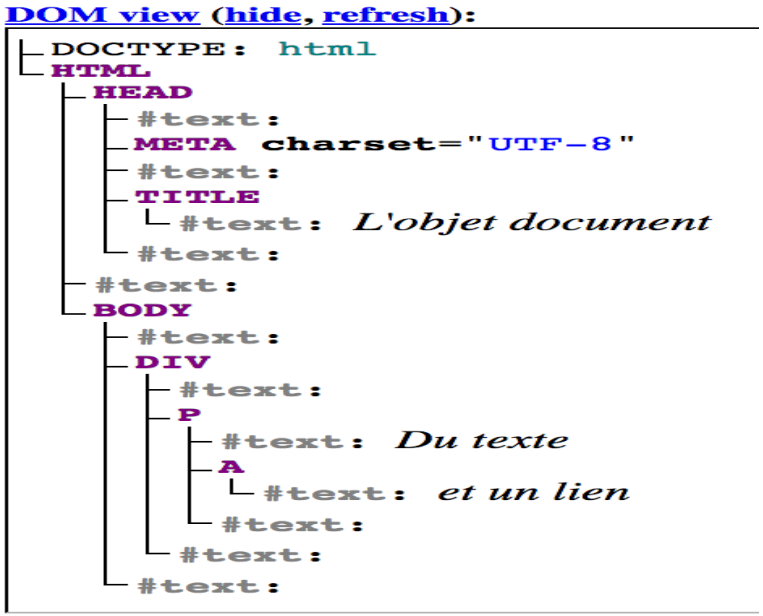
- 1998 : DOM Level 1 (Core + HTML).
- 2000 : DOM Level 2 (`getElementById`, modèle d'évènements, espaces de noms XML, CSS).
- 2004 : DOM Level 3 (support XPath, sérialisation en XML).
- 2015 : DOM Level 4 du WHATWG.
- 2015-. . . : DOM *Living Standard* du WHATWG.

Des bibliothèques permettent d'assurer la compatibilité du code JS pour différents navigateurs : eg. jQuery, prototype.

L'arbre DOM d'un document HTML

Arborescence de noeuds de différents types représentant

- Les éléments HTML.
- Leur contenu : noeud(s) texte et/ou noeud(s) élément.



dom-document.html

```
 1 <!DOCTYPE html>
 2 <html><head><meta
charset="utf-8" />
 3 <title>L'objet document</title>
 4 <link rel="stylesheet"
href="dom.css"></head>
 5 <body>
 6   <div>
 7     <p>
 8       Du texte
 9       <a>et un lien</a>
10     </p>
11   </div>
12 </body></html>
```

Les attributs d'éléments sont des noeuds non-connectés.

Types de noeuds DOM

La propriété `nodeType` renvoie le type du noeud.

Constante	Valeur	Description
`Node.ELEMENT_NODE`	`1`	Un noeud `Element` tel que `<p>` ou `<div>`.
`Node.TEXT_NODE`	`3`	Le `Text` actuel de l'`Element` ou `Attr`.
`Node.PROCESSING_INSTRUCTION_NODE`	`7`	Une `ProcessingInstruction` d'un document XML tel que la déclaration `<?xml-stylesheet ... ?>`.
`Node.COMMENT_NODE`	`8`	Un noeud `Comment`.
`Node.DOCUMENT_NODE`	`9`	Un noeud `Document`.
`Node.DOCUMENT_TYPE_NODE`	`10`	Un noeud `DocumentType` c'est-à-dire `<!DOCTYPE html>` pour des documents HTML5.
`Node.DOCUMENT_FRAGMENT_NODE`	`11`	Un noeud `DocumentFragment`.

Depuis le DOM-4, suppression des constantes

2 (ATTRIBUTE), 4 (CDATA_SECTION), 5 (ENTITY_REFERENCE), 6 (ENTITY) et 12 (NOTATION).

