JavaScript
DOM*

IFNTI Sokodé – L3

28 Avril 2021

> Le terme DOM est un terme standardisé représentant une interface de programmation pour des documents HTML ou XML. Cette interface représente le document sous une forme d'un arbre. Cet arbre possède des noeud qui sont les éléments eux même. Cela permet au JavaScript d'y accéder et d'en manipuler le contenu et les attributs

Le DOM est égaleùent un ensemble d'API qui vont permettre de manipuler le `document`. Le `document` est une propriété de l'objet `window` que nous avons vu au cours précédent.

1 Présentation rapide

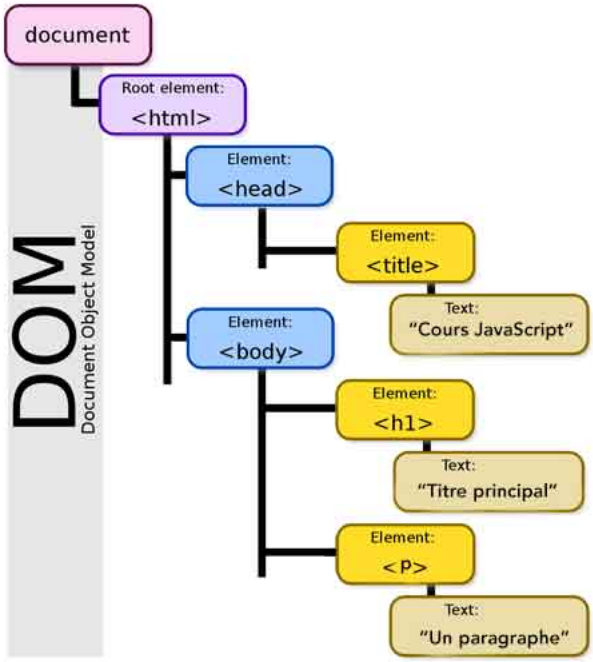
Le DOM represente la page HTML comme un arbre où chaque noeud (Node) est un élément HTML qui peut alors être accéder en tant qu'objet dont on va pouvoir manipuler les attributs.

Exemple tiré du cours de Pierre Giraud :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours JavaScript</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Titre Principal</h1>
    <p>Un paragraphe</p>
  </body>
</html>
```

Le navigateur va générer l'arbre suivant :

*Fortement inspiré de `https://openclassrooms.com/fr/courses/5543061-ecrivez-du-javascript-pour-le-web/`



Nous retrouvons le noeud `html` qui contient deux branches qui débouchent sur les noeuds `head` et `body` qui eux mêmes contiennent ...

Si vous inspectez les éléments d'une page web grâce aux outils du développeur contenu dans vos navigateurs, vous retrouverez cet arbre.

2 Accédez aux éléments du DOM

Tout commence avec le `document`. Cet objet est en réalité un élément de l'objet `window` qui va être appelé implicitement.

C'est donc lui qui contient les fonctions dont vous aurez besoin pour retrouver les éléments que vous cherchez.

Nous allons voir ensemble les principales fonctions de recherche d'éléments du DOM.

2.1 document.getElementById()

C'est sûrement la méthode la plus utilisée pour retrouver un élément, car c'est aussi la seule qui nous permette de retrouver facilement un élément précis.

Comme son nom l'indique, elle va rechercher un élément grâce à son id . Rappelez-vous qu'il ne doit y avoir qu'un seul élément avec un id donné, cette méthode est donc une candidate parfaite pour retrouver un élément particulier.

`getElementById(<id>)` prend en paramètre l'`id` de l'élément que vous recherchez et vous retournera cet élément s'il a été trouvé.

Par exemple, si l'on part du code HTML suivant :

```
<p id="my-anchor">My content</p> ,
```

on pourra trouver cet élément avec le code JavaScript suivant :

```
const myAnchor = document.getElementById('my-anchor');
```

2.2 document.getElementsByClassName()

Cette méthode fonctionne de la même manière que la précédente, mais fera sa recherche sur la class des éléments et retournera la liste des éléments qui correspondent.

Pour un rappel de ce qu'est une classe, vous pouvez retourner voir le cours Apprenez à créer votre site web avec HTML5 et CSS3.

`getElementsByClassName(<classe>)` prend en paramètre la classe des éléments à rechercher et vous retournera une liste d'éléments correspondants.

Par exemple, si l'on part du code HTML suivant :

