جامعة شعيب الدكالي
Université Chouaïb Doukkali

Faculté des Sciences El Jadida

Technologie du Web

Prof. Mohamed NABIL

Département Informatique

Année Universitaire 2022/2023

Langage JavaScript && DOM

1. Nœuds DOM JavaScript

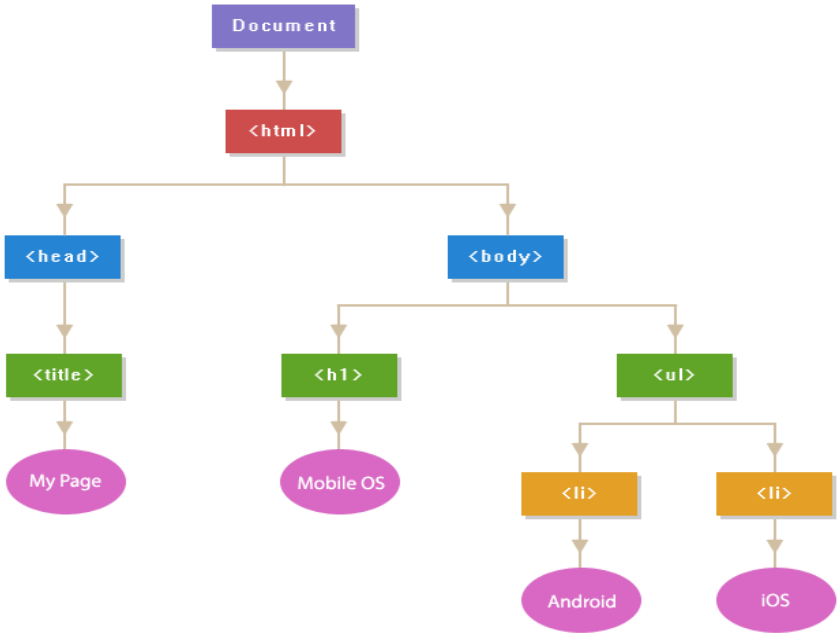
1.1. Comprendre le modèle d'objet de document

Le modèle d'objet de document, DOM (Document Object Model), est un modèle indépendant de la plate-forme et du langage pour représenter les documents HTML. Il définit la structure logique des documents et la manière dont ils peuvent être consultés et manipulés par un programme d'application. Dans le DOM, toutes les parties du document, telles que les éléments, les attributs, le texte, etc. sont organisées dans une structure arborescente hiérarchique; semblable à un arbre généalogique dans la vraie vie qui se compose de parents et d'enfants. Dans la terminologie DOM, ces différentes parties du document sont appelées nœuds. Le modèle d'objet de document qui représente un document HTML est appelé DOM HTML (en anglais HTML DOM). De même, le DOM qui représente le document XML est appelé DOM XML (en anglais XML DOM).

Dans ce chapitre, nous aborderons le DOM HTML qui fournit une interface standard pour accéder et manipuler des documents HTML via JavaScript. Avec le DOM HTML, vous pouvez utiliser JavaScript pour créer des documents HTML, naviguer dans leur structure hiérarchique, ajouter, modifier ou supprimer des éléments et des attributs ou leur contenu, etc. Presque tout ce qui se trouve dans un document HTML peut être consulté, modifié, supprimé ou ajouté en utilisant JavaScript avec l'aide de DOM HTML. Pour comprendre cela plus clairement, considérons le document HTML simple suivant:

```
<!DOCTYPE html>
<html>
<head>
	<title>My Page</title>
</head>
<body>
	<h1>Mobile OS</h1>
	<ul>
		<li>Android</li>
		<li>iOS</li>
	</ul>
</body>
</html>
```

Le document HTML ci-dessus peut être représenté par l'arborescence DOM suivante:



Le diagramme ci-dessus illustre les relations **parent/enfant** entre les nœuds. Le nœud le plus haut (le nœud Document) est le nœud racine de l'arborescence DOM, qui a un enfant, l'élément <**html**>. Alors que les éléments <**head**> et <**body**> sont les nœuds enfants du nœud parent <**html**>. Les éléments <**head**> et <**body**> sont également frères car ils sont au même niveau. En outre, le contenu textuel à l'intérieur d'un élément est un nœud enfant de l'élément parent. Ainsi, par exemple, "Mobile OS" est considéré comme un nœud enfant du <**h1**> qui le contient, et ainsi de suite.