Noms de noeuds DOM

La propriété `nodeName` renvoie le nom du noeud.

Interface	nodeName
Attr	identique à `Attr.name`
CDATASection	"#cdata-section"
Comment	"#comment"
Document	"#document"
DocumentFragment	"#document-fragment"
DocumentType	identique à `DocumentType.name`
Element	identique à `Element.tagName`
Entity	nom de l'entité
EntityReference	nom de la référence d'entité
Notation	nom de la notation
ProcessingInstruction	identique à `ProcessingInstruction.target`
Text	"#text"

Types et noms de noeuds DOM

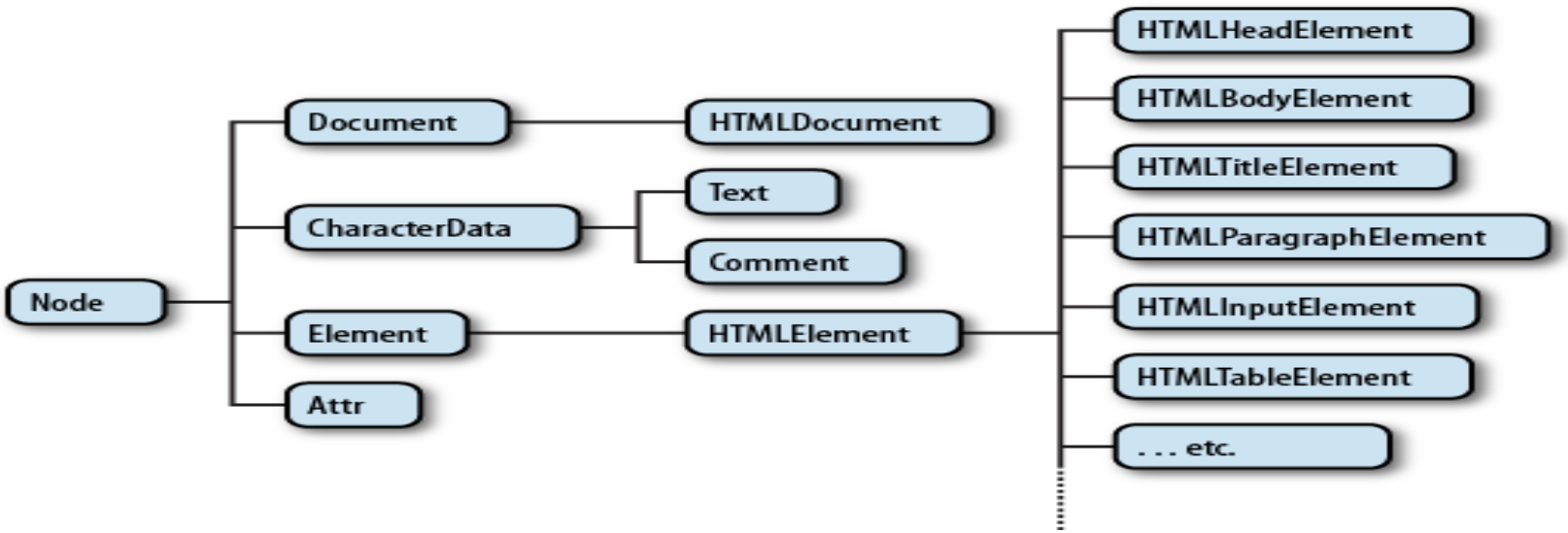
dom-nodetype.html

```
<!DOCTYPE html>
<html><head><meta charset="utf-8" />
<title>Les propriétés nodeType/nodeName</title>
<link rel="stylesheet" href="dom.css"></head>
<body>
    <div id="myDiv">
      Du texte.
    </div>
  <script>
  var myDiv = document.getElementById("myDiv");
  alert(myDiv.nodeType); // 1
  alert(myDiv.nodeName); // DIV
  var myDivId = myDiv.getAttributeNode("id");
  alert(myDivId.nodeType); // 2
  alert(myDivId.nodeName); // ID
  </script>
</body></html>
```

