

JAVASCRIPT

Synthèse de cours

Mots clé : JavaScript, dom, événements, ...

- M.TOULOUSE -

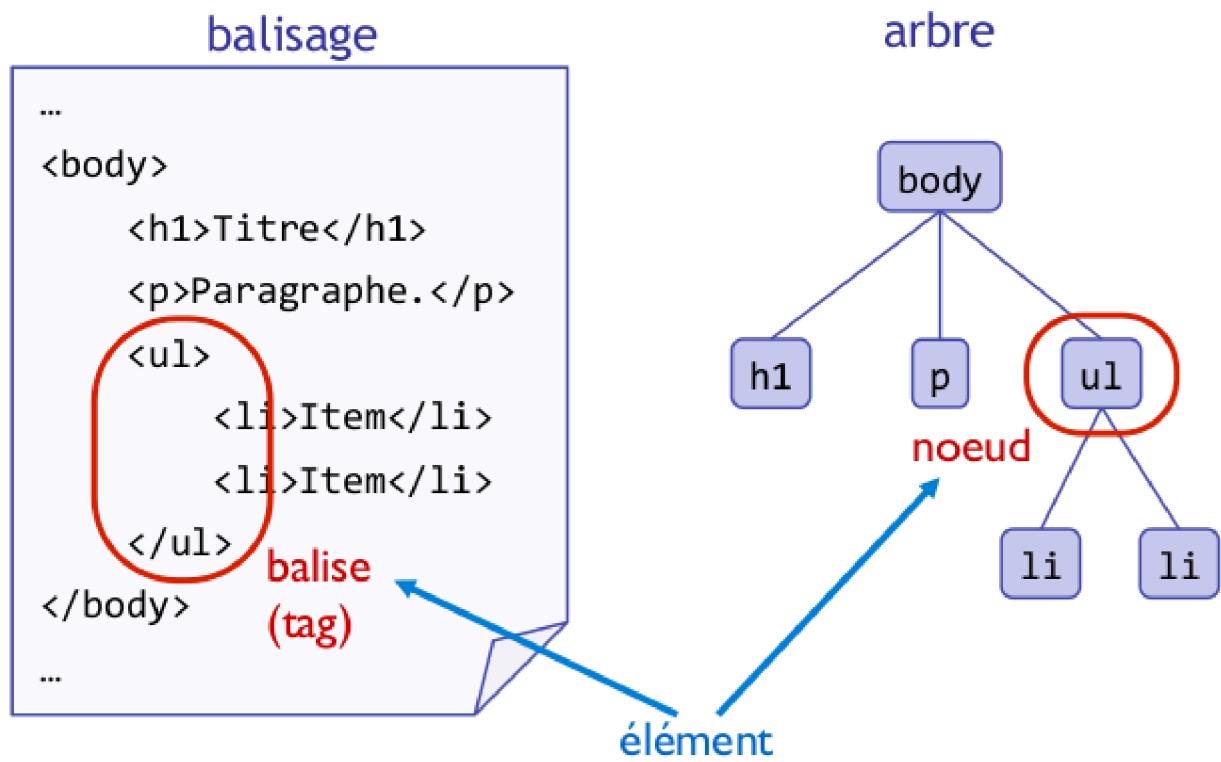
JAVASCRIPT

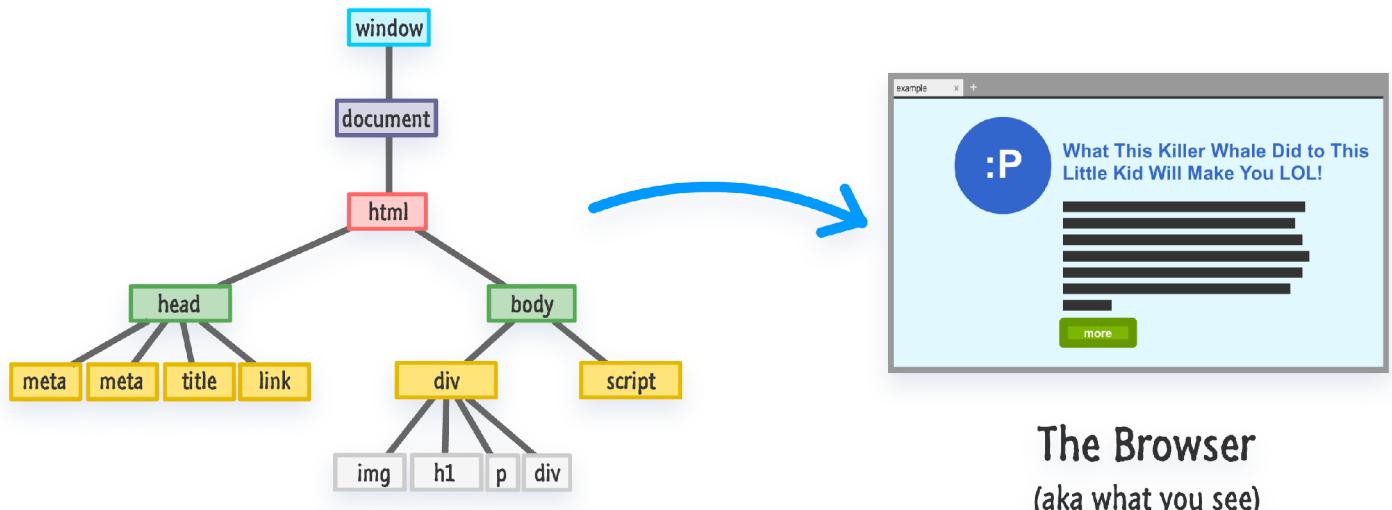
1. DOM

1.1. Définition

Le **Document Object Model** ou **DOM** (modèle objet de document) est une interface de programmation pour les documents HTML, XML et SVG.

