JavaScript
DOM*

IFNTI Sokodé – L3

29 Avril 2021

1 Modifier le DOM

1.1 Modifions le contenu

Un élément possède 4 propriétés que nous pouvons consulter et/ou modifier :
— `innerHTML` permet de récupérer/modifier la syntaxe HTML interne d'un élément
— `outerHTML` permet de récupérer/modifier la syntaxe HTML complète d'un élément (y compris l'élément lui même)
— `textContent` représente le contenu texte du noeud
— `innerText` représente le contenu texte VISIBLE de l'élément.

Observons Regarder le code content.html et content.js fourni. Regarder comment cela est rendu sur le navigateur.

1. remarquez la différence de rendu entre `innerHTML` et `textContent`
2. remarquez la redéfinition de tag par `outerHTML`
3. remarquez ensuite la différence entre `innerText` et `textContent`. Changer la propriété `visibility` du span en le faisant passer de `hidden` à `visible`. Que devrait-il se passer ? Que se passe-t-il ?

1.2 Créons de nouveaux éléments

La création d'un nouvel élément se fait avec `document.createElement(<tag>)` avec tag étant le tag de l'élément que l'on souhaite créer.

Attention cet élément existe mais il n'est pas ajouté au DOM donc il n'existe pas dans le document.

1.2.1 Ajoutons du contenu

Pour ajouter un élément à un noeud nous utilions la méthode `appendChild` qui s'appelle à partir du noeud dans lequel nous souhaitons ajouter l'élément et en passant ledit élément en paramètre de la méthode.

Example :

```
const newElt = document.createElement("div");
let elt = document.getElementById("main");

elt.appendChild(newElt);
```

Il existe un grand nombre de méthodes capables d'insérer du contenu avant, après, entre, ... la liste est longue, découvrez la au besoin.

Si l'élément ajoutez est un élément qui est déjà dans le DOM. Celui-là sera déplacé.

1.2.2 Supprimons/Remplaçons un élément

De la même manière nous pouvons supprimer (`removeChild(<element>)`) et remplacer (`replaceChild(<nouveau>,<anci`
des éléments.

Example :

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

```
const newElt = document.createElement("div");
let elt = document.getElementById("main");
elt.appendChild(newElt);

// Supprime l'élément newElt de l'élément elt
elt.removeChild(newElt);
// Remplace l'élément newElt par un nouvel élément de type article
elt.replaceChild(document.createElement("article"), newElt);
```

1.2.3 Cloner

Il est aussi possible de cloner un élément.

Cloner un élément se fait avec la méthode `cloneNode()` qui prend en paramètre un booléen. Ce booléen indique à la méthode si les enfant sont clonés (true) ou pas (false).

Example :

```
let p1 = document.getElementByid("p1");
let cloneP1 = p1.cloneNode(true);
```

1.3 Exerçons nous

Récupérez le document manip.html et créons un fichier manip.js pour faire quelques manipulations :

1. créons un paragraphe, ajoutons le à notre page
2. déplacer le paragraphe p2 en dehors du div
3. cloner le paragraphe p1

1.4 Modifions la classe

Nous pouvons accéder aux classes d'un élément via la propriété `classList`. Cette propriété est un tableau dans lequel nous pouvons :

— ajouter (add) des classes
— retirer (remove) des classes
— remplacer (replace(<old>, <new>)) une classe
— vérifier si la liste contient (contains) une classe.

1.5 Changer de style

Les éléments ont une propriété `style` qui permet de récupérer et de changer le style de l'élément.

`style` possède une propriété pour chaque style existant (backgroundColor, color, ...).

1.6 Modifions des attributs

Il est également possible d'ajouter des attributs avec la fonction `setAttribute(<name>;<value>)`. Cette fonction est couplée à `getAttribute` et `removeAttribute`

Example : avec un elt de type input.

```
// Change le type de l'input en un type password
elt.setAttribute("type", "password");

// Change le nom de l'input en my-password
elt.setAttribute("name", "my-password");

// Retourne my-password
elt.getAttribute("name");
```

1.7 Jouons avec les styles

Récupérez style.html, style.css et créer style.js

1. Créez un nouvel élément de type paragraphe p ;
2. Ajoutez votre nouvel élément dans l'élément ayant pour id main ;

3. Ajoutez ce contenu HTML dans l'élément que vous avez créé lors de la première tâche : Mon grand contenu ;
4. Ajoutez la classe important à l'élément que vous avez créé lors de la première tâche ;
5. Votre élément est maintenant rouge, mais on voudrait qu'il soit vert. Modifiez les styles de votre élément (en JavaScript) pour qu'il soit vert.