Les commentaires à l'intérieur du document HTML sont également des nœuds dans l'arborescence DOM, même si cela n'affecte en aucune façon la représentation visuelle du document. Les commentaires sont utiles pour documenter le code, cependant, vous aurez rarement besoin de les récupérer et de les manipuler.

Les attributs HTML tels que **id, classe, titre, style, …** sont également considérés comme des nœuds dans la hiérarchie DOM mais ils ne participent pas aux relations parent/enfant comme le font les autres nœuds. Ils sont accessibles en tant que propriétés du nœud d'élément qui les contient.

Chaque élément d'un document HTML tel qu'une image, un hyperlien, un formulaire, un bouton, un titre, un paragraphe, etc. est représenté à l'aide d'un objet JavaScript dans la hiérarchie DOM, et chaque objet contient des propriétés et des méthodes pour décrire et manipuler ces objets. Par exemple, la propriété style des éléments DOM peut être utilisée pour obtenir ou définir le style en ligne d'un élément.

Le DOM est une représentation des différents composants du navigateur et du document Web actuel (HTML) qui peut être consulté ou manipulé à l'aide d'un langage de script tel que JavaScript.

2. Sélecteurs DOM JavaScript

2.1. Sélection d'éléments DOM dans JavaScript

JavaScript est le plus couramment utilisé pour obtenir ou modifier le contenu ou la valeur des éléments HTML sur la page, ainsi que pour appliquer certains effets tels que afficher, masquer, animations, etc. Mais, avant de pouvoir effectuer une action, vous devez rechercher ou sélectionner l'élément HTML cible.

2.2. Sélection des éléments les plus hauts

Les éléments les plus élevés d'un document HTML sont disponibles directement en tant que propriétés du document. Par exemple, l'élément <**html**> est accessible avec la propriété **document.documentElement**, tandis que l'élément <**head**> est accessible avec la propriété **document.head** et l'élément <**body**> est accessible avec la propriété **document.body**. Voici un exemple:

```
<!DOCTYPE html>
<html lang="en">
<head>
      <meta charset="utf-8">
      <title>JS Select Topmost Elements</title>
</head>
<body>
      <script>
      // Display lang attribute value of html element
      alert(document.documentElement.getAttribute("lang"));//
      Outputs: en
      // Set background color of body element
      document.body.style.background = "yellow";
      // Display tag name of the head element's first child
      alert(document.head.firstElementChild.nodeName);//Outputs:
      meta
      </script>
</body>
</html>
```

Mais faites attention. Si **document.body** est utilisé avant la balise <**body**> (par exemple à l'intérieur de <**head**>), il retournera **null** au lieu de l'élément body. Étant donné que le point auquel le script est exécuté, la balise <**body**> n'a pas été analysée par le navigateur, donc **document.body** est vraiment nul à ce stade. L'exemple suivant permet de comprendre cela:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>JS Document.body Demo</title>
<script> alert("From HEAD: " + document.body); // Outputs: From
```

```
HEAD: null (since <body> is not parsed yet)
</script>
</head>
<body>
<script>
alert("From BODY: " + document.body); // Outputs: From BODY:
[object HTMLBodyElement]
</script>
</body>
</html>
```

2.3. Sélection d'éléments par ID

Vous pouvez sélectionner un élément en fonction de son ID unique avec la méthode **getElementById**(). C'est le moyen le plus simple de trouver un élément HTML dans l'arborescence DOM. L'exemple suivant sélectionne et met en surbrillance un élément ayant l'attribut ID **id = "mark".**

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>JS Select Element by ID</title>
</head>
<body>
<p id="mark">This is a paragraph of text.</p>
<p>This is another paragraph of text.</p>
<script>
// Selecting element with id mark
var match = document.getElementById("mark");
// Highlighting element's background
match.style.background = "yellow";
</script>
</body>
</html>
```

La méthode **getElementById**() retournera l'élément en tant qu'objet si l'élément correspondant a été trouvé, ou null si aucun élément correspondant n'a été trouvé dans le document.

