JavaScript
Manipulation du DOM

Par Paul Bourgeois

Récupérer des éléments du DOM

Le DOM (Document Object Model) est une représentation du HTML comme un arbre d'élément. Chaque balise HTML est traitée comme un objet JavaScript. L'objet « document » représente la page HTML alors que l'objet « window » représente la fenêtre de votre navigateur.

Il est possible de sélectionner les éléments du DOM en passant par leur ID, leur class, leur balise, via une requête de sélection CSS ou encore via d'autres éléments proches :

Sélection par ID :

```
let elementAvecID = document.getElementByID(''monID'');
```

Cela permet de sélectionner dans le document HTML un élément avec l'ID précisé.
Exemple ici ce pourrait être <p id= ''monID''> </p>

Sélection par class :

```
let elementsAvecClasse = document.getElementsByClassName(''maClasse'');
```

Cela permet de sélectionner les éléments du documents HTML qui ont une classe maClasse. Tous ces éléments sont stockés dans un tableau

Sélection par balise :

```
let elementsAvecBaliseDeTypeDiv =
document.getElementsByTagName(''nomDeBalise'');
```

On sélectionne ici les éléments de type <nomDeBalise> </nomDeBalise>

Sélecteur CSS :

```
let elementPrecisAvecCSS = document.querySelector(''.maClasse > div *'');
```

Document.querySelector retourne le premier élément trouvé qui correspond à la requête CSS spécifiée. Il est possible récupérer tous les éléments qui correspondent à la requête en utilisant document.querySelectorAll

Sélecteurs relatifs :

```
let elementsEnfant = element.getChildren; //met tous les éléments enfant dans un tableau
let elementParent = element.getParent;     // récupère l'élément parent
let elementFrereSuivant = element.nextElementSibling;    //élément suivant
let elementFrerePrecedant = element.previousElementSibling;   //élément précédant
```

Manipuler les éléments du DOM

C'est bien joli de sélectionner des éléments mais que peut-on en faire exactement ?

Manipulation du contenu HTML ou textuel d'un élément:

```
let texte = element.textContent ;
element.textContent = ''Je modifie le texte de mon élément.'' ;
```

Ici nous récupérons le texte de l'élément HTML element. On pourra peut-être en avoir besoin. Ensuite, on remplace le texte de cet élément par une phrase essentielle.

```
let contenuHTML = element.innerHTML ;
element.innerHTML = ''<header><h1>Je modifie le contenu de ma page HTML </h1>''
+ ''<p>J'y met un <strong>petit</strong> paragraphe <em>aussi</em>.</p></header>'' ;
```

Dans ce code, on récupère le contenu HTML de l'élément element. Ensuite on remplace son contenu par un header avec un titre et un paragraphe avec du texte.

Manipulation des classes d'un élément :

```
const listeDesClasses = element.classList;  //renvoie a collection des classes de l'élément
element.classList.add(''maNouvelleClasse'');    //ajoute une nouvelle classe à l'élément
element.classList.remove(''classeASupprimer'');    //supprimer une classe de l'élément
listeDesClasses.contains(''cetteClasse'');          //vérifie si l'élément contient la classe cetteClasse
listeDesClasses.replace(''ancienneClasse'', ''nouvelleClasseRemplacante'');   //remplace une classe
```

Créer, ajouter et supprimer un élément :

Il est possible de créer de nouveaux éléments avec document.createElement(''typeDeBalise''). Cependant, cet élément ne sera pas directement ajouté dans le document. Il faudra l'ajouter à un élément parent ou remplacer un élément déjà existant dans l'élément parent.

```
let divElement = document.createElement(''div'') ;
elementParent.appendChild(divElement) ;
elementParent.replaceChild(divElement, document.createElement(''article'') ;
```

```
elementParent.removeChild(elementParent.children[0]) ;
```

Comme la dernière ligne l'indique, il est possible de supprimer un élément d'un élément parent.

Pour le code HTML suivant :

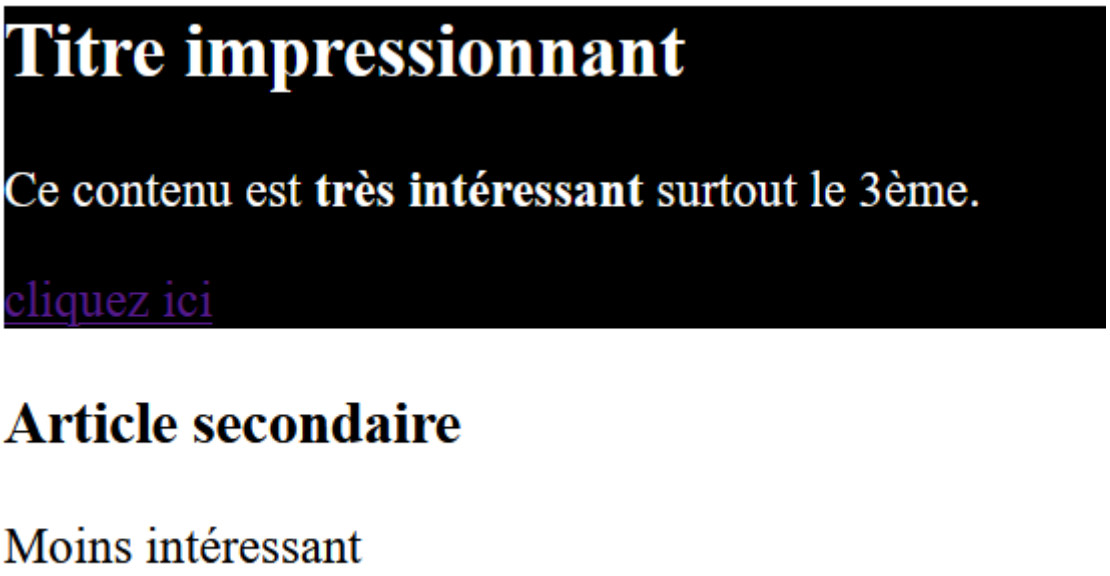
```html
<body>
    <main>
        <article class="article-principal">
            <h1>Titre impressionnant</h1>
            <p>Ce contenu est <strong>très
            intéressant</strong> surtout le 3ème.</p>
            <a href="#">cliquez ici</a>
        </article>
        <article>
            <h3>Article secondaire</h3>
            <p>Moins intéressant</p>
        </article>
    </main>
    <!--<script src="./testJS.js"></script>-->
</body>
```