2 Les événements

Repris du cours de Pierre Giraud : `https://www.pierre-giraud.com/javascript-apprendre-coder-cours/addeventlistener-gestion-evenement/`

En JavaScript, un évènement est une action qui se produit et qui possède deux caractéristiques essentielles :
— C'est une action qu'on peut « écouter », c'est-à-dire une action qu'on peut détecter car le système va nous informer qu'elle se produit ;
— C'est une action à laquelle on peut « répondre », c'est-à-dire qu'on va pouvoir attacher un code à cette action qui va s'exécuter dès qu'elle va se produire.

Par exemple, on va pouvoir détecter le clic d'un utilisateur sur un bouton d'un document et afficher une boite de dialogue ou un texte suite à ce clic. On parlera donc « d'évènement clic ».

Il existe de nombreux évènements répertoriés en JavaScript (plus d'une centaine). Les évènements qui vont nous intéresser particulièrement sont les évènements liés au Web et donc au navigateur. Ces évènements peuvent être très différents les uns des autres :
— Le chargement du document est un évènement ;
— Un clic sur un bouton effectué par un utilisateur est un évènement ;
— Le survol d'un élément par la souris d'un utilisateur est un évènement ;
— etc

Nous n'allons bien évidemment pas passer en revue chaque évènement mais allons tout de même nous arrêter sur les plus courants.

2.1 les gestionnaires d'événement

Pour répondre à un événement nous avons deux actions à faire :
— écouter le déclenchement d'un événement
— exécuter du code en réponse au déclenchement de l'action
Il y a trois manière d'écouter le déclenchement d'un événement :

1. avec les attributs HTML
2. avec des propriétés JavaScripts
3. avec la méthode addEventListener().

La dernière méthode est considérée comme la plus performante et c'est celle que nous préconiserons. Néanmoins vous rencontrerez les autres. C'est pour cela que nous vous les présentons.

2.1.1 Attribut HTML

C'est une vielle méthode qui consiste à ajouter un attribut à l'élément que l'on souhaite. Ces événements sont précédé par « on »
— `onclick`
— `onmouseover`
— `onmouseout`
— etc...
Exemple :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cours JavaScript</title>
        <meta charset="utf-8">
        <link rel="stylesheet" href="cours.css">
        <script src='cours.js' async></script>
```

```
    </head>

    <body>
        <h1>Titre principal</h1>
        <p>Un premier paragraphe</p>
        <button onclick="alert('Bouton cliqué')">Cliquez moi !</button>
        <div onmouseover="this.style.backgroundColor='orange'"
             onmouseout="this.style.backgroundColor='white'">
            <p>Un paragraphe dans un div</p>
            <p>Un autre paragraphe dans le div</p>
        </div>
    </body>
</html>
```

Il s'agit d'une vielle façon de faire qui mélange le JavaScript au html... et rappelez vous ce n'est pas très efficace.

2.1.2 Propriétés JavaScript

Les éléments par héritage, possède des propriétés liés aux événements qui portent le même nom que les attributs HTML :

exemple :

```
//On sélectionne le premier button et le premier div du document
let b1 = document.querySelector('button');
let d1 = document.querySelector('div');

//On utilise les propriétés gestionnaires d'évènement avec nos éléments
b1.onclick = function(){alert('Bouton cliqué')};
d1.onmouseover = function(){this.style.backgroundColor ='orange'};
d1.onmouseout = function(){this.style.backgroundColor='white'};
```

Cette façon de faire à un défaut majeur : il est impossible d'avoir plusieurs réaction à une même action sur cette élément.

2.1.3 addEventListener

La dernière méthode, et de loin celle qui est recommandée, consiste à utiliser la méthode `addEventListener` de notre élément afin de lui accrocher une fonction de réponse à l'événement.

Exemple :

```
//On sélectionne le premier button et le premier div du document
let b1 = document.querySelector('button');
let d1 = document.querySelector('div');