2.4. Sélection d'éléments par nom de classe

De même, vous pouvez utiliser la méthode **getElementsByClassName()** pour sélectionner tous les éléments ayant des noms de classe spécifiques. Cette méthode renvoie un objet de

type tableau de tous les éléments enfants qui ont tous les noms de classe donnés. Regardons l'exemple suivant:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>JS Select Elements by Class Name</title>
</head>
<body>
<p class="test">This is a paragraph of text.</p>
<div class="test">This is another paragraph of text.</div>
<p>This is one more paragraph of text.</p>
<script>
// Selecting elements with class test
var matches = document.getElementsByClassName("test");
// Displaying the selected elements count
document.write("Number of selected elements: " + matches.length);
// Applying bold style to first element in selection
matches[0].style.fontWeight = "bold";
// Applying italic style to last element in selection
matches[matches.length - 1].style.fontStyle = "italic";
// Highlighting each element's background through loop
for(var elem in matches) {
matches[elem].style.background = "yellow";
}
</script>
</body>
</html>
```

2.5. Sélection d'éléments par nom de balise

Vous pouvez également sélectionner des éléments HTML par nom de balise en utilisant la méthode **getElementsByTagName**(). Cette méthode renvoie également un objet de type tableau de tous les éléments enfants avec le nom de balise donné.

```
<!DOCTYPE html>
<html lang="en">
<head>
      <meta charset="utf-8">
      <title>JS Select Elements by Tag Name</title>
```

```
</head>
<body>
<p>This is a paragraph of text.</p>
<div class="test">This is another paragraph of text.</div>
<p>This is one more paragraph of text.</p>
<script>
// Selecting all paragraph elements
var matches = document.getElementsByTagName("p");
// Printing the number of selected paragraphs
document.write("Number of selected elements: " + matches.length);
// Highlighting each paragraph's background through loop
for(var elem in matches) {
matches[elem].style.background = "yellow";
}
</script>
</body>
</html>
```

2.6. Sélection d'éléments avec des sélecteurs CSS

Vous pouvez utiliser la méthode **querySelectorAll**() pour sélectionner des éléments qui correspondent au sélecteur CSS spécifié. Les sélecteurs CSS offrent un moyen très puissant et efficace de sélectionner des éléments HTML dans un document. Cette méthode renvoie une liste de tous les éléments qui correspondent aux sélecteurs spécifiés. Vous pouvez l'examiner comme n'importe quel tableau, comme indiqué dans l'exemple suivant:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>JS Select Elements with CSS Selectors</title>
</head>
<body>
<ul>
<li>Bread</li>
<li class="tick">Coffee</li>
<li>Pineapple Cake</li>
</ul>
<script>
// Selecting all li elements
```

```
var matches = document.querySelectorAll("ul li");
// Printing the number of selected li elements
document.write("Number of selected elements: " + matches.length + "<hr>")
// Printing the content of selected li elements
for(var elem of matches) {
document.write(elem.innerHTML + "<br>");
}
// Applying line through style to first li element with class tick
matches = document.querySelectorAll("ul li.tick");
matches[0].style.textDecoration = "line-through";
</script>
</body>
</html>
```

3. Style DOM JavaScript

3.1. Styliser les éléments DOM en JavaScript

Vous pouvez également appliquer un style aux éléments HTML pour modifier la présentation visuelle des documents HTML de manière dynamique à l'aide de JavaScript. Vous pouvez définir presque tous les styles pour les éléments tels que les polices, les couleurs, les marges, les bordures, les images d'arrière-plan, l'alignement du texte, la largeur et la hauteur, la position, etc.

3.2. Définition de styles en ligne sur les éléments

Les styles en ligne sont appliqués directement à l'élément HTML spécifique à l'aide de l'attribut **style**. En JavaScript, la propriété style est utilisée pour obtenir ou définir le style en ligne d'un élément. L'exemple suivant définira la couleur et les propriétés de police d'un élément avec id = "intro".

```
<!DOCTYPE html>
<html lang="en">
<head>
	<meta charset="utf-8">
	<title>JS Set Inline Styles Demo</title>
</head>
<body>
	<p id="intro">This is a paragraph.</p>
	<p>This is another paragraph.</p>
	<script>
		// Selecting element
		var elem = document.getElementById("intro");
		// Appling styles on element
```

```
            elem.style.color = "blue";
            elem.style.fontSize = "18px";
            elem.style.fontWeight = "bold";
      </script>
</body>
</html>
```

De nombreuses propriétés CSS, telles que **font-size, background-image, text-decoration, etc**. contiennent des tirets **(-)** dans leurs noms. Puisque, en JavaScript, le trait d'union est un opérateur réservé et il est interprété comme un signe moins, il n'est donc pas possible d'écrire une expression, comme: **elem.style.font-size**. Par conséquent, en JavaScript, les noms de propriété CSS qui contiennent un ou plusieurs traits d'union sont convertis en mot de style intercapitalisé. Cela se fait en supprimant les tirets et en mettant en majuscule la lettre immédiatement après chaque tiret, ainsi la propriété CSS **font-size** devient la propriété DOM **fontSize**, **border-left-style** devient **borderLeftStyle**, et ainsi de suite.