- Le **DOM** (Document Object Model) est une interface qui fait partie du **BOM** (Browser Object Model) et grâce à laquelle nous allons pouvoir manipuler le contenu HTML et les styles de nos pages.
- Il fournit une représentation structurée du document sous forme d'un arbre et définit la façon dont la structure peut être manipulée par les programmes, en termes de style et de contenu.
- Le DOM représente le document comme un ensemble de nœuds et d'objets possédant des propriétés et des méthodes.
- Les nœuds peuvent également avoir des gestionnaires d'événements qui se déclenchent lorsqu'un événement se produit.
- Cela permet de manipuler des pages web grâce à des scripts et/ou des langages de programmation.
- Les nœuds peuvent être associés à des gestionnaires d'événements.
- Une fois qu'un événement est déclenché, les gestionnaires d'événements sont exécutés.

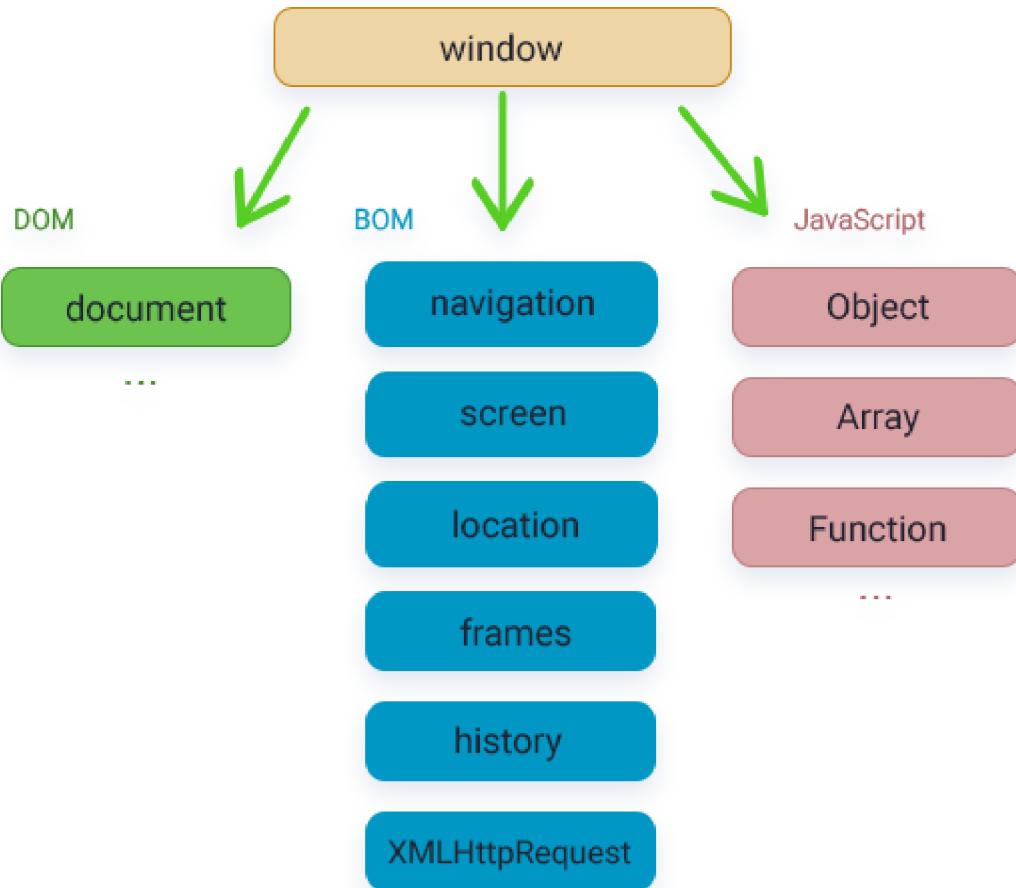




The Browser
(aka what you see)

The DOM

Ne pas confondre le DOM avec le BOM (Browser Object Model) !



1.2. Sélectionner des éléments

1.2.1. Accès direct aux éléments du DOM

1.2.1.1. méthode `getElementById()`

Permet de sélectionner un élément d'identifiant donné dans une page

```
const element = document.getElementById(id);
```

javascript

1.2.1.2. méthode `getElementsByName()`

Permet de sélectionner les éléments portant un nom de balise donné dans une page.

La liste renournée est *live*, c'est à dire qu'elle se met à jour automatiquement à chaque changement de l'arbre DOM.

```
let elements = document.getElementsByName(tagName)
```

javascript

1.2.1.3. méthode `getElementsByClassName()`

Renvoie un objet de type tableau de tous les éléments enfants qui ont tous les noms de classe données.

```
let element = document.getElementsByClassName('test')
```

javascript

1.2.1.4. méthode `querySelector()`

Retourne le premier élément dans le document correspondant au sélecteur (ou groupe de sélecteurs) spécifié(s), ou `null` si aucune correspondance n'est trouvée.

```
let element = document.querySelector(".maclasse");
```

javascript

1.2.1.5. méthode `querySelectorAll()`

Renvoie une `NodeList` statique représentant une liste des éléments du document qui correspondent au groupe de sélecteurs spécifiés. On parcourt cette NodeList avec une boucle `forEach()`.

```
const res = document.querySelectorAll("p");
```

javascript

1.2.1.6. méthode getElementByName

Renvoie un objet `NodeList` contenant la liste des éléments portant un attribut `name` avec la valeur spécifiée en argument sous forme d'objet.

