TP 10: Manipulation DOM avancée

Objectifs:

- Inspecter la source d'un évènement du DOM
- Manipuler la structure du DOM en JavaScript

<!--
TODO

- Introduction aux templates
- Introduction à jQuery
  -->

> Slides du TP

Plan du cours:

1. Utilisation de la classe `Event`

- Exercice: détecter "Entrée" dans un champ texte

2. Navigation dans le DOM

- Exercice: changer la couleur du parent au clic

3. Modification de la structure du DOM

- Exercice: construire une page web en JavaScript
- Exercice: supprimer le champ de recherche de Google

Dans les chapitres précédents, nous avons:

- utilisé l'API fournie par le navigateur pour accéder aux éléments du DOM d'une page web/HTML,
- modifié les classes et le rendu d'éléments,
- défini une réaction aux évènements déclanchés par l'utilisateur,
- et introduit les concepts de Programmation Orientée Objet, et de classes.

Dans ce chapitre, nous allons voir:

- comment en savoir plus sur un évènement qui a été déclanché par l'utilisateur,
- comment naviguer dans le DOM d'une page web/HTML, de noeud en noeud,
- et comment en modifier la structure.

---

1. Utilisation de la classe `Event`

Reprenons un exemple de page web avec gestion d'évènement *click* sur un élément HTML:

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

Pour afficher un `alert` à chaque fois que l'utilisateur cliquera sur ce bouton, nous avons vu qu'il fallait affecter une fonction à la propriété `onclick` de l'objet JavaScript représentant ce bouton:

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

En effet, les propriétés d'évènements (`onclick`, `onchange`, ou autre) d'un élément permettent d'indiquer au navigateur quelle fonction appeler lorsque l'évènement correspondant est déclenché par l'utilisateur.

Ce que nous n'avons pas encore vu, en revanche, c'est que le navigateur passe systématiquement un paramètre lorsqu'il appelle cette fonction: une instance de la classe `Event`.

Affichons la valeur de ce paramètre `event` (objet-instance de la classe `Event`) dans la console, afin d'en découvrir les propriétés:

```
document.getElementById('mon-bouton').onclick = function(event) {
  console.log(event);
};
```

La propriété la plus couramment utilisée est `event.currentTarget`. En effet, elle a pour valeur l'élément HTML sur lequel l'évènement a été intercepté par notre fonction.

Cette propriété est particulièrement utile dans le cas où nous voulons affecter une même fonction de gestion d'évènement sur plusieurs éléments, mais que cette fonction a besoin de savoir sur lequel de ces éléments l'évènement a été déclenché.

Par exemple:

```
<button>Mon 1er Bouton</button>
<button>Mon 2ème Bouton</button>
<button>Mon 3ème Bouton</button>
```

```
var boutons = document.getElementsByTagName('button');
for (var i = 0; i < boutons.length; i++) {
  boutons[i].onclick = function(event) {
    alert('bouton clické: ' + event.currentTarget.innerHTML);
    // => le contenu du bouton clické va être affiché dans l'alert
  };
}
```

À noter que, selon le type d'évènement auquel est associée une fonction, le paramètre `event` peut être une instance de sous-classes contenant des propriétés et méthodes supplémentaires.

Par exemple:

- le paramètre `event` d'une fonction liée à l'évènement `mousemouse` (appelée à chaque mouvement de la souris) est une instance de la classe `MouseEvent` qui possède les propriétés `clientX` et `clientY`, afin de connaître la position de la souris sur l'écran.
- le paramètre `event` d'une fonction liée à l'évènement `keydown` (appelée quand l'utilisateur tape un caractère dans un champ) est une instance de la classe `KeyboardEvent` qui possède

les propriétés `key` et `keyCode`, permettant de connaître quelle touche a été tapée par l'utilisateur.

Par ailleurs, la classe `Event` et ses sous-classes définissent une méthode `preventDefault()` qui permet d'annuler le traitement habituel d'un évènement qui a été intercepté par une fonction. Cette méthode est par exemple utile pour empêcher l'ouverture d'un hyperlien `<a>`, la soumission d'un formulaire, ou l'ajout de caractères dans un champ.

Exercice: détecter "Entrée" dans un champ texte

Créer une page HTML simple contenant un champ `<input>` dont la valeur est effacée quand l'utilisateur presse la touche "Entrée".

<!--

> Solution: jsfiddle
> -->

---

2. Navigation dans le DOM

Toute page HTML est structurée sous forme hiérarchique (arbre): chaque élément HTML a un élément parent, et peut avoir plusieurs éléments enfants.