3.3. Obtention d'informations de style à partir d'éléments

De même, vous obtenez les styles appliqués aux éléments HTML à l'aide de la propriété **style**. L'exemple suivant obtiendra les informations de style de l'élément ayant **id** = "intro".

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>JS Get Element's Style Demo</title>
</head>
<body>
<p id="intro" style="color:red; font-size:20px;">This is a paragraph.</p>
<p>This is another paragraph.</p>
<script>
// Selecting element
var elem = document.getElementById("intro");
// Getting style information from element
alert(elem.style.color); // Outputs: red
alert(elem.style.fontSize); // Outputs: 20px
alert(elem.style.fontStyle); // Outputs nothing
</script>
</body>
</html>
```

La propriété **style** n'est pas très utile lorsqu'il s'agit d'obtenir des informations de style à partir des éléments, car elle ne renvoie que les règles de style définies dans l'attribut style de

l'élément et non celles qui viennent d'ailleurs, telles que les règles de style dans les feuilles de style incorporées, ou feuilles de style externes. Pour obtenir les valeurs de toutes les propriétés CSS qui sont réellement utilisées pour rendre un élément, vous pouvez utiliser la méthode **window.getComputedStyle()**, comme indiqué dans l'exemple suivant:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>JS Get Computed Style Demo</title>
<style type="text/css">
      #intro {
            font-weight: bold;
            font-style: italic;
      }
</style>
</head>
<body>
<p id="intro" style="color:red; font-size:20px;">This is a paragraph.</p> <p>This is another paragraph.</p>
<script>
// Selecting element
var elem = document.getElementById("intro");
// Getting computed style information
var styles = window.getComputedStyle(elem);
alert(styles.getPropertyValue("color")); // Outputs: rgb(255, 0, 0)
alert(styles.getPropertyValue("font-size")); // Outputs: 20px
alert(styles.getPropertyValue("font-weight")); // Outputs: 700
alert(styles.getPropertyValue("font-style")); // Outputs: italic
</script>
</body>
</html>
```

La valeur **700** de la propriété CSS **font-weight** est la même que celle du mot-clé **bold**. Le mot-clé de couleur **red** est le même que **rgb (255,0,0)**, qui est la notation RVB d'une couleur.

3.4. Ajout de classes CSS aux éléments

Vous pouvez également obtenir ou définir des classes CSS sur les éléments HTML à l'aide de la propriété **className**. Puisque, **class** est un mot réservé en JavaScript, JavaScript utilise la propriété **className** pour faire référence à la valeur de l'attribut de classe HTML. L'exemple suivant montre comment ajouter une nouvelle classe ou remplacer toutes les classes existantes à un élément <**div**> ayant **id** = "info".

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>JS Add or Replace CSS Classes Demo</title>
<style>
      .highlight {
            background: yellow;
      }
</style>
</head>
<body>
<div id="info" class="disabled">Something very important!</div>
<script>
// Selecting element
var elem = document.getElementById("info");
elem.className = "note";
// Add or replace all classes with note class
elem.className += " highlight";
// Add a new class highlight
</script>
</body>
</html>
```

Il existe encore une meilleure façon de travailler avec les classes CSS. Vous pouvez utiliser la propriété **classList** pour obtenir, définir ou supprimer facilement des classes CSS d'un élément. Cette propriété est prise en charge dans tous les principaux navigateurs, à l'exception d'Internet Explorer avant la version 10. Voici un exemple:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>JS classList Demo</title>
<style>
      .highlight {
            background: yellow;
      }
</style>
```

```
</head>
<body>
<div id="info" class="disabled">Something very important!</div>
<script>
// Selecting element
var elem = document.getElementById("info");
elem.classList.add("hide"); // Add a new class
elem.classList.add("note", "highlight"); // Add multiple classes
elem.classList.remove("hide"); // Remove a class
elem.classList.remove("disabled", "note"); // Remove multiple classes
elem.classList.toggle("visible"); // If class exists remove it, if not add it
// Determine if class exist
if(elem.classList.contains("highlight")) {
      alert("The specified class exists on the element.");
}
</script>
</body>
</html>
```

4. DOM JavaScript obtient et met les attributs

JavaScript fournit plusieurs méthodes pour ajouter, supprimer ou modifier l'attribut d'un élément HTML.