Interfaces DOM

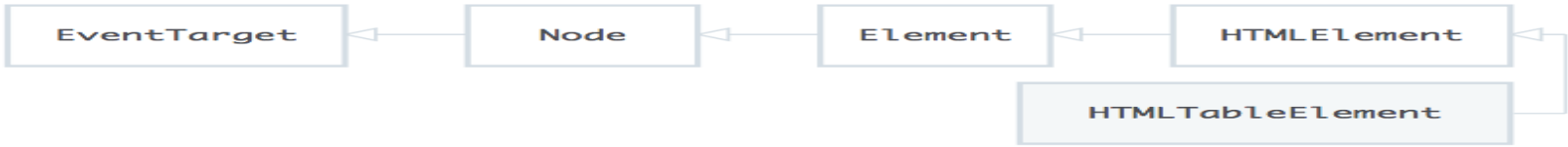
JS fournit une hiérarchie d'interfaces

Propriétés et méthodes auxquelles se conforment les différents noeuds d'un DOM selon leur type.



Chaque objet modélisant un noeud du DOM implémente les propriétés et méthodes de sa chaîne d'interfaces.

Héritage d'interfaces : exemple



dom-interfaces.html

```
1  <!DOCTYPE html>
2  <html><head><meta charset="utf-8"/>
3  <title>Interfaces DOM</title>
4  </head>
5  <body>
6    <table id="myTable">
7      <tr><td>A1</td><td>A2</td></tr>
8      <tr><td>B1</td><td>B2</td></tr>
9    </table>
10   <script>
11   var table = document.querySelector('#myTable');
12   // HTMLTableElement interface : attribut rows
13   table.rows[0].style.backgroundColor = "red";
14   // HTMLElement interface : attribut title
15   table.title = "Une aide";
16   // Element interface : attribut class
17   table.className = "mesTables";
18   // Node interface : attribut baseURI
19   alert('Base URL de la table :' + table.baseURI);
20   </script>
21 </body></html>
```

Les types d'objets importants

`document` (interface `Document`)

- L'objet correspondant au noeud racine du DOM.

`element` (interface `Element`).

- Un objet correspondant à un noeud du DOM.

`nodeList` (interface `NodeList`).

- Une collection d'`element`s statique ou dynamique.
- Traversable avec `for`, `for...of` et méthode `forEach`.
- Convertible en tableau avec `Array.from()`.

`attribute` (interface `Attr`).

- Un objet correspondant à un attribut d'`element`.

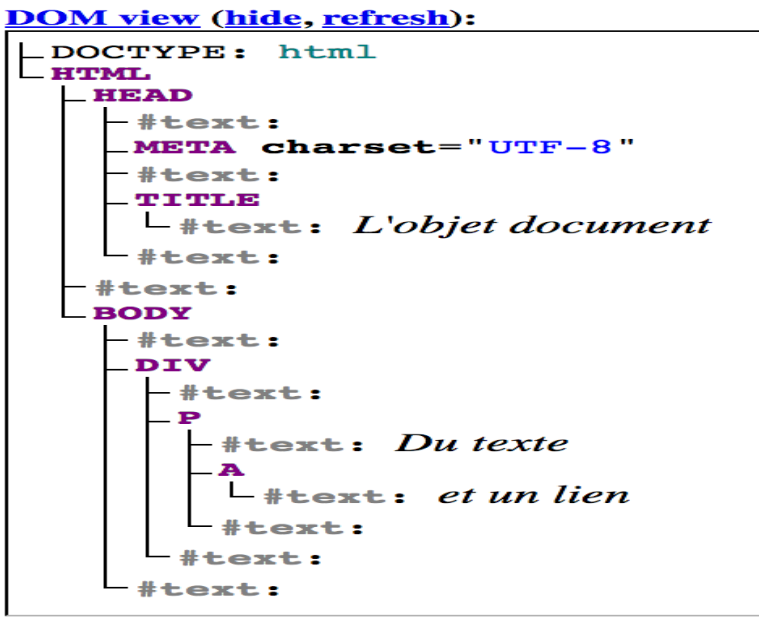
`namedNodeMap` (interface `NamedNodeMap`).

