Apprendre la JavaScript - Introduction au DOM

Le Document Object Model (DOM)

Le DOM est une représentation du document HTML qui peut être modifiée avec JavaScript. Cela signifie que vous pouvez ajouter, supprimer ou modifier des éléments HTML, des attributs, des classes, etc.

Il faut voir le DOM comme un arbre. Chaque élément HTML est un nœud dans cet arbre. Les nœuds peuvent être des éléments, des attributs, du texte, etc.

La racine de cet arbre est l'objet `document`. C'est à partir de cet objet que vous pouvez accéder à tous les autres éléments du document. Elle est l'équivalent de la balise `<html>` dans le document HTML.

https://bioub.github.io/dom-visualizer/

Meilleures pratiques

Il est important de ne pas abuser de la manipulation du DOM. Cela peut ralentir votre application. Il est préférable de manipuler le DOM le moins possible.

Il est donc préférable d'enregistrer les éléments fréquemment utilisés du DOM dans des variables globales pour éviter de les rechercher à chaque fois que vous en avez besoin.

Rechercher dans le DOM

Traverser le DOM depuis la racine vers les branches

querySelector

La méthode `querySelector` permet de rechercher un élément dans le DOM en utilisant un sélecteur CSS. Elle renvoie le premier élément qui correspond au sélecteur. Si aucun élément ne correspond, elle renvoie `null`.

```
let element = document.querySelector("h1");
let element = document.querySelector(".my-class");
let element = document.querySelector("#my-id");
let element = document.querySelector("input[data-attribute='value']");
```

querySelectorAll

La méthode `querySelectorAll` permet de rechercher tous les éléments dans le DOM qui correspondent à un sélecteur CSS. Elle renvoie une liste de tous les éléments qui correspondent au sélecteur. Si aucun élément ne correspond, elle renvoie une liste vide.

```
```

```
let elements = document.querySelectorAll("h1");
let elements = document.querySelectorAll(".my-class");
let elements = document.querySelectorAll("#my-id");
let elements = document.querySelectorAll("input[data-attribute='value']");
```

getElementById

Une autre méthode pour rechercher un élément dans le DOM est `getElementById`. Elle permet de rechercher un élément par son identifiant. Elle renvoie l'élément qui correspond à l'identifiant. Si aucun élément ne correspond, elle renvoie `null`.

Cette méthode est plus rapide que `querySelector` et `querySelectorAll` car les id sont uniques dans le document et sont indexés par le navigateur. Donc la liste est moins longue à parcourir.

À noter qu'il ne faut ma mettre le `#` devant l'identifiant.

```
let element = document.getElementById("my-id");
```

children, nextElementSibling, previousElementSibling

Ces propriétés permettent de rechercher les enfants, l'élément adjacent suivant et l'élément adjacent précédent d'un élément. Ces propriétés renvoient `null` s'il n'y a pas d'enfant, de frère suivant ou de frère précédent. Elles sont disponibles sur tous les éléments du DOM.

```
let children = element.children;
let firstChild = element.firstElementChild;
let lastChild = element.lastElementChild;
let nextSibling = element.nextElementSibling;
let previousSibling = element.previousElementSibling;
```

Traverser le DOM depuis les racines vers la racine

parentNode

La propriété `parentElement` permet de rechercher l'élément parent d'un élément. Elle renvoie `null` si l'élément n'a pas de parent.

```
let parent = element.parentElement;
```

closest

Pour chercher plus haut dans l'arbre, on peut utiliser la méthode `closest`. Elle permet de rechercher l'élément parent le plus proche qui correspond à un sélecteur CSS. Elle renvoie `null` si aucun élément ne correspond. C'est un peu l'inverse de `querySelector`.

```
let parent = element.closest(".my-class"); // Cherche l'élément parent le plus
proche qui a la classe "my-class"
```

Optimiser les recherches

Acceder à certains éléments directement dans l'objet Document

Il est possible d'accéder directement à certains éléments du DOM à partir de l'objet `document`. Par exemple, il est possible d'accéder directement à la balise `<body>` avec `document.body`. Ceci est plus efficace que d'utiliser `document.querySelector("body")` car le navigateur a déjà une référence directe à cet élément.

- `document.head`
- `document.forms` // Sert pour valider les formulaires
- `document.images`
- `document.links` // Sert pour tester si les liens sont valides par exemples

Rechercher dans les enfants d'un élément au lieu de la racine

Au lieu d'effectuer la méthode `querySelector` sur l'objet `document`, il est possible de l'effectuer sur un élément spécifique.

Par exemple, si vous voulez rechercher un élément dans un formulaire, vous pouvez effectuer la méthode `querySelector` sur l'objet `form` au lieu de l'objet `document`. Vous chercherez alors dans les enfants de l'objet `form` au lieu de chercher dans tout le document. Cela rendra la recherche plus rapide.

```
let form = document.querySelector("form");
let input = form.querySelector("input");
```

Modifier le DOM

Créer ou cloner un élément

innerHTML

La propriété `innerHTML` permet de créer ou de remplacer le contenu HTML d'un élément. Elle est très puissante, mais elle est aussi très dangereuse car elle peut supprimer tout le contenu HTML d'un élément. Il est donc préférable de l'utiliser avec précaution.