4.1. Obtenir la valeur d'attribut de l'élément

La méthode **getAttribute()** est utilisée pour obtenir la valeur actuelle d'un attribut sur l'élément. Si l'attribut spécifié n'existe pas sur l'élément, il renverra **null**. Voici un exemple:

```
<a href="https://www.google.com/" target="_blank" id="myLink"> Google
</a>
<script>
// Selecting the element by ID attribute
var link = document.getElementById("myLink");
// Getting the attributes values
var href = link.getAttribute("href");
alert(href); // Outputs: https://www.google.com/
var target = link.getAttribute("target");
alert(target); // Outputs: _blank
</script>
```

4.2. Définition des attributs sur les éléments

La méthode **setAttribute()** est utilisée pour définir un attribut sur l'élément spécifié. Si l'attribut existe déjà sur l'élément, la valeur est mise à jour; sinon, un nouvel attribut est ajouté avec le nom et la valeur spécifiés. Le code JavaScript de l'exemple suivant ajoutera une classe et un attribut désactivé (disabled) à l'élément <**button**>.

```
<button type="button" id="myBtn">Click Me</button>
<script>
// Selecting the element
var btn = document.getElementById("myBtn");
// Setting new attributes
btn.setAttribute("class", "click-btn");
btn.setAttribute("disabled", "");
</script>
```

De même, vous pouvez utiliser la méthode **setAttribute()** pour mettre à jour ou modifier la valeur d'un attribut existant sur un élément HTML. Le code JavaScript de l'exemple suivant mettra à jour la valeur de l'attribut **href** existant d'un élément d'ancrage (<**a**>).

```
<a href="#" id="myLink">Tutorial Republic</a>
<script>
// Selecting the element
var link = document.getElementById("myLink");
// Changing the href attribute value
link.setAttribute("href", "https://www.tutorialrepublic.com");
</script>
```

4.3. Suppression d'attributs d'éléments

La méthode **removeAttribute()** est utilisée pour supprimer un attribut de l'élément spécifié. Le code JavaScript de l'exemple suivant supprimera l'attribut **href** d'un élément d'ancrage.

```
<a href="https://www.google.com/" id="myLink">Google</a>
<script>
// Selecting the element
var link = document.getElementById("myLink");
// Removing the href attribute
link.removeAttribute("href");
</script>
```

5. Manipulation du DOM JavaScript

Jusqu'à maintenant nous avons appris à sélectionner et à styliser les éléments DOM HTML. Dans cette partie, nous allons apprendre comment ajouter ou supprimer des éléments DOM dynamiquement, obtenir leur contenu, etc.

5.1. Ajout de nouveaux éléments au DOM

Vous pouvez créer explicitement un nouvel élément dans un document HTML à l'aide de la méthode **document.createElement()**. Cette méthode crée un nouvel élément, mais ne l'ajoute pas au DOM; vous devrez le faire dans une étape distincte, comme indiqué dans l'exemple suivant:

```
<div id="main">
<h1 id="title">Hello World!</h1>
<p id="hint">This is a simple paragraph.</p>
</div>
<script>
// Creating a new div element
var newDiv = document.createElement("div");
// Creating a text node
var newContent = document.createTextNode("Hi, how are you doing?");
// Adding the text node to the newly created div
newDiv.appendChild(newContent);
// Adding the newly created element and its content into the DOM
var currentDiv = document.getElementById("main");
document.body.appendChild(newDiv, currentDiv);
</script>
```

