JDeDev

# Formation Javascript

## Partie 2 : Le DOM et la gestion des formulaire

En partenariat avec


**Julien Demettre EIRL ▸ Expert Windev Freelance**
+33 (0)6 49 24 41 62 ▸ julien@jdedev.fr ▸ www.jdedev.fr
Siège social ▸ 125 Rue de Viller, 54300 LUNEVILLE - FRANCE
SIRET ▸ 832 481 220 00024 | APE ▸ 5829C

JDeDev

# Sommaire

## PRÉSENTATION

Le DOM (Document Object Model) ou modèle objet de document est une API standard des navigateurs. Cette API permet de récupérer, modifier, ajouter ou supprimer un élément HTML (balise). Le dom est accessible grâce à l'objet **document** accessible partout dans du code navigateur.

Un code javascript client à pour but principal de manipuler le DOM.

À noter : l'objet document n'est pas accessible dans du code serveur

## RÉCUPÉRATION D'ÉLÉMENTS

Il existe plusieurs manières de récupérer des éléments du HTML, vous pouvez grâce a ces fonctions soit un élément soit une liste d'éléments qu'il vous faudra parcourir pour agir sur ces derniers. Un élément récupéré ou crée de toute pièce est un objet de type **Element.**

Si vous manipulez un élément récupéré dans le document, votre page se mettra automatiquement à jour car l'élément reste attaché à la page.

### Attraper un élément par son id :

```
const monId = document.getElementById('monId')
console.log(monId);
```

Ce code attrapera n'importe quel balise HTML qui possède un id ayant pour valeur **monId**

```
<div id="monId"></div>
```

### Attraper un élément par son sélecteur CSS

```
const monId = document.querySelector('#monId')
console.log(monId);
```

Ici on attrape le premier élément disponible dans la page correspondant au sélecteur CSS fournit **#monId**, si vous savez bien utiliser les sélecteurs CSS vous pourrez attraper n'importe quelle balise de votre code HTML

## Attraper une liste d'éléments par une classe

Attention ici nous n'auront pas un seul élément mais bien une liste qui peut contenir plusieurs éléments

```
const mesGris = document.getElementsByClassName('gris')
for (let unGris of mesGris) {
    console.log(unGris);
};
```

Dans mon cas je récupère ces deux éléments

```
<div class="gris"></div>
<p class="gris"></p>
```

## Attraper une liste d'éléments par son nom de balise

Nous aurons également à faire à une liste d'éléments

```
const mesDiv = document.getElementsByTagName('div')
for (let unDiv of mesDiv) {
    console.log(unDiv);
};
```

Dans mon cas je récupère ces deux éléments

```
<div class="gris"></div>
<div id="monId"></div>
```

## Attraper une liste d'éléments par son sélecteur CSS

Nous aurons également à faire à une liste d'éléments

```
const mesDiv = document.querySelectorAll('div')
for (let unDiv of mesDiv) {
    console.log(unDiv);
};
```

Dans mon cas je récupère ces deux éléments

```
<div class="gris"></div>
<div id="monId"></div>
```

## Créer un élément

L'API document vous permet également de créer un élément directement dans le code javascript, il vous faut juste le nom de la balise.

Attention un élément crée en javascript n'est pas encore attaché au document, il faudra l'ajouter dans votre page.

```
const maDiv = document.createElement('div')
```

## Modifier un élément

Un élément HTML qu'il soit récupéré ou créé peut être manipulé par javascript. Pour ce qui est des attributs de l'élément, vous pouvez y accéder de cette manière : élément.nomAttribut = valeur

```
const monImg = document.createElement('img')
monImg.src = "./logo.png"
```

Vous pouvez ainsi récupérer ou modifier la valeur d'un attribut de votre élément.