- Une collection dynamique d'`attribute`s.
- Traversable avec `for`.
- Convertible en tableau avec `Array.from()`.

L'objet `document`

Est un objet propriété de `window`

- Représente le noeud racine du DOM.
- Parent de l'élément `HTML`.
- Consulter l'outil `DOM` de Firefox ou Live DOM Viewer.



dom-document.html

```
 1 <!DOCTYPE html>
 2 <html><head><meta
charset="utf-8" />
 3 <title>L'objet document</title>
 4 <link rel="stylesheet"
href="dom.css"></head>
 5 <body>
 6   <div>
 7     <p>
 8       Du texte
 9       <a>et un lien</a>
10     </p>
11   </div>
12 </body></html>
```

Accès aux éléments HTML

Avec les méthodes d'objets `Document` ou `Element` :

`getElementById(identifiant)`

- Renvoie l'élément d'attribut `id` égal à `identifiant`.

`getElementsByTagName(balise)`

- Renvoie le tableau (`HTMLCollection`) des éléments de balise `balise`.

`getElementsByName(nom)`

- Renvoie le tableau (`HTMLCollection`) des éléments de formulaires (HTML 5) d'attribut `name` égal à `nom`.

`querySelector(sélecteur)`

- Renvoie le 1er élément correspondant au `sélecteur` CSS.

`querySelectorAll(sélecteur)`

- Renvoie le tableau d'éléments (`HTMLCollection`) correspondant au `sélecteur` CSS.

Accès aux éléments HTML

dom-getelements.html

```
<!DOCTYPE html>
<html><head><meta charset="utf-8" />
<title>Accès aux éléments HTML</title>
<link rel="stylesheet" href="dom.css"></head>
<body>
  <div id="myDiv">
    <p>
      Du texte
      <a>et un lien</a>
    </p>
  </div>
  <script>
  var div = document.getElementById("myDiv");
  alert(div);
  var divs = document.getElementsByTagName("div");
  for(var i=0, c=divs.length; i<c; i++) {
    alert("Element n\u{00B0}"+(i+1)+" : "+divs[i]);
  }
  </script>
</body></html>
```

Accès aux éléments HTML avec sélecteurs CSS3

Séquence	Signification
*	tout élément
E	tout élément de type E
E[foo]	tout élément E portant l'attribut "foo"
E[foo="bar"]	tout élément E portant l'attribut" foo" et dont la valeur de cet attribut est exactement "bar"
E[foo~="bar"]	tout élément E dont l'attribut "foo" contient une liste de valeurs séparées par des espaces, l'une de ces valeurs étant exactement égale à "bar"
E[foo^="bar"]	tout élément E dont la valeur de l'attribut "foo" commence exactement par la chaîne "bar"
E[foo$="bar"]	tout élément E dont la valeur de l'attribut "foo" finit exactement par la chaîne "bar"
E[foo*="bar"]	tout élément E dont la valeur de l'attribut "foo" contient la sous-chaîne "bar"
E[lang|="en"]	tout élément E dont l'attribut 'lang" est une liste de valeurs séparées par des tirets et commençant (à gauche) par "en"
E:root	un élément E, racine du document
E:nth-child(n)	un élément E qui est le n-ième enfant de son parent
E:nth-last-child(n)	un élément E qui est le n-ième enfant de son parent en comptant depuis le dernier enfant
E:nth-of-type(n)	un élément E qui est le n-ième enfant de son parent et de ce type
E:nth-last-of-type(n)	un élément E qui est le n-ième enfant de son parent et de ce type en comptant depuis le dernier enfant
E:first-child	un élément E, premier enfant de son parent
E:last-child	un élément E, dernier enfant de son parent
E:first-of-type	un élément E, premier enfant de son type
E:last-of-type	un élément E, dernier enfant de son type
E:only-child	un élément E, seul enfant de son parent
E:only-of-type	un élément E, seul enfant de son type