La méthode **appendChild()** ajoute le nouvel élément à la fin de tous les autres enfants d'un nœud parent spécifié. Cependant, si vous souhaitez ajouter le nouvel élément au début de tout autre enfant, vous pouvez utiliser la méthode **insertBefore()**, comme illustré dans l'exemple ci-dessous:

```
<div id="main">
<h1 id="title">Hello World!</h1>
<p id="hint">This is a simple paragraph.</p>
</div>
<script>
// Creating a new div element
var newDiv = document.createElement("div");
// Creating a text node
var newContent = document.createTextNode("Hi, how are you doing?");
// Adding the text node to the newly created div
newDiv.appendChild(newContent);
// Adding the newly created element and its content into the DOM
var currentDiv = document.getElementById("main");
document.body.insertBefore(newDiv, currentDiv);
</script>
```

5.2. Obtenir ou définir du contenu HTML sur DOM

Vous pouvez également obtenir ou définir facilement le contenu des éléments HTML avec la propriété **innerHTML**. Cette propriété définit ou obtient le balisage HTML contenu dans l'élément, c'est-à-dire le contenu entre ses balises d'ouverture et de fermeture. Consultez l'exemple suivant pour voir comment cela fonctionne:

```
<div id="main">
<h1 id="title">Hello World!</h1>
<p id="hint">This is a simple paragraph.</p>
</div>
<script>
// Getting inner HTML conents
var contents = document.getElementById("main").innerHTML;
alert(contents); // Outputs inner html contents
// Setting inner HTML contents
var mainDiv = document.getElementById("main");
mainDiv.innerHTML = "<p>This is <em>newly inserted</em> paragraph.</p>";
</script>
```

Comme vous pouvez voir avec quelle facilité vous pouvez insérer de nouveaux éléments dans DOM à l'aide de la propriété **innerHTML**, mais il y a un problème, la propriété **innerHTML** remplace tout le contenu existant d'un élément. Donc, si vous souhaitez insérer le HTML dans le document sans remplacer le contenu existant d'un élément, vous pouvez utiliser la méthode **insertAdjacentHTML()**. Cette méthode accepte deux paramètres: la position dans laquelle insérer et le texte HTML à insérer. La position doit être l'une des valeurs suivantes: "**beforebegin**", "**afterbegin**", "**beforeend**" et "**afterend**". Cette méthode est prise en charge dans tous les principaux navigateurs. L'exemple suivant montre la visualisation des noms de position et son fonctionnement.

```
<!-- beforebegin -->
<div id="main">
<!-- afterbegin -->
<h1 id="title">Hello World!</h1>
<!-- beforeend -->
</div>
<!-- afterend -->
<script>
// Selecting target element
var mainDiv = document.getElementById("main");
// Inserting HTML just before the element itself, as a previous sibling
mainDiv.insertAdjacentHTML('beforebegin', '<p>This is paragraph
```

```
one.</p>');
// Inserting HTML just inside the element, before its first child
mainDiv.insertAdjacentHTML('afterbegin', '<p>This is paragraph two.</p>');
// Inserting HTML just inside the element, after its last child
mainDiv.insertAdjacentHTML('beforeend', '<p>This is paragraph three.</p>');
// Inserting HTML just after the element itself, as a next sibling
 mainDiv.insertAdjacentHTML('afterend', '<p>This is paragraph four.</p>');
</script>
```

Les positions **beforebegin** et **afterend** ne fonctionnent que si le nœud est dans l'arborescence DOM et a un élément parent.

5.3. Suppression des éléments existants du DOM

De même, vous pouvez utiliser la méthode **removeChild()** pour supprimer un nœud enfant du DOM. Cette méthode renvoie également le nœud supprimé. Voici un exemple:

```
<div id="main">
<h1 id="title">Hello World!</h1>
<p id="hint">This is a simple paragraph.</p>
</div>
<script>
var parentElem = document.getElementById("main");
var childElem = document.getElementById("hint");
parentElem.removeChild(childElem);
</script>
```

Il est également possible de supprimer l'élément enfant sans connaître exactement l'élément parent. Recherchez simplement l'élément enfant et utilisez la propriété **parentNode** pour trouver son élément parent. Cette propriété renvoie le parent du nœud spécifié dans l'arborescence DOM. Voici un exemple:

```
<div id="main">
<h1 id="title">Hello World!</h1>
<p id="hint">This is a simple paragraph.</p>
</div>
<script>
var childElem = document.getElementById("hint");
childElem.parentNode.removeChild(childElem);
</script>
```