On utilise cette méthode pour sélectionner des éléments de formulaire par exemple.

1.2.1.7. accès avec les propriétés de Document

L'interface `Document` fournit également des propriétés qui permettent d'accéder directement à certains éléments ou qui retournent des objets contenant des listes d'éléments.

```
corps = document.body;
tete = document.head;
titre = document.title;
...
```

javascript

1.3. Manipuler le texte

1.3.1. `textContent` – définir ou obtenir le text d'un noeud.

Représente le contenu textuel d'un noeud et de ses descendants.

```
<div id="note">
    Bonjour !
    <span style="display:none">Texte caché.</span>
    <!-- my comment -->
</div>
```

html

Obtenir le texte :

```
let elt = document.getElementById('note');
console.log(elt.textContent);
// Bonjour !
// Texte caché.
```

javascript

Définir le texte :

```
let text = element.textContent;
element.textContent = "nouveau texte";
```

javascript

1.3.2. innerText

Propriété représentant le contenu textuel « visuellement rendu » d'un nœud.

Utilisé en lecture, il renvoie une approximation du texte que l'utilisateur obtiendrait s'il sélectionnait le contenu d'un élément avec le curseur, et le copiait dans le presse-papier.

```
<div id="note">
  Bonjour !
  <span style="display:none">Texte caché.</span>
  <!-- my comment -->
</div>
```

html

```
let elt = document.getElementById('note');
console.log(elt.innerText); // Bonjour
```

javascript

1.3.3. innerHTML

Récupère ou définit la syntaxe HTML décrivant les descendants de l'élément.

```
<ul id="menu">
  <li>Home</li>
  <li>Contact</li>
</ul>
```

html

```
let menu = document.getElementById('menu');
console.log(menu.innerHTML);
```

javascript

sortie :

```
<li>Home</li>
<li>Services</li>
```

html

1.4. Naviguer dans les éléments DOM

1.4.1. Nœuds parents

1.4.2. parentNode et parentElementNode

La propriété en lecture seule `parentNode` renvoie le parent du nœud spécifié dans l'arborescence de DOM.

```
<div id="main">
  <p class="note">Texte</p>
</div>
```

html

```
let note = document.querySelector('.note');
console.log(note.parentNode);
// div
```

javascript

1.4.3. Nœuds enfants

1.4.3.1. childNodes

Renvoie une `NodeList` de noeuds enfants de l'élément donné avec le premier noeud enfant affecté à l'index 0.

```
let noeuds = elementDeReference.childNodes;
```

javascript

1.4.3.2. children

Renvoie une collection dynamique qui contient les éléments enfants de l'élément courant (celui sur lequel elle a été appelée).

`Element.children` contient uniquement des nœuds qui sont des éléments. Pour obtenir l'ensemble des enfants, y compris les nœuds qui ne sont pas des éléments comme les textes et les commentaires, il faudra utiliser `childNodes`.

```
const monElement = document.getElementById('toto');
for (let i = 0; i < monElement.children.length; i++) {
  console.log(monElement.children[i].tagName);
}
```

javascript

1.4.3.3. `firstChild` et `firstChildElement`

La propriété en lecture seule `Node.firstChild` renvoie le premier nœud enfant de l'arbre ou `null` s'il n'en a pas.

Si le noeud est un `Document`, il renvoie le premier noeud de la liste de ses enfants directs.

```
<body>
  <ul id="menu">
    <li class="first">Home</li>
    <li>Products</li>
    <li class="current">Customer Support</li>
    <li>Careers</li>
    <li>Investors</li>
    <li>News</li>
    <li class="last">About Us</li>
  </ul>
</body>
```

html

```
let firstChild = parentElement.firstChild;
let content = document.getElementById('menu');
let firstChild = content.firstChild.nodeName;
console.log(firstChild);
```

javascript

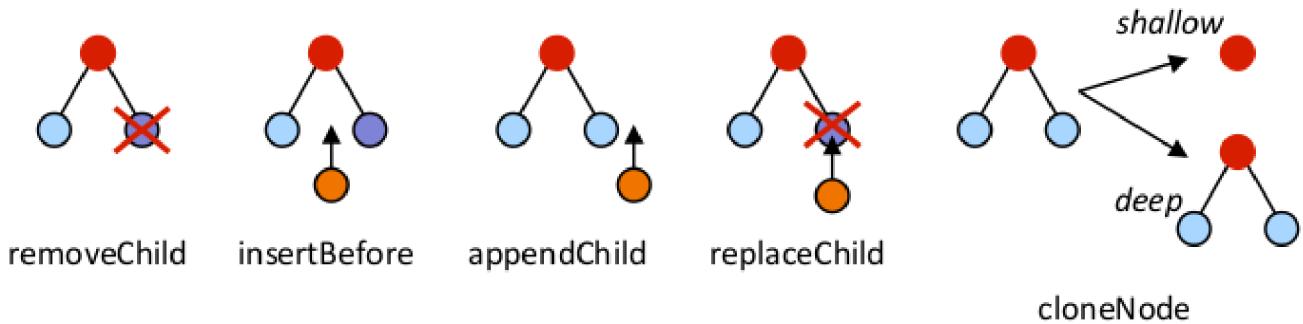
1.4.3.4. `lastChild` et `lastChildElement`

à venir

1.4.4. relation parallèle dans l'arbre : `previousSibling` et `nextSibling`

La propriété en lecture seule `Node.nextSibling` renvoie le nœud (Node) suivant immédiatement le nœud spécifié dans la liste des enfants `childNodes` de son nœud parent, ou `null` si le noeud spécifié est le dernier dans cette liste.