Accès aux éléments HTML avec sélecteurs CSS3

`E:empty`	un élément E qui n'a aucun enfant (y compris noeuds textuels purs)
`E:link` `E:visited`	un élément E qui est la source d'un hyperlien dont la cible n'a pas encore été visitée (:link) ou a déjà été visitée (:visited)

`E:active`
`E:hover`
`E:focus`
	un élément E pendant certaines actions de l'usager

`E:target`	un élément E qui est la cible de l'URL d'origine contenant lui-même un fragment identifiant.
`E:lang(c)`	un élément E dont le langage (humain) est c (le langage du document spécifie comment le langage humain est déterminé)
`E:enabled` `E:disabled`	un élément d'interface utilisateur E qui est actif ou inactif.
`E:checked` `E:indeterminate`	un élément d'interface utilisateur E qui est coché ou dont l'état est indéterminé (par exemple un bouton-radio ou une case à cocher)
`E:contains("foo")`	un élément E dont le contenu textuel concaténé contient la sous-chaîne "foo"
`E::first-line`	la première ligne formatée d'un élément E
`E::first-letter`	le premier caractère formaté d'un élément E
`E::selection`	la partie d'un élément E qui est actuellement sélectionnée/mise en exergue par l'usager
`E::before`	le contenu généré avant un élément E
`E::after`	le contenu généré après un élément E
`E.warning`	*Uniquement en HTML*. Identique à E[class~="warning"].
`E#myid`	un élément E dont l'ID est égal à "myid".
`E:not(s)`	un élément E qui n'est pas représenté par le sélecteur simple s
`E F`	un élément F qui est le descendant d'un élément E
`E > F`	un élément F qui est le fils d'un élément E
`E + F`	un élément F immédiatement précédé par un élément E
`E ~ F`	un élément F précédé par un élément E

Accès aux éléments HTML avec sélecteur CSS

dom-selecteur.html

```
<!DOCTYPE html>
<html><head><meta charset="utf-8" />
<title>Accès aux éléments HTML avec sélecteurs CSS</title>
<link rel="stylesheet" href="dom.css"></head>
<body>
  <div id="menu">
    <div class="item">
      <span>Elément 1</span> <span>Elément 2</span>
    </div>
    <div class="publicité">
      <span>Elément 3</span> <span>Elément 4</span>
    </div>
  </div>
  <script>
    var query = document.querySelector("#menu .item span");
    var queryAll = document.querySelectorAll("#menu .item span");
    alert(query.innerHTML); // Elément 1
    alert(queryAll.length); // 2
    // Elément 1 - Elément 2
    alert(queryAll[0].innerHTML + '-' + queryAll[1].innerHTML);
  </script>
</body></html>
```

Accès aux attributs d'éléments HTML

Avec les méthodes d'objets `Element` :

`getAttribute(att)`

- Renvoie la valeur (`string`) de l'attribut de nom `att`.