À noter que `innerHTML` retire également les écouteurs d'événements et les données associées à l'élément car il supprime et recrée l'élément.

```
element.innerHTML = "<h2>Hello, world!</h2>";
```

insertAdjacentHTML

La méthode `insertAdjacentHTML` permet d'insérer du contenu HTML à un endroit spécifique dans un élément. Elle est plus sûre que `innerHTML` car elle ne supprime pas le contenu HTML existant, elle l'ajoute simplement. On utilise cette méthode pour ajouter du contenu HTML depuis une chaîne de caractères dans le code JavaScript.

La méthode prend deux arguments. Le premier argument est une chaîne de caractères qui spécifie où insérer le contenu.

Les valeurs possibles sont `"beforebegin"`, `"afterbegin"`, `"beforeend"` et `"afterend"`. Le deuxième argument est une chaîne de caractères qui spécifie le contenu HTML à insérer.

```
let name = "Malek";
let elementAjout = `
<div class="my-class">
<span>${name}</span>
</div>
`;

element.insertAdjacentHTML("beforeend", elementAjout);
```

clodeNode

La méthode `cloneNode` permet de cloner un élément. Elle prend un argument booléen qui spécifie si les enfants de l'élément doivent également être clonés. Si l'argument est `true`, les enfants sont également clonés. Si l'argument est `false`, seuls l'élément et ses attributs sont clonés.

Cloner implique plus de travail mais permet de séparer le code HTML du JavaScript. La balise servant à cloner doit être cachée dans le HTML.

Pour modifier les attributs ou ajouter des écoutants d'événements, il faut d'abord cloner l'élément, l'ajouter au DOM, puis modifier les attributs ou ajouter les écouteurs d'événements en récupérant le dernier élément ajouté.

Pour ajouter un élément cloné à un élément parent, il faut utiliser la méthode `append`.

```
<div class="gabarit">
    <h1>Titre</h1>
    <h2>Sous-titre</h2>
```

```
</div>
<div class="conteneur"></div>
```

```
let gabarit = document.querySelector(".gabarit");
let conteneur = document.querySelector(".conteneur");

// Cloner le gabarit, true pour cloner les enfants aussi
let clone = gabarit.cloneNode(true);

// Modifier le titre et le sous-titre du clone
let titre = clone.querySelector("h1");
titre.textContent = "Nouveau titre";

let sousTitre = clone.querySelector("h2");
sousTitre.textContent = "Nouveau sous-titre";

// On peut ajouter un écouteur d'événement sur le clone
clone.addEventListener("click", function () {
    console.log("Cliqué");
});

// Ajoute le clone à la fin du conteneur, on utilise prepend pour ajouter au début
// Obligatoire sinon le clone reste dans le vide
conteneur.append(clone);
```

balise html template

La balise `<template>` permet de cacher du code HTML dans le document. Elle n'est pas affichée dans le document, mais elle est disponible pour le JavaScript. Cela permet de cloner du code HTML sans l'afficher dans le document.

Lorsqu'on sélectionne un élément avec `document.querySelector`, il faut utiliser `content` pour accéder au contenu de la balise `<template>`. Il est ensuite possible de cloner le contenu de la balise `<template>` avec `cloneNode`.

```
<template id="my-template">
    <h1>Hello, world!</h1>
</template>
```

```
let conteneur = document.querySelector(".conteneur");
let template = document.getElementById("my-template");
```

```
let contenu = template.content;
let clone = contenu.cloneNode(true);
conteneur.append(clone);
```

Supprimer un élément HTML

remove

La méthode `remove` permet de supprimer un élément du DOM. Elle est très simple à utiliser. L'élément est supprimé du DOM et de la mémoire. Il est donc impossible de le récupérer. Les écouteurs d'événements et les données associées à l'élément sont également supprimés.

```
let element = document.querySelector(".my-class");
element.remove();
```

Il est possible de supprimer tous les enfants d'un élément en utilisant la propriété `children` et une boucle `for`.

```
let enfants = document.querySelectorAll(".my-class > *");

enfants.forEach(function (enfant) {
    enfant.remove();
});

// OU
let element = document.querySelector(".my-class");
let enfants = element.children;

while (enfants.length > 0) {
    element.lastElementChild.remove();
}
```

supprimer les enfants avec innerHTML

Il est possible de supprimer un élément en utilisant la propriété `innerHTML`. Il suffit de définir `innerHTML` sur une chaîne de caractères vide. Cela supprime tout le contenu HTML de l'élément.

```
let element = document.querySelector(".my-class");
element.innerHTML = "";
```