

# Programmation Web – Avancé

## JavaScript & Node.js

### Partie 05 : Le DOM et la gestion d'événements

#### Version 2020



Attribution –  
Partage dans les  
Mêmes Conditions  
4.0 International  
(CC BY-SA 4.0)

*Presentation template*  
by [SlidesCarnival](https://www.slidescarnival.com/)

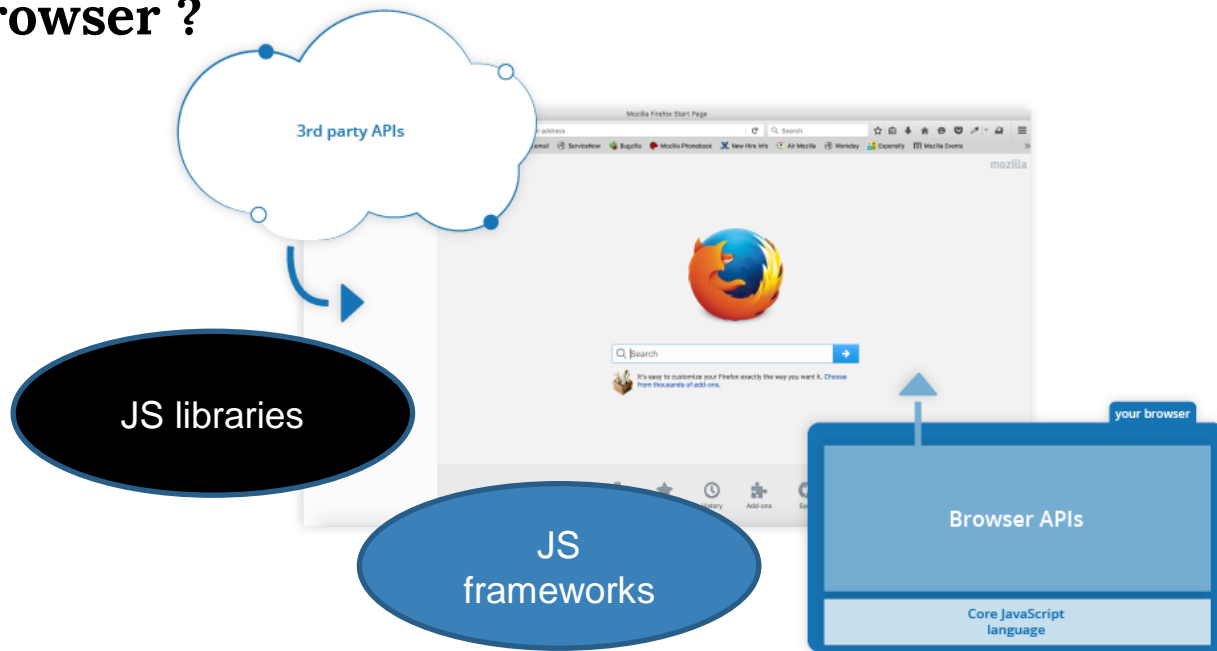


# Interaction de base avec l'API DOM

Comment interagir avec son browser ?



## Comment interagir avec son browser ?



**Relation entre APIs,  
le browser et le JS [15.]**



## Comment interagir avec son browser ?

---

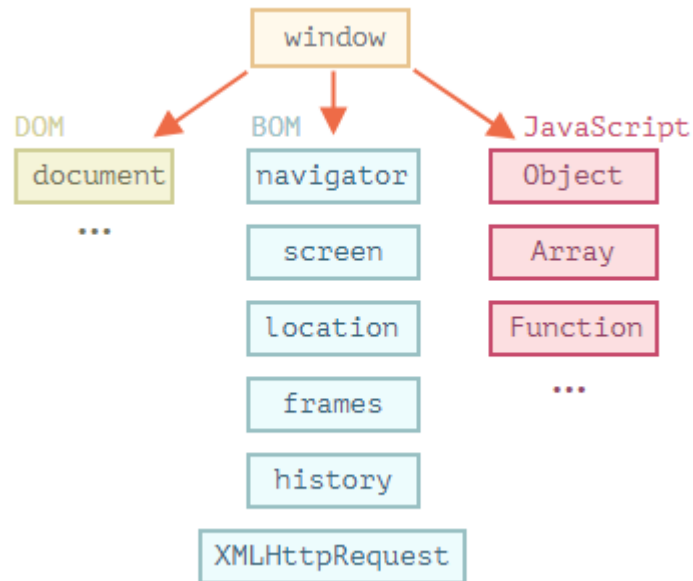
### ● Browser APIs :

- Manipulation de documents : **DOM API**
- Dessiner / animer : **Canvas** & **WebGL**
- Communiquer des données avec un serveur :  
**XMLHttpRequest** & **Fetch API** (techniques AJAX)
- Audio & Video APIs
- Web Storage APIs (cookies, localStorage...)
- ...