`setAttribute(att,val)`

- Fixe à `val` la valeur de l'attribut de nom `att`.

dom-attributs-acces.html

```
1  <!DOCTYPE html>
2  <html><head><meta charset="utf-8" />
3  <title>Accès aux attributs d'éléments HTML</title>
4  <link rel="stylesheet" href="dom.css"></head>
5  <body>
6    <a id="myLink" href="http://www.anywhere.com">
7      Un lien modifié dynamiquement.
8    </a>
9    <script>
10     var myLink = document.querySelector("#myLink");
11     var href = myLink.getAttribute("href");
12     alert(href);
13     myLink.setAttribute("href","http://es6-features.org");
14   </script>
15 </body></html>
```

Accès direct aux attributs d'éléments HTML

Par une propriété d'objets `Element` de même nom que l'attribut

dom-attributs-acces-direct-1.html

```
 1 <!DOCTYPE html>
 2 <html><head><meta charset="utf-8" />
 3 <title>Accès direct aux attributs d'éléments HTML</title>
 4 </head>
 5 <body>
 6   <a id="myLink" href="">un lien</a>
 7   <script>
 8     var myLink = document.querySelector("#myLink");
 9     var href = myLink.href;
10     alert(href);
11     href = "http://es6-features.org";
12   </script>
13 </body></html>
```

Accès direct aux attributs d'éléments HTML

CamelCase utilisée pour les propriétés correspondant aux

- Attributs à nom composé (eg. `readonly`) ou comportant un tiret (eg. `background-color`).
- Attributs dont le nom est réservé en JS : `class` (`className` et `classList`), `for` (`htmlFor`).

dom-attributs-acces-direct-2.html

```
1  <!DOCTYPE html>
2  <html><head><meta charset="utf-8" />
3  <title>Accès direct aux attributs d'éléments HTML</title>
4  <style>
5  .bleu { background:lightblue;} .rouge { color:red;} .noir { color:black;}
6  </style>
7  </head>
8  <body>
9    <a id="myLink" class="rouge" href="">Un lien</a>
10   <script>
11     var myLink = document.querySelector("#myLink");
12     alert("Prêt ?");
13     myLink.style.textTransform = "uppercase";
14     myLink.classList.add("bleu");
15     myLink.classList.remove("rouge");
16     myLink.classList.toggle("noir");
17   </script>
18 </body></html>
```

Récupérer le code HTML

Sous forme de chaîne avec propriété `Element.innerHTML`

dom-innerhtml.html

```
<!DOCTYPE html>
<html><head><meta charset="utf-8" />
<title>Accès au code HTML</title>
<link rel="stylesheet" href="dom.css"></head>
<body>
  <div id="myDiv">
    <p>
      Du texte
      <a>et un lien</a>
    </p>
  </div>
  <script>
  var div = document.querySelector("#myDiv");
  alert(div.innerHTML);
  </script>
</body></html>
```

Ajouter ou éditer du code HTML

Avec `Element.innerHTML`

dom-innerhtml-1.html

```
<!DOCTYPE html>
<html><head><meta charset="utf-8" />
<title>Edition/ajout de code HTML</title>
<link rel="stylesheet" href="dom.css"></head>
<body>
  <div id="myDiv">
    <p>
      Du texte
      <a>et un lien</a>
    </p>
  </div>
  <script>
  var div = document.querySelector("#myDiv");
  div.innerHTML = "<h1>Un titre écrase le paragraphe</h1>";
  div.innerHTML += "<h1>Et un second titre en ajout</h1>";
  </script>
</body></html>
```

Naviguer dans le DOM : parent

La propriété `Node.parentNode` permet d'accéder au parent d'un noeud

dom-parent.html

```
1  <!DOCTYPE html>
2  <html><head><meta charset="utf-8" />
3  <title>La propriété parentNode</title>
4  <link rel="stylesheet" href="dom.css"></head>
5  <body>
6    <blockquote>
7      <p id="myP">
8        Un paragraphe.
9      </p>
10   </blockquote>
11   <script>
12   var bq = document.querySelector("#myP").parentNode;
13   alert(bq); // object HTMLQuoteElement
14   </script>
15 </body></html>
```