5.4. Remplacement des éléments existants dans DOM

Vous pouvez également remplacer un élément dans HTML DOM par un autre en utilisant la méthode **replaceChild()**. Cette méthode accepte deux paramètres: le nœud à insérer et le nœud à remplacer. Il a la syntaxe comme **parentNode.replaceChild (newChild, oldChild);**. Voici un exemple:

```
<div id="main">
<h1 id="title">Hello World!</h1>
<p id="hint">This is a simple paragraph.</p>
</div>
<script>
var parentElem = document.getElementById("main");
var oldPara = document.getElementById("hint");
// Creating new elememt
var newPara = document.createElement("p");
var newContent = document.createTextNode("This is a new paragraph.");
newPara.appendChild(newContent);
// Replacing old paragraph with newly created paragraph
parentElem.replaceChild(newPara, oldPara);
</script>
```

6. Navigation dans JavaScript DOM

Le nœud DOM fournit plusieurs propriétés et méthodes qui vous permettent de naviguer ou de parcourir l'arborescence du DOM et d'apporter des modifications très facilement. Dans la section suivante, nous allons apprendre à naviguer vers le haut, le bas et les côtés dans l'arborescence DOM à l'aide de JavaScript.

6.1. Accès aux nœuds enfants

Vous pouvez utiliser les propriétés **firstChild** et **lastChild** du nœud DOM pour accéder respectivement au premier et au dernier nœud enfant direct d'un nœud. Si le nœud n'a aucun élément enfant, il renvoie **null**.

```
<div id="main">
<h1 id="title">My Heading</h1>
<p id="hint"><span>This is some text.</span></p>
</div>
<script>
var main = document.getElementById("main");
console.log(main.firstChild.nodeName); // Prints: #text
var hint = document.getElementById("hint");
console.log(hint.firstChild.nodeName); // Prints: SPAN
</script>
```

Le **nodeName** est une propriété en lecture seule qui renvoie le nom du nœud actuel sous forme de chaîne. Par exemple, il renvoie le nom de la balise pour le nœud d'élément, #text pour le nœud de texte, #comment pour le nœud de commentaire, #document pour le nœud de document, etc.

Si vous remarquez l'exemple ci-dessus, le **nodeName** du nœud premier enfant de l'élément **DIV** principal renvoie **#text** au lieu de **H1**. Parce que les espaces tels que les espaces, les tabulations, les retours à la ligne, etc. sont des caractères valides et ils forment des nœuds **#text** et deviennent une partie de l'arborescence DOM. Par conséquent, puisque la balise <**div**> contient une nouvelle ligne avant la balise <**h1**>, elle créera donc un nœud **#text**.

Pour éviter le problème avec **firstChild** et **lastChild** renvoyant des nœuds **#text** ou **#comment**, vous pouvez également utiliser les propriétés **firstElementChild** et **lastElementChild** pour renvoyer uniquement le premier et le dernier nœud d'élément, respectivement. Mais cela ne fonctionnera pas dans IE 9 et les versions antérieures.

```
<div id="main">
<h1 id="title">My Heading</h1>
<p id="hint"><span>This is some text.</span></p>
</div>
<script>
var main = document.getElementById("main");
alert(main.firstElementChild.nodeName); // Outputs: H1
main.firstElementChild.style.color = "red";
var hint = document.getElementById("hint");
alert(hint.firstElementChild.nodeName); // Outputs: SPAN
hint.firstElementChild.style.color = "blue";
</script>
```

De même, vous pouvez utiliser la propriété **childNodes** pour accéder à tous les nœuds enfants d'un élément donné, où le premier nœud enfant reçoit l'index 0. Voici un exemple:

```
<div id="main">
<h1 id="title">My Heading</h1>
<p id="hint"><span>This is some text.</span></p>
</div>
<script>
var main = document.getElementById("main");
// First check that the element has child nodes
if(main.hasChildNodes()) {
      var nodes = main.childNodes;
      // Loop through node list and display node name
      for(var i = 0; i < nodes.length; i++) {
            alert(nodes[i].nodeName);
```

```
        }
}
</script>
```

Le **childNodes** renvoie tous les nœuds enfants, y compris les nœuds non élémentaires tels que les nœuds de texte et de commentaire. Pour obtenir une collection d'éléments uniquement, utilisez plutôt la propriété **children**.

```
<div id="main">
<h1 id="title">My Heading</h1>
<p id="hint"><span>This is some text.</span></p>
</div>
<script>
var main = document.getElementById("main");
// First check that the element has child nodes
if(main.hasChildNodes()) {
        var nodes = main.children;
        // Loop through node list and display node name
        for(var i = 0; i < nodes.length; i++) {
                alert(nodes[i].nodeName);
        }
}
</script>
```