## Comment interagir avec son browser ?

- Utilisation d'objets JS associés aux Browser APIs
- Browser Object Model : pas de standard...



**Environnement du browser**

**[23.]**



## Comment interagir avec son browser ?

---

- **window** : objet manipulant la fenêtre du browser même : **alert()**, **prompt()**...
- **navigator** (===**window.navigator**) : objet contenant de l'information sur le browser
  - **navigator.appName**
  - **navigator.appVersion**



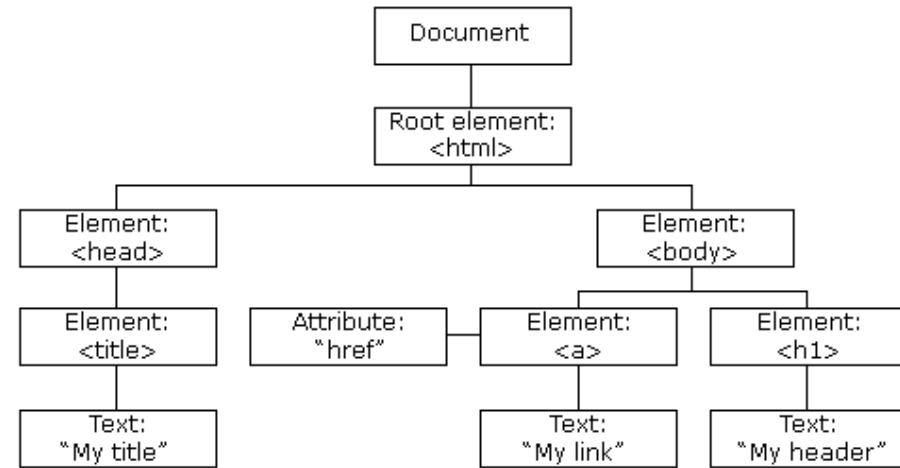
## Comment interagir avec son browser ?

- **document** : (**===window.document**) objet manipulant le corps de l'HTML de la page.
- **location** : (**===window.location**) objet représentant l'URL de la page.
  - **location.href** : cette URL.
  - **location.reload()** : recharge la page.
  - **location.replace('https://...')** : navigue à cette URL.



## Document Object Model (DOM)

- Représentation objet d'une page HTML / CSS par le browser
- Mise à jour automatique d'une page du browser lors de manipulation JS sur un objet issu du DOM
- MàJ tant du HTML que du CSS : « CRUD operations » sur les éléments HTML et leur style (CSS)



**The HTML DOM Tree of Objects [24.]**





## Accéder à un élément HTML

- **getElementById()** : retourner l'élément dont l'attribut ID contient le paramètre donné

```
<button id="myBtn1">Click please</button>
```



```
let btn1 = document.getElementById("myBtn1");
```





## Accéder à un élément HTML

- **querySelector()** : retourner le 1er élément qui match un sélecteur CSS

```
<button id="myBtn2">Click please</button>  
<div class="message"></div>
```



```
let btn2 = document.querySelector("#myBtn2"); // HTML id attribute  
let msg = document.querySelector(".message"); // CSS class name  
let duplicateMsg = document.querySelector("div"); // HTML tag name
```





## Accéder à un élément HTML

- **querySelectorAll()** : retourner tous les élément qui match un CSS selector (sous forme d'un **NodeList** object)

```
<button id="myBtn1">Click please</button>
<button id="myBtn2">Click please</button>
<button id="myBtn3">Click please</button>
```



```
let btnArray = document.querySelectorAll("button");
// Nodelist of HTMLButtonElements is returned
let btn3 = btnArray[2];
```





## Accéder à un élément HTML

---

- Autres méthodes [\[25.\]](#)



## **Norme ECMAScript : le JS standard**

---

- Débouche sur multiples implémentations du JS (ActionScript, JScript, JavaScript, CommonJS...)
- Utilisation primaire pour des scripts côté client, mais de plus en plus utilisé côté serveur avec Node.js
- Version en cours approuvée : 11th Edition – ECMAScript 2020



## Norme ECMAScript : le JS standard

---

- Exemple de nouveaux éléments ES6 (ou ES 2015) : arrow functions, mots-clés **let** et **const**, promise, **class**...
- Suivre le développement via [\[26.\]](#) ou [\[27.\]](#)



## **Norme ECMAScript : le JS standard**

---

- Support pour tous les browsers modernes : ES5
- Support de ES6 (ES 2015) par la majorité des browsers, mais pas IE !