```
<div>
    <div class="content">Contenu 1</div>
    <div class="content">Contenu 2</div>
    <div class="content">Contenu 3</div>
</div>
```

on pourra retrouver la liste des éléments ayant la classe content avec le code JavaScript suivant :

```
const contents = document.getElementsByClassName('content');
const firstContent = contents[0];
```

2.3 document.getElementsByTagName()

Avec cette méthode, vous rechercherez tous les éléments avec un nom de balise bien précis (par exemple tous les liens (a), tous les boutons (button)...).

De la même manière que la méthode précédente, vous récupérerez la liste des éléments correspondants.

`getElementsByTagName(<name>)` prend en paramètre le nom de la balise à rechercher et vous retournera la liste des éléments correspondants.

Si l'on part du code HTML suivant :

```
<div>
    <article>Contenu 1</article>
    <article>Contenu 2</article>
    <article>Contenu 3</article>
</div>
```

on pourra retrouver la liste des éléments de type article avec le code JavaScript suivant :

```
const articles = document.getElementsByTagName('article');
const thirdArticle = articles[2];
```

2.4 document.querySelector()

Cette méthode est plus complexe, mais aussi beaucoup plus puissante car elle vous permet de faire une recherche complexe dans le DOM, en mélangeant plusieurs procédés. Il s'agit en fait d'un sélecteur qui permet de cibler certains éléments.

Par exemple, `document.querySelector("#myId p.article > a")` fera une recherche dans l'élément ayant pour id `#myId` , les éléments de type `<p>` qui ont pour classe `article` , afin de récupérer le lien (`<a>`) qui est un enfant direct (pas des enfants de ses enfants).

Ainsi, avec le code HTML suivant :

```
<div id="myId">
    <p>
        <span><a href="#">Lien 1</a></span>
        <a href="#">Lien 2</a>
        <span><a href="#">Lien 3</a></span>
    </p>
    <p class="article">
        <span><a href="#">Lien 4</a></span>
        <span><a href="#">Lien 5</a></span>
        <a href="#">Lien 6</a>
    </p>
    <p>
        <a href="#">Lien 7</a>
        <span><a href="#">Lien 8</a></span>
        <span><a href="#">Lien 9</a></span>
    </p>
</div>
```

Que nous retournera la recherche JavaScript suivante :

```
const elt = document.querySelector("#myId p.article > a");
```

Les sélecteurs sont les sélecteurs CSS.

`querySelector()` ne renvoie pas une liste des résultats, mais le premier élément qui correspond à la recherche.

`querySelector(<selector>)` prend en paramètre le sélecteur et vous retournera le premier élément trouvé, ou null si aucun élément n'a été trouvé.

Pour retourner une liste de résultats qui correspondent à la recherche que vous souhaitez faire il faudra utiliser la fonction `querySelectorAll` , qui fonctionne de la même manière. :D

3 Les recherches depuis un élément

Il n'y a pas qu'avec `document` que vous pouvez rechercher des éléments. Chaque élément est un objet JavaScript avec ses propriétés et ses fonctions. Et parmi ces dernières, il en existe pour parcourir les enfants et le parent de chaque élément !

- `element.children` : cette propriété nous retourne la liste des enfants de cet élément ;
- `element.parentElement` : cette propriété nous retourne l'élément parent de celui-ci ;
- `element.nextElementSibling` / `element.previousElementSibling` : ces propriétés nous permettent de naviguer vers l'élément suivant / précédent de même niveau que notre élément.

Par exemple, avec le code HTML suivant :

```
<div id="parent">
    <div id="previous">Précédent</div>
    <div id="main">
        <p>Paragraphe 1</p>
        <p>Paragraphe 2</p>
    </div>
    <div id="next">Suivant</div>
</div>
```

et si l'on considère que nous avons le code JavaScript suivant :

```
const elt = document.getElementById('main');
```

qu'aurons nous ? :

- `elt.children`
- `elt.parentElement`
- `elt.nextElementSibling`
- `elt.previousElementSibling`