Bachelor CDWFS
Applications web dynamiques et interactives
2025-2026

Javascript

Cibler les éléments HTML

5 Méthodes

Méthode getElementById()

getElementById() prend en paramètre un id et retourne l'élément html ayant cet id.

Étant donné que les id d'éléments doivent être uniques, ils constituent un moyen utile d'accéder rapidement à un élément spécifique.

Par exemple :

```
<body>
      <h1 id='titre'>Titre principal</h1>
      <p>Un paragraphe</p>
 </body>
```

On pourra retrouver l'élément ayant l'id titre avec le code JavaScript suivant :

```
const titre = document.getElementById('titre') ;
```

Méthode getElementsByClassName()

La méthode getElementsByClassName() prend en paramètre une classe et retourne une liste d'éléments ayant cette classe.

Par exemple :

```
<h1 id='titre'>Titre principal</h1>
<p class='text'>Un paragraphe</p>
<p class='text'>Un paragraphe</p>
```

On pourra retrouver la liste des éléments de classe « text » avec le code JavaScript suivant :

```
const paragraphes = document.getElementsByClassName('text');
const paragraphe = paragraphes[1];
```

À l'instar de la méthode document.getElementsByClassName() qui effectue une recherche sur le document entier ; la methode **element**.getElementsByClassName() effectue une recherche **sur les descendants de l'élément spécifié.**

Exemple :

```
<p class='text'>Un paragraphe</p>
<div id='header'>
    <p class='text'>Un paragraphe</p>
    <p class='text'>Un paragraphe</p>
</div>
```

Pour récupérer les paragraphes dans le div#head :

```
const head = document.getElementById('titre') ;
const paragraphes = head.getElementsByClassName('text');
```

Méthode getElementsByTagName()

La méthode getElementsByTagName() prend en paramètre un nom de balise et retourne la liste des éléments html ayant cette balise.

Exemple :

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

On pourra retrouver la liste des balises « article » avec le code JavaScript suivant :

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

À l'instar de la méthode document.getElementsByTagName() qui effectue une recherche sur le document entier ; la methode **element**.getElementsByTagName () effectue une recherche **sur les descendants de l'élément spécifié**.

Méthode querySelector()

La méthode querySelector() prend en paramètre un sélecteur et retourne **le premier** élément qui correspond à la recherche. Elle ne renvoie **pas** une liste des résultats.

Par exemple :

```
<p class='text'>Paragraphe 1</p>
<div id='container' class='main'>
    <h1 id='titre'>Titre principal</h1>
     <p class='text'>Paragraphe 2</p>
     <p class='text'>Paragraphe 3</p>
</div>
<p class='text'>Un paragraphe</p>
```

On peut retrouver le premier élément correspondant au sélecteur « #container p.text » avec le code JavaScript suivant :

```
const paragraphe = document.querySelector('#container p.text') ;
console.log(paragraphe.textContent) ; // Affiche Paragraphe 2 ;
```

À l'instar de la méthode document.querySelector() qui effectue une recherche sur le document entier ; la méthode **element**.querySelector() effectue une recherche **sur les descendants de l'élément spécifié**.

Méthode querySelectorAll()

Pour retourner **une liste de résultats** qui correspondent à la recherche que vous souhaitez faire il faut utiliser la fonction querySelectorAll(), qui fonctionne de la même manière

On peut retrouver tous les éléments correspondant au sélecteur « #container p.text » avec le code JavaScript suivant :

```
const paragraphes = document.querySelectorAll('#container p.text') ;
```

À l'instar de la méthode document.querySelectorAll() qui effectue une recherche sur le document entier ; la méthode element. querySelectorAll() effectue une recherche **sur les descendants de l'élément spécifié**.

Exercice

```
<h1 id="titre">Gestion d'événements</h1>
<p id="p1" class="paragraphe" >
       <img id="img1" src="https://picsum.photos/300/230" alt="image1"/>
</p>
<p id="p2" class="paragraphe" >
       <img id="img2" src="https://picsum.photos/300/230" alt="image2"/>
       <button id="mask">Masquer l'image</button>
</p>
```

Ecrire les instructions suivante :

- Ecrire l'instruction qui enregistre dans une constante l'élément html ayant l'id p1.

- Ecrire l'instruction qui enregistre dans une constante les éléments html ayant la classe paragraphe.

- Ecrire l'instruction qui enregistre dans une constante les éléments html <img>.

- Ecrire l'instruction qui enregistre dans une constante les éléments html descendants directs de l'élément ayant l'id p1.

Parcours des listes d'éléments

Chaque liste récupérée par les méthodes getElementsByTagName(), getElementsByClassName() et querySelectorAll() sont des listes de « nœuds » (HTMLCollection ou NodeList).