Avec le CSS suivant :

```css
.article-principal {
    background-color: black;
    color: white;
}

.article-secondaire {
    color: blue;
}
```

Notre page resssemble à ceci :



En appliquant le JavaScript suivant :

```
let mainArticleClass = "article-principal";

let mainArticle = document.getElementsByClassName(mainArticleClass)[0];
if (!mainArticle.classList.contains("flex-article")) {
    mainArticle.classList.add("flex-article");
}
mainArticle.classList.replace(mainArticleClass, "article-secondaire");
mainArticle.textContent = "plus du tout intéressant";

let nextElement = mainArticle.nextElementSibling;
if(!nextElement.classList.contains(mainArticleClass)) {
    nextElement.classList.add(mainArticleClass);
}

let h1Article = document.createElement("h1");
h1Article.innerText = "Article principal !";
nextElement.replaceChild(h1Article, nextElement.children[0]);
```

On change l'article principal en article secondaire et l'article suivant devient l'article principal avec un nouveau titre :

plus du tout intéressant

Article principal !

Moins intéressant

La gestion des événements

Un événement permet d'ajouter une réaction (changement de valeur, d'attribut, de couleur, …) à une action (un clic, un mouvement de souris, écriture de texte, …). Ces événements sont propagé de l'élément le plus contenu au plus contenant : du lié cliqué à la div qui le contient à l'article qui contient la div à la balise main, … L'objet « event » contient les informations liées à l'événement.

Agir lors d'un événement :

Il existe trois manière d'utiliser des événements web. La première est d'utiliser les propriétés du gestionnaire d'événement. Ce sont des propriétés qui existent dans les objets de type Element que vous manipulez en JavaScript. Par exemple la propriété *onclick* permet d'agir lorsque vous cliquerez sur votre élément :

```
let monLien= document.getElementByID(‘’unLien’’) ;
monLien.onclick = function () {
    let couleurJaune =  ‘’rgb(255, 255, 0) ;
    document.body.style.backgroundColor = couleurJaune ;
}
```

Avec ce code, en cliquant sur le lien avec l'ID unLien, la couleur de fond de notre page passe en jaune.
Il existe de nombreuses propriétés de ce type qui sont listées ici, leur noms sont sous la forme on+nom de l'événement géré.

Une autre manière pour gérer les événements est l'utilisation d'EventListener (traduction littérale : écouteur d'événement). En utilisant la fonction addEventListener, on ajoute un EvenListener à nos éléments :

```
monElement.addEventListener(‘’click’’, maFonction) ;
```

Ici, maFonction est une fonction qui sera lancée lorsque l'événement de click sera reçu sur monElement.

L'avantage de cette technique est de pouvoir ajouter plusieurs actions différentes sur un même élément et sur un même événement, ce qui n'es p as possible autrement :

```
monLien.addEvenListener(‘’click’’, changerBackGround) ;
monLien.addEventListener(‘’click’’, lancerAlerte) ;
```

Il existe de nombreux événement. Vous pourrez trouver UNE LISTE COMPLETE ICI
Les plus fréquents sont :

- click : clic de la souris
- change : la valeur d'un champ a changé au moment de la perte de focus
- mousemove : la souris se déplace sur un élément

- mouseover : la souris entre dans un élément ou un de ses enfants

Enfin, il est également de gérer des événement directement dans les balises HTML :

```
<button onclick=''afficherDate()''>Essayez-moi</button>
```

Cette tehcnique est à proscrire. Aujourd'hui on évite d'écrire nos script directement dans nos pages HTML.

La propagation d'événement et le comportement par défaut :

De base, un événement lancé sur un élément est immédiatement propagé sur ses ancêtres (élément parent et plus anciens). Cela peut parfois créer des conflits, il faut donc stopper cette propagation à l'aide de la méthode stopPropagation().
De plus, vous voulez peut-être empêcher le fonctionnement classique de l'élément sur lequel a lieu votre événement. Par exemple, qu'un lien ne revoie nul part en cliquant dessus. Cela est également possible avec preventDefault() qui stoppera le fonctionnement par défaut de votre élément.

```
let monLien = document.getElementsByTagName(''a'')[0] ;
    monLien.addEventListener(''click'', function(event) {
    event.stopPropagation() ;
    event.preventDefault() ;
    alert(''Vous avez cliqué sur mon lien !'') ;
}) ;
```