### Cas spécial des classes

Si vous souhaitez accéder aux classes attribuées à un élément vous pouvez le faire de deux manières :

#### Interagir directement avec la liste

```
monImg.classList.add("gris")
```

Vous pouvez utiliser les méthodes : add, remove et toggle

#### Interagir avec la chaine représentant la liste

```
monImg.className = "btn gris"
```

### Cas spécial des styles CSS

L'attribut style de l'élément HTML est lui-même un objet qui peu contenir toutes les propriétés CSS au format camel case

```
monImg.style.backgroundColor = "red"
```

## Gestion des évènements

Le but de javascript sur navigateur est de gérer les interactions avec l'utilisateur, pour le moment nous avons vu comment manipuler la page et ces éléments, nous allons voir comment programmer notre élément HTML pour qu'il réagisse aux actions de l'utilisateur (clic de souris, survol, appuis sur touche de clavier, drag and drop, etc…)

Pour gérer un évènement nous allons utiliser la méthode addEventListener

```
function handleClick(event) {
    alert('On a cliqué')
}
monImg.addEventListener("click",handleClick)
```

La méthode addEventListener prends deux paramètres : un nom d'évènement et un callback (fonction qui sera appelé automatiquement par le navigateur lorsque l'évènement se produit). Lorsque le navigateur lance la fonction callback il envoie un paramètre de type objet qui est une représentation de l'évènement (nom évènement, élément HTML source de l'évènement, éventuellement paramètre de l'évènement : touche clavier, position de la souris, etc…).

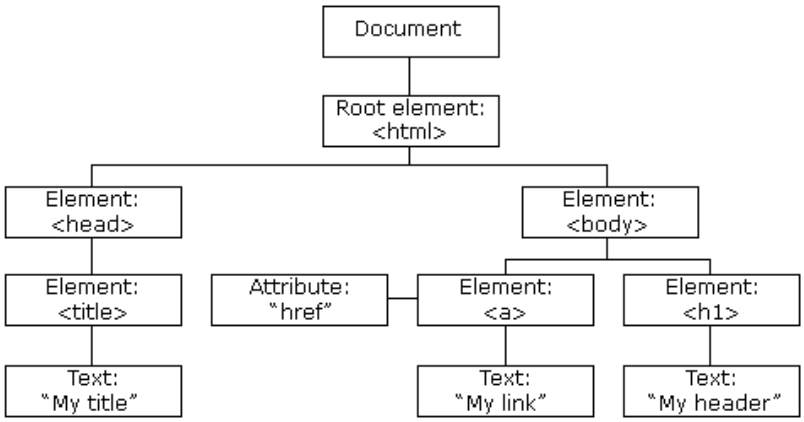
À noter : Javascript étant souple dans son typage, un développeur peut choisir de récupérer l'objet event envoyé par le navigateur ou pas

À noter 2 : n'oubliez pas qu'il y a plusieurs manières d'écrire une fonction

À noter 3 : attention addEventListener ajoute une nouvelle fonction à exécuter lors d'un évènement, ainsi vous pouvez avoir un souci d'un même évènement sur un élément qui déclenche plusieurs fois votre callback, cela peut être source de bug

## AJOUTER UN ÉLÉMENT

La structuration d'un document HTML peut être représenté sou forme d'un arbre avec un élément HTML d'entrée (la balise <html>) qui peut contenir plusieurs autres éléments HTML




Pour ajouter un nouvel élément à l'arbre, il vous faudra votre élément mais également l'élément parent sur lequel vous souhaitez l'accrocher

```
const container = document.getElementById('section1')
const monImg = document.createElement('img')
monImg.src = "./logo.png"
monImg.style.backgroundColor = "red"
function handleClick(event) {
    alert('On a cliqué')
}
monImg.addEventListener("click", handleClick)
container.append(monImg)
// ou
container.appendChild(monImg)
```

## Supprimer un élément

Pour supprimer un élément du document, vous devez l'attraper

```
const container = document.getElementById('section1')
container.remove()
```

Vous pouvez aussi récupérer un parent puis l'élément à retirer avant de supprimer (ancienne manière de faire)

```
const container = document.getElementById('section1')
const elem = document.querySelector('#section1>img')
container.removeChild(elem)
```

## Gestion des formulaires

Vous verrez globalement deux manières de contrôler un formulaire : avec ou sans l'API de validation HTML5

### Avec API

Cette méthode est plus moderne car elle utilise des attributs HTML (mais pourrais être considéré comme moins sécurisé)

Avec l'API fournit par les navigateurs et HTML5 vous pouvez définir un type précis pour un input, vous avez également des attributs required, min, max et pattern (pour saisir une expression régulière de validation)

L'avantage de cette méthode, l'utilisation en CSS des pseudos classes :valid et :invalid qui permettent directement un rendus de style automatisé sans avoir à écrire la moindre ligne de javascript.

