5: Manipuler le Web avec DOM

JavaScript et le DOM

Dans les chapitres précédents, nous avons exécuté des programmes JavaScript simples de manière interactive, depuis la console de notre navigateur.

Dans ce chapitre, nous allons voir:

- comment associer un programme JavaScript à une page web,
- et comment modifier cette page dynamiquement depuis notre programme.

Terminologie: quelques rappels sur le Web

- WWW: *World Wide Web*, c'est le nom donné à l'ensemble des pages liées entre elles sur Internet via le protocole HTTP (nous en reparlerons dans le chapitre suivant). La plupart de ces pages sont décrites en langage HTML.
- HTML: *HyperText Markup Language*, c'est un langage qui permet de décrire la structure et le contenu d'une page web, en utilisant des éléments (balises).
- CSS: *Cascading Style Sheets*, c'est un langage qui permet de mettre en page et styliser les éléments HTML d'une page web en leur appliquant des propriétés.
- JavaScript: c'est le seul langage qui permet de donner des instructions exécutables depuis une page web au format HTML.
- Navigateur Web / *Web Browser*: c'est un logiciel d'affichage de pages web (généralement composées de fichiers HTML, CSS et JavaScript), et permettant à l'utilisateur d'interagir avec celles-ci.
- DOM: *Document Object Model*, c'est à la fois le nom qu'on donne à l'ensemble des éléments qui constituent une page Web ouverte dans le navigateur (telle qu'elle est structurée en mémoire), et à l'API qui permet de manipuler ces éléments.
- API: *Application Programming Interface*, est un ensemble de fonctions fournies par un logiciel (par exemple: un navigateur Web, ou un serveur Web) qui permettent à d'autres programmes d'interagir / échanger des informations avec lui.

Associer un programme JavaScript à une page web

Vous devez savoir qu'une page web peut être associée à une feuille de style CSS. Pour cela, le code source HTML de cette page doit contenir un élément `<link>` donnant l'URL du fichier CSS correspondant.

De la même façon, pour associer un programme JavaScript à une page web, il suffit d'ajouter un élément `<script>` donnant l'URL du programme en question (dont le fichier porte généralement l'extension `.js`).

Exemple de code source HTML d'une page web:

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <h1>Bonjour !</h1>
    <script src="script.js"></script>
  </body>
</html>
```

Trois choses importantes à remarquer:

- l'élément `<script>` peut être défini dans le `<head>` ou dans le `<body>`, mais il est généralement recommandé de l'insérer juste avant la fin du `<body>`;
- l'URL du script doit être fournie via l'attribut `src`;
- mais surtout, contrairement aux éléments `<link>`, les éléments `<script>` ne doivent pas être exprimés sous forme d'une balise auto-fermante (finissant par `/>`) => il est impératif d'**utiliser une balise fermante `</script>` après chaque balise ouvrante `<script>`**.

Les scripts ainsi intégrés dans le `<body>` de la page seront exécutés au fur et à mesure qu'ils sont découverts et chargés par le navigateur.

> Remarque, il est aussi possible d'intégrer directement notre programme JavaScript entre les balises `<script>` et `</script>`, pour éviter de le stocker dans un fichier séparé. Cette méthode n'est pas recommandée car elle peut causer des erreurs de syntaxe.

Accéder aux éléments de la page Web depuis JavaScript

Les navigateurs Web (tels que Google Chrome) donnent accès à une *API* (voir définition plus haut) permettant à nos programmes JavaScript d'interagir avec le *DOM* de la page Web à laquelle ils sont liés.

C'est à dire qu'un script intégré dans une page peut utiliser des fonctions permettant de manipuler le contenu de cette page. Par exemple: pour récupérer des informations saisies par l'utilisateur dans des champs de la page, ou encore modifier le contenu et/ou l'affichage de la page.

Pour accéder à un élément de la page, il faut identifier précisément (c.a.d. sans ambiguïté) cet élément auprès du navigateur. Par exemple: en l'adressant par son identifiant unique (attribut `id` de l'élément).

Pour cela, l'API du DOM met à notre disposition un *objet* (cf chapitre précédent) appelé `document`, et cet objet contient plusieurs fonctions. Nous allons d'abord nous intéresser à la fonction `querySelector()` qui permet d'accéder à l'objet représentant un élément de la page, en fonction de sa classe, son id, etc.

Supposons que notre page Web contienne les éléments suivants:

```
<body>
  <p id="premier-paragraphe">Bonjour</p>
  <p class="deuxieme-paragraphe">le monde</p>
</body>
```

Nous pouvons alors accéder au premier paragraphe en JavaScript (ex: depuis un script rattaché à cette page, ou depuis la console du navigateur) de la manière suivante:

```
document.querySelector('#premier-paragraphe');
// => retourne un objet qui représente l'élément HTML correspondant

document.querySelector('.deuxieme-paragraphe');
// => retourne un objet qui représente l'élément HTML correspondant
```

En exécutant cet appel de fonction dans la console, on voit s'afficher ce qu'elle retourne: un objet JavaScript qui représente l'élément HTML ayant `premier-paragraphe` ou l'élément `deuxieme-paragraphe` comme identifiant.

Réagir aux actions de l'utilisateur sur la page

Maintenant que nous savons accéder aux données d'une page HTML depuis un programme JavaScript, et faire en sorte que ce programme s'exécute au chargement de la page, nous allons voir comment exécuter des instructions JavaScript en réponse à une action de l'utilisateur sur la page.

À chaque fois que l'utilisateur interagit avec une page Web, le navigateur déclenche des *événements*. Ces événements sont mis à disposition par l'API du DOM, afin qu'un programme JavaScript puisse les intercepter et réagir à certains d'entre eux.

Quelques exemples d'événements:

- `click`: l'utilisateur a cliqué sur un élément
- `change`: l'utilisateur a changé la valeur d'un champ de saisie
- `mouseover`: l'utilisateur a survolé un élément à l'aide de la souris

> Vous trouverez la liste des évènements standard sur cette page: Event reference - MDN.

On peut définir le comportement (la réaction) que doit adopter le navigateur lorsqu'un événement survient, en y associant une fonction JavaScript.

> Par exemple, nous pourrions définir une fonction `direBonjour()` contenant `alert('bonjour !')`, puis demander au navigateur d'appeler cette fonction à chaque fois que l'utilisateur clique sur un bouton.

Il existe plusieurs moyens d'intercepter des événements en y attachant une fonction:

- la fonction `addEventListener()` (que nous verrons peut-être plus tard)
- et les propriétés `on*` associées à chaque nom d'événement, ex: `onclick`, `onchange`, `onmouseover`...

Pour l'instant, nous allons employer la méthode la plus simple: affecter une fonction à la propriété d'un élément correspondante à l'événement choisi.

Imaginons une page HTML contenant un bouton:

```
<body>
  <button id="mon-bouton">Mon Beau Bouton</button>
</body>
```

Pour afficher un `alert` à chaque fois que l'utilisateur cliquera sur ce bouton (événement `click`), nous devons affecter une fonction à la propriété `onclick` de l'objet JavaScript représentant ce bouton:

```
document.querySelector('#mon-bouton').onclick = function direBonjour() {
  alert('bonjour !');
};
```

Une autre façon d'écrire ce même code est la suivante:

```
document.querySelector('#mon-bouton').addEventListener('click', function() {
  alert('bonjour !');
}, false);
```