1.5. Manipuler les éléments



1.5.1. Crée un élément : `createElement()`

Crée un élément HTML du type spécifié par `tagName`.

```
let element = document.createElement(htmlTag);
```

javascript

Genèse : de la création à l'affectation d'un élément dans la page.

```
let div = document.createElement('div'); // création
div.innerHTML = '<p>CreateElement example</p>'; // ajout d'un bout de code HTML
document.body.appendChild(div); // rattacheement de la div au document
```

javascript

1.5.2. Insérer un élément : `prepend()`

La méthode `Element.prepend()` permet d'insérer un ensemble d'objets `Node` ou des chaînes de caractères avant le premier élément enfant de l'élément courant.

Les chaînes de caractères sont insérées comme des noeuds Texte.

```
let div = document.createElement("div");
let p = document.createElement("p");
let span = document.createElement("span");
div.append(p);
div.prepend(span);
console.log(div.childNodes); // NodeList [ <span>, <p> ]
```

javascript

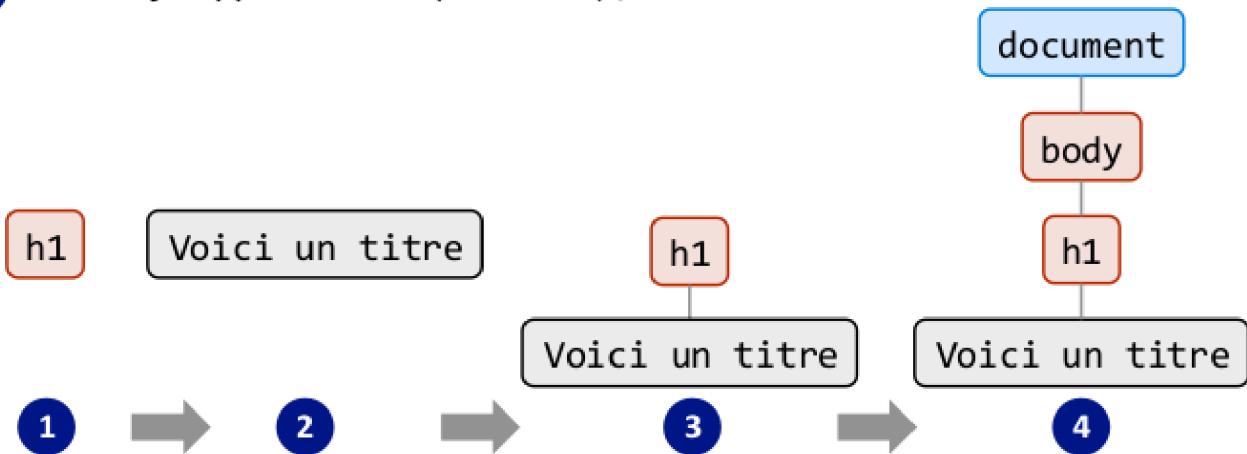
La méthode `Element.append()` ajoute un ensemble d'objets `Node` ou de chaînes de caractères après le dernier enfant de l'élément courant.

Les chaînes de caractères sont insérées comme des noeud Texte.

```
let div = document.createElement("div");
let p = document.createElement("p");
div.append(p);
console.log(div.childNodes); // NodeList [ <p> ]
```

javascript

```
1 var d = window.document;
2 var noeudH1 = d.createElement("h1");
3 var noeudTexte = d.createTextNode("Voici un titre");
4 noeudH1.appendChild(noeudTexte);
4 d.body.appendChild(noeudH1);
```



1.6. Déplacer un noeud

A venir

1.7. Cloner un noeud

A venir

1.8. Supprimer un noeud

A venir

1.9. Modifier les attributs

1.9.1. setAttribute()

Ajoute un nouvel attribut ou change la valeur d'un attribut existant pour l'élément spécifié. Si l'attribut existe déjà, la valeur est mise à jour, sinon un nouvel attribut est ajouté avec le nom et la valeur spécifiés.

```
Element.setAttribute(name, value);
```

javascript

```
let btnEnvoi = document.querySelector('#envoi');
if (btnEnvoi) {
    envoi.setAttribute('name', 'send');
    envoi.setAttribute('disabled', '');
}
```

javascript

1.9.2. getAttribute()

`getAttribute` renvoie la valeur d'un attribut donné de l'élément spécifié. Si l'attribut n'existe pas, la valeur renvoyée sera soit `null` soit `""` (une chaîne vide).

```
let value = element.getAttribute(name);
```

javascript

```
<a href="https://www.monsite.fr" target="_blank" id="js">monsite</a>
```

html

```
let lien = document.querySelector('#js');
if (lien) {
    let target = lien.getAttribute('target');
    console.log(target);
} // _blank
```

javascript

1.9.3. removeAttribute()