//On utilise la méthode addEventListener pour gérer des évènements
b1.addEventListener('click', function(){alert('Bouton cliqué')});
d1.addEventListener('mouseover', function(){this.style.backgroundColor ='orange'});
d1.addEventListener('mouseover', function(){this.style.fontWeight ='bold'});
d1.addEventListener('mouseout', function(){this.style.backgroundColor='white'});
```

Il est possible de supprimer un gestionnaire d'evenement avec `removeEventListener('evenement', nom_fonction)`. Cela suppose que nous avons donc donner un nom à la fonction.

2.2 Propagation d'événement

2.2.1 Petit exercice introductif

Récupérez le html et css intitulé propagation et créer un fichier js du même nom.

Nous souhaitons lancer une bataille de clics ! Nous avons un article qui correspond à notre parent (ID parent), et nous avons un lien (balise a) qui correspond à notre enfant (ID child). Le but est de cliquer soit dans le parent, soit dans l'enfant et d'afficher le nombre de clics dans chaque élément directement dans les éléments `#parent-count` et `#child-count`.

1. Commencez par écouter les événements click depuis l'élément `#parent`. Puis affichez le nombre de clics dans l'élément `#parent-count`.
2. Faites la même chose mais avec l'élément `#child`. Il faudra afficher le nombre de clics sur cet élément dans l'élément `#child-count`. Maintenant, dès que vous cliquez sur le parent ou l'enfant les compteurs se mettent à jour.

Mais vous avez sans doute remarqué que lorsque vous cliquez sur l'enfant, le compteur du parent se met aussi à jour ?

2.2.2 Retour à la théorie

Lorsqu'un évènement se déclenche, celui-ci va en fait naviguer à travers le DOM et passer à travers les différents gestionnaires d'évènement disposés dans le document. On dit également que l'évènement se « propage » dans le DOM.

ici, vous devez bien comprendre qu'un évènement (représenté en JavaScript par un objet) va toujours suivre le même chemin en JavaScript : il va toujours se propager en partant de l'ancêtre le plus lointain par rapport à la cible (généralement html) de l'évènement jusqu'à la cible de l'évènement puis faire le chemin inverse.

Dans notre cas l'événement `click` va suivre le chemin suivant : `html` → `body` → `article` → `a`.
Une fois arrivez à `a` il va faire le chemin inverse et remonter l'arborescence. `a` → `article` → `body` → `html`
Ce n'était pas toujours le cas mais aujourd'hui, les gestionnaires d'événement répondent pendant la phase de remontée.

Il est possible de changer cela : En réalité, la fonction addEventListener prend un troisième argument.
- de type booleén
  - true : la fonction est déclenchée pendant la phase de descente
  - false : la fonction est déclenchée pendant la phase de remontée
- par défaut vaut false

Comment l'événement se propage ? Il est littéralement passé à chacun des gestionnaire d'événement qui va le passer en paramètre de la fonction `callback`. Cette fonction va donc pouvoir le récupérer pour mieux le contrôler.

2.2.3 Comment arrêter la remontée ?

Il suffit d'utiliser la méthode `stopPropagation()` de l'événement.

Si plusieurs réaction sont attachées au même événement et que l'on veut se prémunir des autres réactions nou sutiliserons `stopImmediatePropagation`.

Exercice Et si on revenait à notre bataille de click ?

2.2.4 Comment éviter l'action par défaut

Souvent nous ne voulons pas que le comportement par défaut se déclenche. Exemple : envoie d'un formulaire, suivi d'un lien ....

pour cela nous allons utiliser la fonction `preventDefault()`

2.3 Récupérez des informations d'événement

Il est possible de récupérer un certains nombre d'informations lors du déclenchement d'un événement :
- récupérer les mouvements de la souris
- lire le contenu d'un champ texte

2.3.1 Mouvement de la souris

Afin de détecter le mouvement de la souris, il nous faut écouter l'événement mousemove. Cet événement nous fournit un objet de type MouseEvent . C'est-à-dire que dès que la souris bouge, notre fonction callback sera appelée avec un paramètre de type MouseEvent , qui contient les données sur le mouvement de la souris.
Voici, entre autres, ce que cet objet nous permet de récupérer :

— clientX / clientY : position de la souris dans les coordonnées locales (contenu du DOM) ;
— offsetX / offsetY : position de la souris par rapport à l'élément sur lequel on écoute l'événement ;
— pageX / pageY : position de la souris par rapport au document entier ;
— screenX / screenY : position de la souris par rapport à la fenêtre du navigateur ;
— movementX / movementY : position de la souris par rapport à la position de la souris lors du dernier événement mousemove

Voici un exemple illustrant tout ça :

```
elt.addEventListener('mousemove', function(event) {

    const x = event.offsetX; // Coordonnée X de la souris dans l'élément
    const y = event.offsetY; // Coordonnée Y de la souris dans l'élément

});
```

2.3.2 Contenu d'un champ texte

Il y a beaucoup d'événement javaScript (voir ici : https ://developer.mozilla.org/fr/docs/Web/Events). Prenons un petit exemple : Essayons de capturer le texte d'un champs quand l'utilisateur le modifie.

On voit qu'il y a un événement de type `change` Comment fonctionne-t-il ?

Exercice Récupérons gender.html et gendre.css et ajoutons notre fichier gendre.js

Nous avons ici un formulaire et nous aimerions restituer les réponses de notre formulaire dans l'élément du dessous. De plus, nous voudrions afficher la position de la souris quand elle se trouve dans cet élément.

1. Ecoutez les événements input sur l'élément `#name` afin de savoir quand le contenu du champ texte est changé. Affichez le contenu actuel dans l'élément `#res-name`
2. Maintenant nous voulons écouter l'événement du changement de choix du genre (`#gender`), et afficher le résultat dans l'élément `#res-gender`.
3. Nous souhaitons maintenant afficher les coordonnées de la souris à l'intérieur de l'élément `#result` dès que celle-ci passe par dessus. Ce que nous voulons, c'est avoir les coordonnées relatives au coin en haut à gauche de l'élément `#result`.