En Javascript vous pouvez directement sur un élément input attrapé vérifié sa validité au format élément.validity.valid.

```
<body>
    <form id="form">
        <input type="email" name="mail" id="mail" required>
        <p id="mailErr"></p>
        <input id="valid" type="submit" value="Envoyer">
    </form>
    <style>
        #mail:invalid {
            background-color: rgba(255, 0, 0, 0.6);
            color: white;
        }
    </style>
    <script>
        const form = document.getElementById('form')
        const mail = document.getElementById('mail')
        const mailErr = document.getElementById('mailErr')
        const valid = document.getElementById('valid')

        valid.addEventListener("click", (event) => {
            event.preventDefault()
            if (!mail.validity.valid) {
                mailErr.style.color = "red"
                mailErr.style.fontWeight = "bold"
                mailErr.innerText = "Email non valide"
            } else {
                mailErr.innerText = ""
            }
        })
    </script>
</body>
```

## Sans l'API de validation

Si l'on ne souhaite pas utiliser l'API de validation, il faudra comparer la valeur du champ avec une Expression régulière

```
<body>
    <form id="form">
        <input type="email" name="mail" id="mail" required>
        <p id="mailErr"></p>
        <input id="valid" type="submit" value="Envoyer">
    </form>
    <script>
        const mail = document.getElementById('mail')
        const mailErr = document.getElementById('mailErr')
        const valid = document.getElementById('valid')

        valid.addEventListener("click", (event) => {
            event.preventDefault()
            const re = new RegExp(/^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$/, "g")
            if (!mail.value.match(re)) {
                mailErr.style.color = "red"
                mailErr.style.fontWeight = "bold"
                mailErr.innerText = "Email non valide"
            } else {
                mailErr.innerText = ""
            }
        })
    </script>
</body>
```

Il existe plusieurs manières de faire du stockage de données côté client : les cookies, le stockage local, le stockage de session, la base indexedDB

## Les cookies

La technologie des cookies est une technologie assez ancienne qui permet de partager des variables entre le back et le front, les deux ont accès aux cookies stockés chez le client. Se système fonctionne en automatique car les cookies sont transférés automatique à chaque requête et à chaque réponse (ce qui n'est pas super en termes de sécurité)

```
document.cookie = "cle=valeur; expires=datefin"
```

La manipulation des cookies en javascprit n'est pas très aisée du fait de la manipulation de cette chaine de caractères

## Le stockage local et de session

Les deux systèmes fonctionnent de la même manière mais n'ont pas le même but. Le stockage de session garde des variables dans le navigateur de manière temporaire. Le stockage de session est vidé lors de la fermeture du navigateur. Le stockage local est lui conservé de manière permanente. Les deux systèmes sont simple d'accès, ils fonctionnent sur la gestion d'une liste d'élément composé d'une clé et d'une valeur texte

```
localStorage.setItem("cle","valeur")
const val = localStorage.getItem("cle")
localStorage.removeItem("cle")
```

De cette manière vous pouvez créer, récupérer ou supprimer une variable du localStorage.

En remplaçant simplement localStorage par sessionStorage vous pourrez simplement changer de mode de stockage

## La base indexedDB

Cette méthode de stockage est plus complexe mais permet de stocker une grande quantité de données structurées. IndexedDB est une vraie base NoSql embarqué dans les navigateurs.

https://developer.chrome.com/docs/devtools/storage/indexeddb?utm_source=devtools&hl=fr

## ENTRAINEZ VOUS

Sur Une page web préparez un formulaire qui permettra d'ajouter une personne dans un tableau.

Pour créer une personne il vous faudra son nom, son prénom, son adresse email, sa rue, son code postal et sa ville.

Évidemment les champs du formulaire d'ajout seront contrôlés (code postal = 5 chiffres, nom prénom et ville doivent avoir au moins un caractère, email doit ressembler à un email)

En cas de mauvais remplissage du formulaire vous afficherez un message d'erreur en rouge en dessous de l'input

L'ensemble de vos personnes seront affichés sous forme de tableau. Pour chaque personne vous proposerez de modifier ou supprimer la personne

Vous stockerez le tableau de Personne dans une variable du stockage de session

Option : vous pouvez trier l'affichage du tableau par colonne croissante ou décroissante

Option 2 : vous pouvez filtrer les personnes en recherchant un nom et/ ou un prénom dans un champ de recherche