La méthode `removeAttribute()`, rattachée à l'interface `Element`, supprime l'attribut ayant le nom indiqué de l'élément.

```
element.removeAttribute(nomAttribut);
```

javascript

```
<a href="https://www.monsite.fr" target="_blank" id="js">monsite</a>
```

html

```
let lien = document.querySelector('#js');
if (lien) {
    let target = lien.removeAttribute('target');
    console.log(target);
} // rien à afficher
```

javascript

1.9.4. hasAttribute()

La méthode `Element.hasAttribute()` renvoie une valeur booléenne indiquant si l'élément courant possède l'attribut spécifié ou non.

```
var result = element.hasAttribute(name);
```

javascript

```
let btnEnvoi = document.querySelector('#envoi');
if (btnEnvoi) {
    let disabled = btnEnvoi.hasAttribute('disabled');
    console.log(disabled);
}
```

javascript

1.10. Manipuler les styles

1.10.1. définir un style en ligne

La propriété en lecture seule `style` renvoie le style *en incise* d'un élément sous la forme d'un objet `CSSStyleDeclaration` contenant une liste de l'ensemble des propriétés de style pour cet élément dont les valeurs sont celles des attributs défini par l'attribut HTML `style`.

```
element.style.color = 'red';
```

javascript

```
element.style.color = 'blue';
element.style.cssText += 'color:red; background-color:yellow';
```

javascript

```
function css(e, styles) {  
    for (const property in styles)  
        e.style[property] = styles[property];  
}
```

javascript

1.10.2. récupérer un style en ligne

La propriété `style` n'est pas utile pour tout savoir des styles appliqués à l'élément, car elle ne représente que les déclarations CSS appliquées à l'élément via l'attribut HTML `style` et pas celles provenant d'autres sources.

Pour obtenir l'ensemble des propriétés CSS d'un élément, il faudra plutôt utiliser `window.getComputedStyle()`.

1.10.3. `getComputedStyle()`

La méthode `window.getComputedStyle()` donne la valeur calculée finale de toutes les propriétés CSS sur un élément.

```
let style = window.getComputedStyle(element [,pseudoElement]);
```

javascript

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>getComputedStyle</title>
    <style type="text/css">
        .message {
            background-color: #fff3d4;
            border: solid 1px #f6b73c;
            padding: 20px;
            color: black;
        }
    </style>
</head>
<body>
    <p class="message" style="color:red">
        Présentation de la méthode getComputedStyle()
    </p>
    <script>
        let message = document.querySelector('.message');
        let style = getComputedStyle(message);
        console.log('color:', style.color);
        console.log('background color:', style.backgroundColor);
    </script>
</body>
</html>
```

html

Ce qui donnera en sortie :

```
color: rgb(255, 0, 0)
background color: rgb(255, 243, 212)
```

javascript

1.10.4. Les classes avec `classList`

1.10.4.1. récupérer une ou des classes

La propriété en lecture seule `Element.classList` retourne une collection directe `DOMTokenList` des attributs de classe de l'élément.

L'utilisation de `classList` est une alternative à la propriété `element.className` qui renvoie une chaîne composée de la liste des classes, séparées par des espaces.

```
<div id="content" class="main red">JavaScript classList</div>
```

html

```
let div = document.querySelector('#content');
for (let cssClass of div.classList) {
    console.log(cssClass);
}
```

javascript

Sortie :

```
main
red
```

1.10.4.2. classList.add()

Ajout d'une seule classe :

```
let div = document.querySelector('#content');
div.classList.add('info');
```

javascript

Ajout de plusieurs classes :

```
let div = document.querySelector('#content');
div.classList.add('info','visible','block');
```

1.10.4.3. classList.remove()

Suppression d'une seule classe :

```
let div = document.querySelector('#content');
div.classList.remove('info');
```

javascript

Suppression de plusieurs classes :

```
let div = document.querySelector('#content');
div.classList.remove('info','visible','block');
```

javascript

1.10.4.4. classList.replace()

```
let div = document.querySelector('#content');
div.classList.replace('info','warning');
```

javascript

1.10.4.5. classList.contains()

```
let div = document.querySelector('#content');
div.classList.contains('warning'); // true
```

javascript

1.10.4.6. classList.toggle()