Par exemple :

```
<div id="main">
    <p>Paragraphe 1</p>
    <p>Paragraphe 2</p>
</div>
```

Pour connaître le nombre d'éléments de cette liste, on utilise la propriété **length** :

```
const items = document.getElementsByTagName('p');
const nombre = liste.length ;
```

Pour parcourir tous les paragraphes de la liste, on peut utiliser **une boucle for** :

```
for (let i = 0; i < items.length; i++) {
  console.log(items[i]);
}
```

Autres syntaxes :

```
Array.from(items, item => {
    console.log(item)
}
```

```
Object.values(items).forEach(item => {
  console.log(item);
});
```

Pour aller plus loin

Noeuds adjacents, parents et enfants

Une page html est une arborescence d'éléments (body, div, p, a, span, etc.)

Lorsqu'on récupère un élément html dans une variable au moyen de l'une des méthodes citées ci-dessus, il est possible de récupérer les éléments adjacents, parents ou enfants.

On utilise les propriétés suivantes :

element.children

cette propriété retourne la liste des enfants de cet élément ;

element.parentElement

cette propriété retourne l'élément parent de celui-ci ;

element.nextElementSibling /element.previousElementSibling

ces propriétés permettent de naviguer vers l'élément suivant / précédent de même niveau que notre élément.

```
<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');
```

nous aurons ceci :

```
console.log(elt.children) ; // retourne les éléments de type  p  qui sont les enfants de
l'élément #main
console.log(elt.parentElement) ; // retourne la  div  qui a l'id  parent
elt.nextElementSibling retourne l'élément qui a l'id  next
elt.previousElementSibling retournera l'élément qui a l'id  previous
```

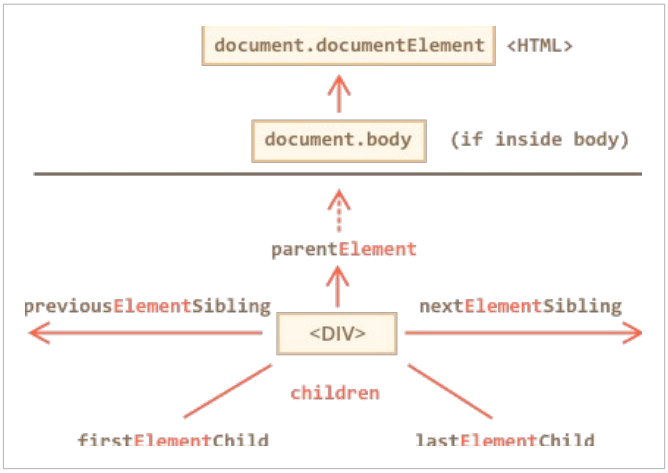


figure 2 : les différents nœuds

Un peu de vocabulaire …

Une page html a une structure **arborescente**. On trouve notamment les balises html, head, body. Dans la balise body, on trouve ensuite d'autres balises qui elles-mêmes contiennent une ou plusieurs autres balises.

Le Modèle Objet de Document (DOM: Document object Model) est la représentation de la structure et du contenu d'une page html sour fomre d' « **objets** »

Le terme « nœud » est un terme générique qui sert à désigner tous les **objets** contenus dans le DOM.

Il existe plusieurs types de nœud :

- **nœud « élément »**

  Ce sont les balises html.

  Ex : div, form, span, p, ...

- **nœud « attribut »**

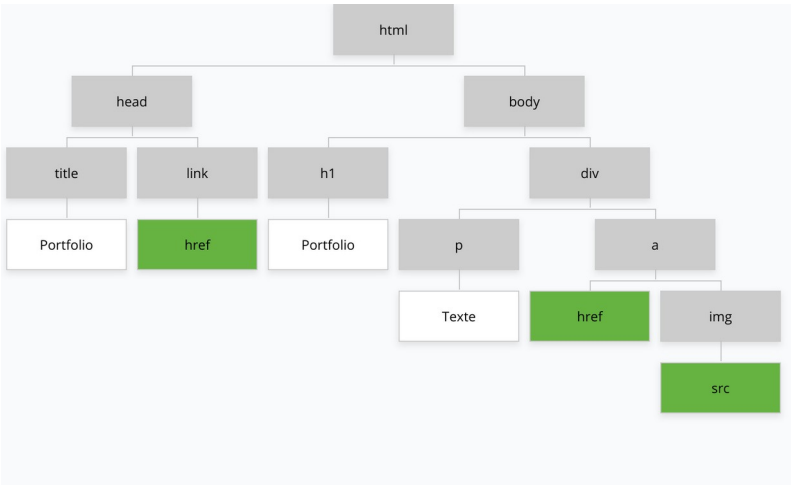
  Ce sont les attributs html.

  Ex : href, src, class, id, style, type, ...

- **nœud « texte »**

  Ce sont les contenus des éléments html

  Ex : le texte d'un paragraphe ou d'un span…



Arbre DOM réalisé avec https://www.gloomaps.com

Légende : En gris : les nœuds « éléments »
En vert : les nœuds « attribut »
En blanc : les nœuds « texte »

Les objets ont des « **attributs** » et des « **méthodes** »
La notion de « **méthode** » fréquemment utilisée est en quelque sorte une « fonction » appliquée à des « objets ».