Naviguer dans le DOM : enfants

`Element.firstChild` et `lastChild` donnent accès aux premier et dernier enfants d'un noeud

dom-firstchild.html

```
<!DOCTYPE html>
<html><head><meta charset="utf-8"/>
<title>Création d'éléments</title>
<link rel="stylesheet" href="dom.css"></head>
<body>
  <div>
    <p id="myP">Du texte <a>un lien</a></p>
  </div>
  <script>
  var myP = document.querySelector("#myP");
  alert(myP.firstChild.nodeName); // #text
  alert(myP.lastChild.nodeName); // A
  </script>
</body></html>
```

`firstElementChild` et `lastElementChild` donnent accès aux premier et dernier enfants qui sont des éléments HTML

Naviguer dans le DOM : contenu texte

`Node.nodeValue` et `CharacterData.data` donnent le contenu texte d'un noeud textuel

dom-nodevalue.html

```
1  <!DOCTYPE html>
2  <html><head><meta charset="utf-8" />
3  <title>Les propriétés nodeValue/data</title>
4  <link rel="stylesheet" href="dom.css"></head>
5  <body>
6    <div>
7      <p id="myP">Du texte <a>un lien</a> et <strong>en emphase</strong></p>
8    </div>
9    <script>
10   var myP = document.querySelector("#myP");
11   alert(myP.firstChild.nodeValue); // Du texte
12   alert(myP.lastChild.firstChild.data); // en emphase
13   </script>
14 </body></html>
```

Naviguer dans le DOM : tableau des enfants

`Node.childNodes` renvoie le tableau des enfants d'un noeud

dom-childnodes.html

```
<!DOCTYPE html>
<html><head><meta charset="utf-8" />
<title>La propriété childNodes</title>
<link rel="stylesheet" href="dom.css"></head>
<body>
  <div>
    <p id="myP">Du texte <a>un lien</a></p>
  </div>
  <script>
    var myP = document.querySelector("#myP");
    var children = myP.childNodes;
    for (var i = 0, c = children.length; i < c; i++) {
      if (children[i].nodeType === Node.ELEMENT_NODE) {
        alert(children[i].firstChild.data); // élément HTML
      } else {
        alert(children[i].data); // noeud texte (ici)
      }
    }
  </script>
</body></html>
```

Naviguer dans le DOM : adelphes

`Node.nextSibling` et `previousSibling` donnent accès au noeud suivant et précédent

dom-nextsibling.html

```
<!DOCTYPE html>
<html><head><meta charset="utf-8" />
<title>Les propriétés nextSibling/previousSibling</title>
<link rel="stylesheet" href="dom.css"></head>
<body>
  <div>
    <p id="myP">Du texte <a>un lien</a></p>
  </div>
  <script>
    var child = document.querySelector("#myP").lastChild;
    while (child) {
      if (child.nodeType === 1) {
        alert(child.firstChild.data); // élément HTML
      } else {
        alert(child.data); // noeud texte (ici)
      }
      child = child.previousSibling;
    }
  </script>
</body></html>
```

`nextElementSibling` et `previousElementSibling` donnent accès aux éléments HTML suivant et précédent

Naviguer dans le DOM

Attention aux noeuds texte vides !

dom-noeuds-texte-vides.html

```
<!DOCTYPE html>
<html><head><meta charset="utf-8" />
<title>Les noeuds texte vides : retours à la ligne, espaces ...</title>
<link rel="stylesheet" href="dom.css"></head>
<body>
  <div>
    <p id="myP1"><a>Une ancre</a></p>
    <p id="myP2">
      <a>Une ancre</a>
    </p>
  </div>
  <script>
    var child1 = document.querySelectorAll("p")[0].firstChild;
    var child2 = document.querySelectorAll("p")[1].firstChild;
    if (child1.nodeType !== child2.nodeType) {
      alert(child2.nodeType); // 3 (Node.TEXT_NODE)
    }
  </script>
</body></html>
```

Créer et insérer des éléments

En 3 temps

1. Création d'un élément avec la méthode `Document.createElement(tag)`.
2. Affectation des attributs avec la méthode `Element.setAttribute(att,val)`.
3. Insertion de l'élément dans le document avec la méthode `Node.appendChild(tag)`.