```
let div = document.querySelector('#content');
div.classList.toggle('visible');
```

javascript

2. Les événements

2.1. Types événements courants

2.1.1. Événements de la souris

Événement	Description
click	Déclenché lorsqu'un élément est cliqué
dblclick	Déclenché lorsqu'un élément est double-clique
mousedown	Déclenché lorsqu'un bouton de la souris est enfoncé sur un élément
mouseup	Déclenché lorsqu'un bouton de la souris est relâché sur un élément
mousemove	Déclenché lorsqu'un élément est survolé par la souris et continue de bouger dans la zone de survol.
mouseenter	Déclenché lorsque le curseur de la souris entre dans la zone d'un élément et ne se propage pas aux éléments enfants. MEILLEURE PRATIQUE que mouseover. Ne se propage pas.
mouseleave	Déclenché uniquement lorsque le curseur de la souris quitte la zone de l'élément, même s'il est encore sur un élément enfant. MEILLEURE PRATIQUE que mouseout. Ne se propage pas.
mouseover	Déclenché lorsque le curseur de la souris survole la zone d'un élément, y compris les éléments enfants.
mouseout	Déclenché lorsque le curseur de la souris quitte la zone de l'élément et tous ses éléments enfants.
contextmenu	Déclenché lorsqu'un menu contextuel est déclenché (par exemple, par un clic droit de la souris)

2.1.2. Événements du clavier

Événement	Description
keydown	Déclenché lorsqu'une touche est enfoncée
keyup	Déclenché lorsqu'une touche est relâchée
keypress	Déclenché lorsqu'une touche est enfoncée et maintenue enfoncée. Cet événement est devenu obsolète et son utilisation n'est pas recommandée.

2.1.3. Événements de formulaire

Événement	Description
submit	Déclenché lorsqu'un formulaire est soumis
reset	Déclenché lorsqu'un formulaire est réinitialisé
focus	Déclenché lorsqu'un élément reçoit le focus

Événement	Description
blur	Déclenché lorsqu'un élément perd le focus
change	Déclenché lorsqu'une valeur d'un élément est modifiée
input	Déclenché lorsqu'un utilisateur entre des données dans un élément de formulaire, comme un champ de texte ou une zone de texte
select	Texte sélectionné par l'utilisateur.

2.1.4. Presse-papier

Événement	Description
copy	Se déclenche lorsque l'utilisateur tente de copier du contenu.
cut	Se déclenche lorsque l'utilisateur tente de couper du contenu.
paste	Se déclenche lorsque l'utilisateur tente de coller du contenu.

2.1.5. Événements de la fenêtre

Événement	Description
load	Déclenché lorsque la page a fini de charger
unload	Déclenché lorsque la page est déchargée
resize	Déclenché lorsque la taille de la fenêtre est modifiée
scroll	Déclenché lorsqu'un utilisateur fait défiler la page

2.1.6. Événements du document (cycle de vie d'une page)

Événement	Description	Spécificités
DOMContentLoaded	Se déclenche lorsque le DOM initial de la page est complètement chargé et analysé.	Ne nécessite pas que les ressources externes soient chargées. C'est souvent utilisé pour initialiser des scripts avant que la page ne soit complètement chargée.
load	Se déclenche lorsque la page entière et toutes ses ressources associées ont été complètement chargées.	Cet événement garantit que toutes les ressources, y compris les images, stylesheets, et scripts, sont chargées. Il est souvent utilisé pour initialiser des éléments qui dépendent de ressources externes.
beforeunload	Se déclenche avant que le document soit déchargé.	Permet de demander une confirmation à l'utilisateur avant de quitter la page, surtout s'il y a des données non sauvegardées.
unload	Se déclenche une fois que la page est sur le point d'être complètement déchargée.	Cet événement est moins couramment utilisé. Il ne permet pas d'effectuer de nombreuses actions, notamment en raison de restrictions de sécurité. Il est principalement utilisé pour nettoyer et libérer des ressources.

2.2. Les 3 manières d'implémenter un gestionnaire d'événement(s)

2.2.0.1. Utiliser un attribut HTML (pas du tout recommandé)

```
<button onclick="alert('Bouton cliqué')">Cliquez moi !</button>
<p onmouseover="this.style.backgroundColor='orange'">survolez-moi avec la souris</p>
```

html

2.2.0.2. Utiliser une propriété d'événement javascript (peu recommandé)

"on" + nom de l'événement géré.

Exemple : onclick, onmouseenter, onmouseout

```
let btn = document.querySelector('button');
let p1 = document.getElementById('p1');

// couplage de la fonction à l'événement en invoquant le nom de la fonction.
btn.onclick = message;
function message() {
  alert('coucou');
};

// couplage de la fonction à l'événement à la volée.
associe
p1.onmouseover = function () { this.style.color = 'red' };
p1.onmouseout = function () { this.style.color = 'black' };
```

javascript

2.2.0.3. Utilisation de la méthode addEventListener (bonne pratique – méthode plus performante)

Cela offre en outre l'avantage de permettre le rattachement d'un gestionnaire à **plusieurs** événements.

```
let btn = document.querySelector('button');
btn.addEventListener('click', () => prompt('Comment allez-vous ?'));
```

javascript

On peut associer un même événement et provoquer plusieurs actions distinctes.