6.2. Accès aux nœuds parents

Vous pouvez utiliser la propriété **parentNode** pour accéder au parent du nœud spécifié dans l'arborescence DOM. Le **parentNode** retournera toujours **null** pour le nœud de document, car il n'a pas de parent.

```
<div id="main">
<h1 id="title">My Heading</h1>
<p id="hint"><span>This is some text.</span></p>
</div>
<script>
var hint = document.getElementById("hint");
alert(hint.parentNode.nodeName); // Outputs: DIV
alert(document.documentElement.parentNode.nodeName);    //    Outputs:
#document
alert(document.parentNode); // Outputs: null
</script>
```

Les nœuds de l'arborescence DOM les plus élevés sont accessibles directement en tant que propriétés du document. Par exemple, l'élément <**html**> est accessible avec la propriété **document.documentElement**, tandis que l'élément <**head**> est accessible avec la propriété **document.head** et l'élément <**body**> est accessible avec la propriété **document.body**.

Cependant, si vous souhaitez obtenir uniquement les nœuds d'élément, vous pouvez utiliser **parentElement**, comme ceci:

```
<div id="main">
<h1 id="title">My Heading</h1>
<p id="hint"><span>This is some text.</span></p>
</div>
<script>
var hint = document.getElementById("hint");
alert(hint.parentNode.nodeName); // Outputs: DIV
hint.parentNode.style.backgroundColor = "yellow";
</script>
```

6.3. Accéder aux nœuds frères

Vous pouvez utiliser les propriétés **previousSibling** et **nextSibling** pour accéder respectivement au nœud précédent et suivant dans l'arborescence DOM. Voici un exemple:

```
<div id="main">
<h1 id="title">My Heading</h1>
<p id="hint"><span>This is some text.</span></p>
<hr>
</div>
<script>
var title = document.getElementById("title");
alert(title.previousSibling.nodeName); // Outputs: #text
var hint = document.getElementById("hint");
alert(hint.nextSibling.nodeName); // Outputs: HR
</script>
```

Vous pouvez également utiliser **previousElementSibling** et **nextElementSibling** pour obtenir l'élément frère précédent et suivant en ignorant tous les nœuds de texte d'espaces. Toutes ces propriétés retournent **null** s'il n'y a pas de tel frère. Voici un exemple:

```
<div id="main">
<h1 id="title">My Heading</h1>
<p id="hint"><span>This is some text.</span></p>
</div>
<script>
```

```
var hint = document.getElementById("hint");
alert(hint.previousElementSibling.nodeName); // Outputs: H1
alert(hint.previousElementSibling.textContent);    //    Outputs:    My Heading
var title = document.getElementById("title");
alert(title.nextElementSibling.nodeName); // Outputs: P
alert(title.nextElementSibling.textContent); // Outputs: This is some text.
</script>
```

6.4. Types de nœuds DOM

L'arbre DOM est composé de différents types de nœuds, tels que des éléments, du texte, des commentaires, etc. Chaque nœud a une propriété **nodeType** que vous pouvez utiliser pour déterminer le type de nœud avec lequel vous traitez. Le tableau suivant répertorie les types de nœuds les plus importants:

Constant	Value	Description
ELEMENT_NODE	1	An element node such as <p> or <img>.
TEXT_NODE	3	The actual text of element.
COMMENT_NODE	8	A comment node i.e. <!-- some comment -->
DOCUMENT_NODE	9	A document node i.e. the parent of <html> element.
DOCUMENT_TYPE_NODE	10	A document type node e.g. <!DOCTYPE html> for HTML5 documents.

Référence:

1. Apprendre à développer avec JavaScript (3e édition) (Français) Broché – 12 décembre 2018 de Christian Vigouroux.
2. Tout JavaScript - 2e éd. (Français) Broché – 21 octobre 2020 de Olivier Hondermarck.
3. https://www.tutorialrepublic.com/html-tutorial/
4. https://developer.mozilla.org/fr/docs/Web/JavaScript
5. https://www.udemy.com/course/cours-javascript/