Reprenons par exemple notre page web ne contenant qu'un bouton:

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

Dans cette page, l'élément `<button>` a l'élément `<body>` comme *parent*, et un *noeud texte* (le contenu textuel de l'élément) comme *enfant*. On peut aussi dire que `<button>` est un *noeud enfant* de `<body>`.

Ici, le seul *enfant* de notre élément `<button>` est un noeud textuel, mais il serait possible que ce même élément ait d'autres éléments comme *enfants*.

Supposons que notre bouton soit référencé par la variable `element`:

```
var element = document.getElementById('mon-bouton');
```

Comme tout élément HTML représenté en JavaScript (et donc instance de la classe `Element`), notre variable `element` possède trois propriétés utiles pour nous aider à naviguer dans ses noeuds parent et enfants:

- `parentNode` est le noeud/élément HTML parent de l'élément,
- `children` est un tableau de d'éléments, enfants directs de l'élément, et
- `childNodes` est un tableau de noeuds HTML (éléments et/ou noeuds textuels), enfants directs de l'élément.

Dans notre exemple:

- `element.parentNode` est l'équivalent de `document.body` ; (l'élément `<body>` de notre page)
- `element.children` est un tableau vide, car il n'y a pas d'éléments à l'intérieur du bouton, et
- `element.childNodes` est un tableau qui ne contient que le noeud textuel ayant pour valeur "`Mon Beau Bouton`".

Exercice: Changer la couleur du parent au clic

Soit la page HTML suivante:

```
<div style="padding: 10px;">
  <button>Mon 1er Bouton</button>
</div>
<div style="padding: 10px;">
  <button>Mon 2ème Bouton</button>
</div>
<div style="padding: 10px;">
  <button>Mon 3ème Bouton</button>
</div>
```

Écrire le code JavaScript permettant de colorier le fond du `<div>` parent en jaune lorsqu'un bouton est cliqué par l'utilisateur.

Pour cela, utiliser une boucle `for`, une seule fonction `onclick`, son paramètre `event`, les propriétés `parentNode` et `style`.

<!--

> Solution: jsfiddle
-->

---

3. Modification de la structure du DOM

À ce stade, nous savons accéder à des éléments du DOM, modifier leur contenu et rendu, et accéder à leurs parent et enfants.

Dans cette partie, nous allons voir comment créer, rattacher et supprimer des noeuds du DOM.

Créer et rattacher un noeud dans le DOM

Pour ajouter un élément HTML dans le DOM d'une page web, le navigateur fournit les méthodes suivantes:

- `document.createElement(nom)` pour créer un élément,
- `document.createTextNode(texte)` pour créer un noeud textuel, et
- `conteneur.appendChild(element)` pour ajouter un élément comme enfant d'un autre élément.

Donc, pour ajouter un bouton au `<body>` d'une page HTML, il faudra exécuter les instructions JavaScript suivantes;

```
// 1. créer l'élément
var bouton = document.createElement('button');
// 2. créer le contenu du bouton (noeud textuel)
var texteDuBouton = document.createTextNode('Mon beau bouton');
// 3. ajouter le contenu au bouton
bouton.appendChild(texteDuBouton);
// 4. ajouter le bouton au body de la page
document.body.appendChild(bouton);
```

Supprimer un noeud du DOM

Le retrait d'un noeud du DOM est l'opération inverse de celle d'ajout (`appendChild()`): elle consiste à appeler la méthode `conteneur.removeChild(noeud)` , où `conteneur` est le noeud parent duquel on souhaîte retirer `noeud` .

Ainsi, pour supprimer le bouton que nous avons ajouté à la page de l'exemple précédent, il faudra exécuter:

```
document.body.removeChild(bouton);
```

... ou, de manière plus générale:

```
bouton.parentNode.removeChild(bouton);
```

Exercice: construire une page web en JavaScript

En partant d'une page HTML vierge, développer le code JavaScript permettant d'ajouter à la page:

- un champ `<input>` ,
- et un bouton qui vide la valeur du champ quand on clique dessus.

Pour cela, utiliser: `createElement()` , `createTextNode()` , `appendChild()` , `document.body` , `onclick` , et `value` .

<!--

> Solution: jsfiddle
> -->

Exercice: supprimer le champ de recherche de Google

En utilisant l'inspecteur et la console de Chrome Dev Tools depuis la page google.com, trouver l'instruction JavaScript permettant de supprimer le champ de recherche.

<!--

> Solution: jsfiddle
> -->