```
let btn = document.querySelector('button');
btn.addEventListener('click', maFonction);
btn.removeEventListener('click', maFonction);
```

javascript

2.3. Propagation d'événements

La propagation des événements est un processus qui permet aux événements de se propager à travers la hiérarchie du Document Object Model (DOM). Il existe deux phases de propagation des événements : le **capturing** (capture) et le **bubbling** (bouillonnement).

Lors de l'utilisation de la méthode `addEventListener()`, il est possible de spécifier la phase de propagation au cours de laquelle le gestionnaire d'événements doit être déclenché en passant un troisième argument.

Ce troisième argument est un **objet d'options** ou un **booléen** :

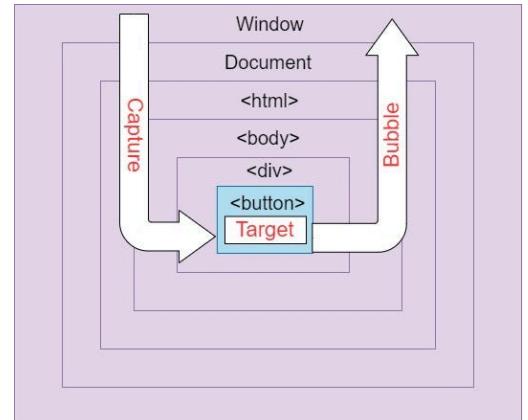
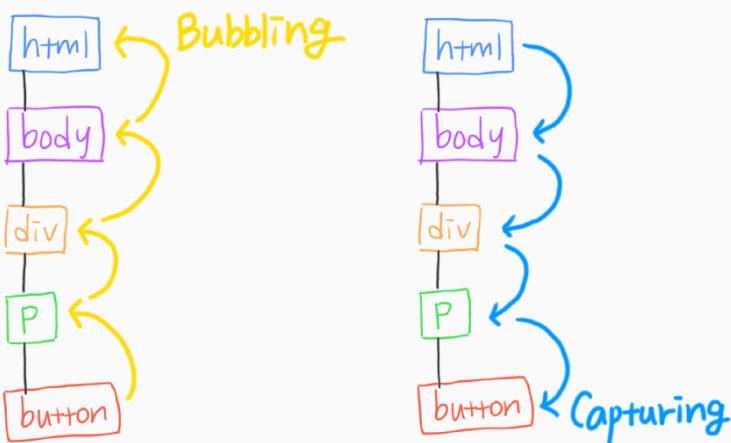
- Si l'argument est `false` ou si l'objet d'options a la propriété `capture` définie à `false`, le gestionnaire d'événements sera déclenché pendant la phase de **bubbling**.
- Si l'argument est `true` ou si l'objet d'options a la propriété `capture` définie à `true`, le gestionnaire d'événements sera déclenché pendant la phase de **capture**.

Exemple :

Ici, le gestionnaire d'événements sera déclenché pendant la phase de **capture**, avant que l'événement n'atteigne l'élément cible.

```
document.querySelector("#parent").addEventListener("click", function(event) {
  console.log("Parent element clicked during capturing phase.");
}, true);
```

javascript



2.3.1. Capturing (capture)

Lors de la phase de **capture**, l'événement se propage depuis l'objet **Window** vers l'élément cible, en passant par tous les éléments parents dans la hiérarchie du DOM. Les gestionnaires d'événements associés à ces éléments sont déclenchés dans l'ordre de la hiérarchie, du plus externe au plus interne.

2.3.2. Bubbling (bouillonnement)

Après que l'événement a atteint l'élément cible, il commence à remonter la hiérarchie du DOM vers l'objet `Window`. Cette phase s'appelle le **bubbling**. Pendant le bubbling, les gestionnaires d'événements associés aux éléments parents sont déclenchés dans l'ordre inverse de la hiérarchie, du plus interne au plus externe.

- Il est important de noter que tous les événements ne passent pas par les phases de `capturing` et de `bubbling`.
- Par exemple, l'événement `focus` ne se propage pas et n'atteint que l'élément cible.

2.3.3. Arrêt de la propagation

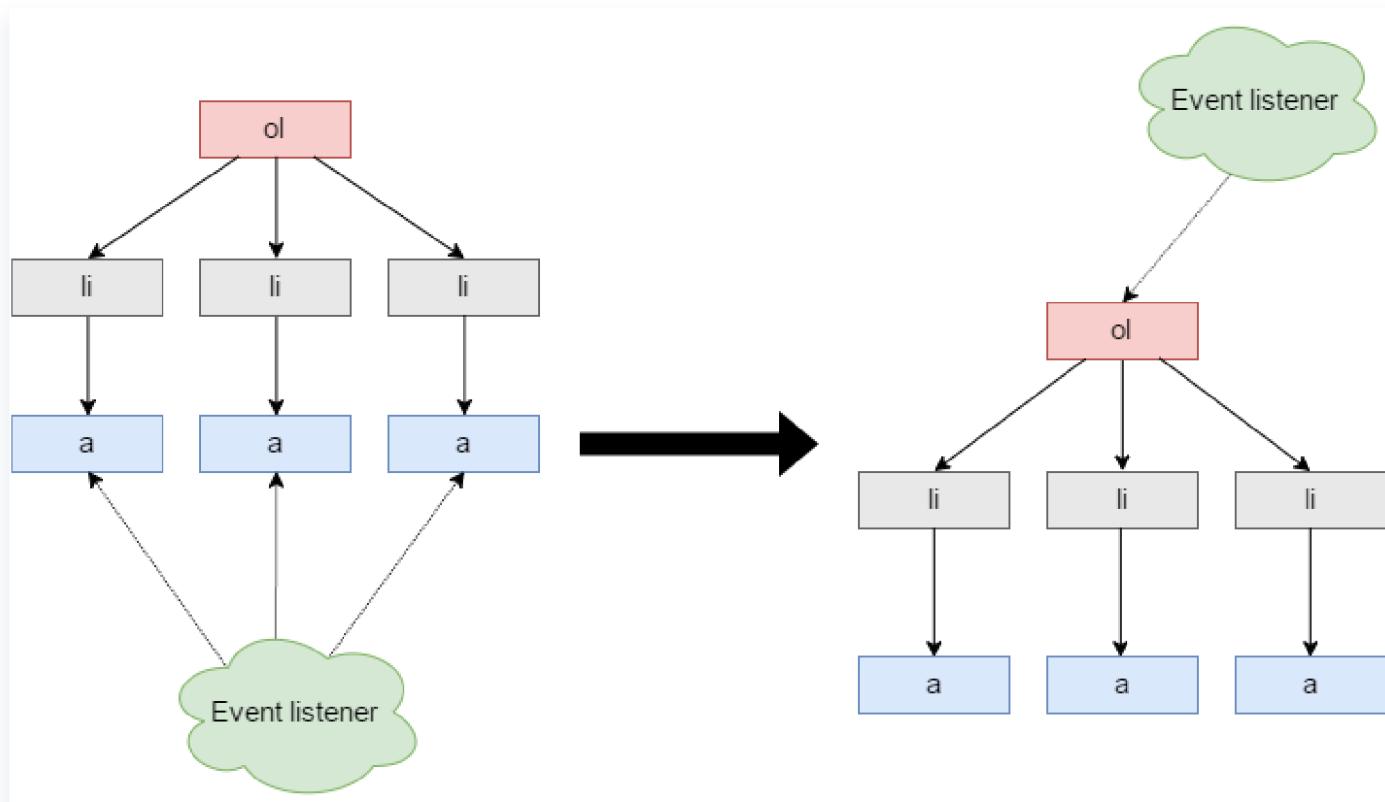
Dans certains cas, il peut être nécessaire d'empêcher la propagation d'un événement à d'autres éléments. Pour cela, vous pouvez utiliser la méthode `stopPropagation()` sur l'objet Event :