Création d'un noeud texte avec la méthode `Document.createTextNode(contenuTexte)`.

Créer et insérer des éléments

dom-createelement.html

```
<!DOCTYPE html>
<html><head><meta charset="utf-8" />
<title>Création et ajout d'éléments</title>
<link rel="stylesheet" href="dom.css"></head>
<body>
  <div>
    <p id="myP">Du texte </p>
  </div>
  <script>
    var myP = document.querySelector("#myP");
    var newLink = document.createElement("a");
    alert(newLink.isConnected); //false
    newLink.id = "myLink";
    newLink.href = "http://www.lemonde.fr";
    newLink.title = "Le Monde";
    newLinkText = document.createTextNode("Le site du Monde");
    newLink.appendChild(newLinkText);
    myP.appendChild(newLink);
    alert(newLink.isConnected); //true
  </script>
</body></html>
```

Cloner des éléments

Avec `Node.cloneNode(flag)`

- Si `flag=true`, clone aussi enfants et attributs du noeud.
- Si `flag=false`, ne clone ni enfants ni attributs.

Les gestionnaires d'évènement enregistrés ne sont pas clonés.

dom-clonenode.html

```
1  <!DOCTYPE html>
2  <html><head><meta charset="utf-8" />
3  <title>Clonage de noeuds</title>
4  <link rel="stylesheet" href="dom.css"></head>
5  <body>
6    <div>
7      <p id="myP">Du texte </p>
8    </div>
9    <script>
10     var myP = document.querySelector("#myP");
11     var p1 = myP.cloneNode(true);
12     var p2 = myP.cloneNode(false);
13     document.querySelectorAll("body")[0].appendChild(p1);
14     document.querySelectorAll("body")[0].appendChild(p2);
15   </script>
16 </body></html>
```

Remplacer des éléments

Avec `Node.replaceChild(newChild,oldChild)`

Remplace l'enfant `oldChild` par `newChild`.

dom-replacechild.html

```
1  <!DOCTYPE html>
2  <html><head><meta charset="utf-8" />
3  <title>Remplacement de noeud enfant</title>
4  <link rel="stylesheet" href="dom.css"></head>
5  <body>
6    <div><p>Du texte</p></div>
7    <script>
8      var myDiv = document.querySelector("div");
9      var myS = document.createElement("span");
10     myS.innerHTML = "Un &lt;span&gt; en remplacement d'un &lt;p&gt;"
11     myDiv.replaceChild(myS,myDiv.firstChild);
12   </script>
13 </body></html>
```

Supprimer des éléments

Avec `Node.removeChild()` sur noeud parent

dom-removechild.html

```
1  <!DOCTYPE html>
2  <html><head><meta charset="utf-8" />
3  <title>Suppression de noeud enfant</title>
4  <link rel="stylesheet" href="dom.css"></head>
5  <body>
6    <div><p>Du texte</p></div>
7    <script>
8      var myP = document.querySelector("div > p");
9      myP.parentNode.removeChild(myP);
10     /* alternative
11     var myDiv = document.querySelector("div");
12     myDiv.removeChild(myP);
13     */
14   </script>
15 </body></html>
```

Insérer des éléments

Avec `Node.insertBefore()` sur noeud adelphe

dom-insertbefore.html

```
<!DOCTYPE html>
<html><head><meta charset="utf-8" />
<title>Insertion d'éléments</title>
<link rel="stylesheet" href="dom.css">
</head>
<body>
  <div><p>Du texte</p></div>
  <script>
    var myDiv = document.querySelector("div");
    var newP = document.createElement("p");
    newPText = document.createTextNode("Voici ");
    newP.appendChild(newPText);
    if(myDiv.hasChildNodes)
      myDiv.insertBefore(newP, myDiv.lastChild);
  </script>
</body></html>
```