```
document.querySelector("#child").addEventListener("click", function(event) {
  event.stopPropagation();
  console.log("Child element clicked, propagation stopped.");
});
```

javascript

2.4. Délégation d'événements

La **délégation d'événements** est une technique utilisée pour **gérer efficacement** les événements sur un **grand nombre d'éléments enfants**. Au lieu d'attacher un gestionnaire d'événements à chaque élément enfant, **un seul gestionnaire d'événements est attaché à l'élément parent**. Ce gestionnaire d'événements utilise la propagation des événements pour réagir aux événements déclenchés par les éléments enfants.



```
<ol id="liens">
  <li><a>lien 1</a></li>
  <li><a>lien 2</a></li>
  <li><a>lien 3</a></li>
</ol>
```

html

```
document.getElementById('liens').addEventListener('click', function(e) {
  if (e.target.tagName === 'A') {
    console.log(e.target.innerText + ' a été cliqué!');
  }
});
```

javascript

2.4.1. e.target

En JavaScript, lorsqu'un événement est déclenché (par exemple, un clic sur un bouton, un mouvement de la souris, une frappe au clavier, etc.), un objet événement est généré. Cet objet contient des informations sur l'événement en question, comme le type d'événement, l'élément sur lequel il s'est produit, la position de la souris, et bien d'autres propriétés.

La propriété `e.target` (où `e` est généralement le nom de la variable utilisée pour l'objet événement) fait référence à l'élément HTML qui a déclenché l'événement.

Exemple :

Si vous avez un bouton HTML et que vous y attachez un gestionnaire d'événements de clic, `e.target` désignera cet élément bouton lorsque l'utilisateur cliquera dessus.

Cliquez sur moi!

```
document.getElementById('monBouton').addEventListener('click', function(e) {
  console.log(e.target); // Affichera l'élément <button> dans la console
});
```

javascript

2.5. preventDefault()

Il est parfois nécessaire d'annuler un événement pour empêcher l'exécution de certaines actions par défaut. La méthode `preventDefault()` est utilisée pour empêcher l'action par défaut associée à un événement. (empêcher le comportement par défaut d'un lien ou bien la soumission de formulaire).

Exemple n°1 :

Ici, lorsque l'utilisateur clique sur le lien, l'action par défaut (ouvrir la page liée) est empêchée et le message "*Link clicked, default action prevented !*" est affiché dans la console.

```
document.querySelector("a").addEventListener("click", function(event) {
  event.preventDefault();
  console.log("Link clicked, default action prevented.");
});
```

javascript

Exemple n°2 : Empêcher la soumission d'un formulaire lorsque certaines conditions ne sont pas remplies.

```
document.querySelector("form").addEventListener("submit", function(event) {
  if (!validateForm()) {
    event.preventDefault();
    console.log("Form submission prevented, validation failed.");
  }
});
```

javascript

2.6. Supprimer un gestionnaire d'événements

Pour supprimer un gestionnaire d'événements, vous pouvez utiliser la méthode `removeEventListener`.

Voici comment cela fonctionne:

1. Lorsque vous ajoutez un gestionnaire d'événements avec `addEventListener`, assurez-vous d'utiliser une fonction nommée ou une référence de fonction, plutôt qu'une fonction anonyme. C'est parce que vous avez besoin de cette référence pour le supprimer plus tard.
2. Utilisez la méthode `removeEventListener` avec **le même nom d'événement** et **la même référence de fonction** pour supprimer le gestionnaire d'événements.

Exemple:

Étape 1: Créez une référence de fonction

```
function handleClick() {  
    console.log('Le bouton a été cliqué!');  
}
```

// Étape 2: Ajoutez le gestionnaire d'événements

```
document.getElementById('monBouton').addEventListener('click', handleClick);
```

// Étape 3: Supprimez le gestionnaire d'événements lorsque nécessaire

```
document.getElementById('monBouton').removeEventListener('click', handleClick);
```