## Norme ECMAScript : le JS standard

- ◎ « **strict mode** » pour du JS moderne
  - Plus de feedback d'erreurs, code plus sûr !
  - Ajout de **"use strict"**; au début d'un script ou d'une fonction
  - Automatiquement ajouté dans les classes & modules
  - Détails [\[28.\]](#)

```
"use strict";  
undeclaredVar = "I am undeclared..."; // Uncaught ReferenceError:  
                                         undeclaredVar is not defined  
console.log(undeclaredVar);
```





## **Norme ECMAScript : le JS standard**

- Ajout de "use strict"; au début de vos scripts
- Ecriture de code moderne conforme à ES6
- En cas de support pour de vieux browsers, utilisation de Babel pour convertir du code ES6+ vers des versions de JS plus ancienne





# Introduction à la gestion d'événements

La programmation événementielle... C'est quoi ?



## Introduction à la gestion d'événements

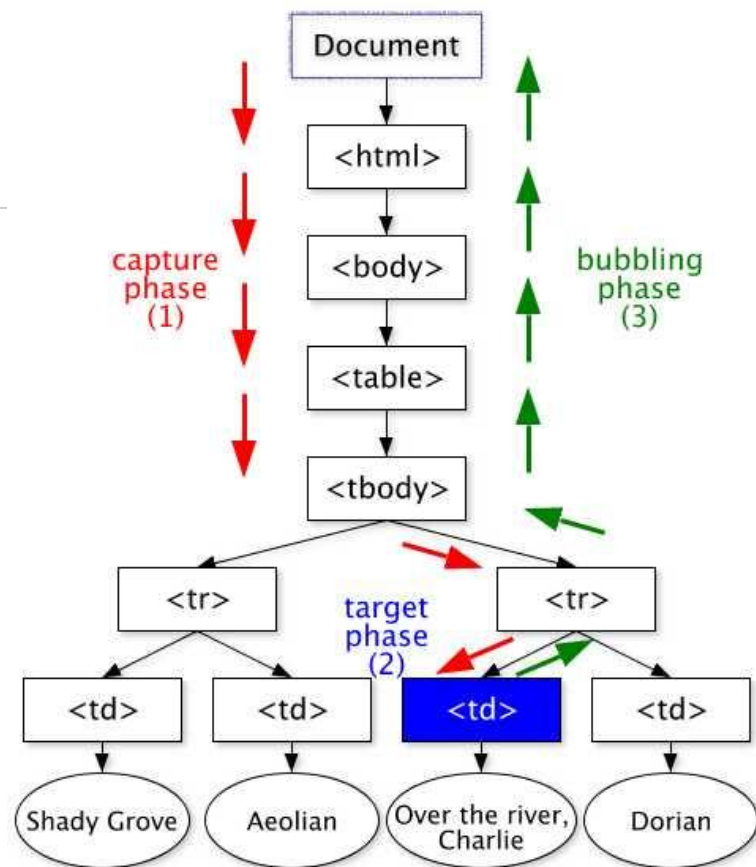
---

- Prise en compte des actions des utilisateurs suite à un événement du browser
- Ajout d'écouteurs d'événements sur des éléments du DOM pour définir des actions
- Type d'événement avec actions par défaut :
  - Exemples : submit d'un formulaire, hyperlink...
  - “Cancelable events” : écouteur d'événement déclenché par l'événement



## Flux des événements du DOM

1. Communication de l'événement au noeud cible par les ancêtres & gestion éventuelle
2. Gestion de l'événement par la cible
3. Communication & gestion potentielle de l'événement par les ancêtres de la cible



**Flux de traitement  
d'un événement [28.]**



## Enregistrement d'un écouteur d'événement via des propriétés

● Propriétés : **onclick**, **onfocus**, **ondblclick**...

```
<button id="myBtn1">Click please</button>
```



```
btn1.onclick = function () {  
  btn1.innerText = "You clicked on me : )";  
  console.log("btn.onclick::anonymous function");  
};
```





## Gestion des écouteurs d'événement via des méthodes modernes

### 🕒 `addEventListener()` & `removeEventListener()`

```
function onClickHandlerForBtn3() {  
  btn3.innerText = "I have also been clicked";  
  console.log("onClickHandlerForBtn3::click");  
}  
  
const onClickHandlerForBtn4 = () => {  
  console.log("onClickHandlerForBtn4::click");  
  btn3.removeEventListener("click", onClickHandlerForBtnExtra);  
};  
  
btn3.addEventListener("click", onClickHandlerForBtn3);  
btn3.addEventListener("click", onClickHandlerForBtnExtra);  
btn4.addEventListener("click", onClickHandlerForBtn4);
```



## Inline event handlers

```
<body onclick="console.log('click')">
```





## Introduction à la gestion d'événements

- Enregistrer vos écouteurs d'événements à l'aide de **addEventListener()**



- Détails [\[30.\]](#)
- Liste des événements [\[31.\]](#)





## DOM et gestion d'événements

### 🕒 DEMO : Gestion d'un clic et du DOM

MàJ du texte associé à des boutons quand on clique dessus.



## Callbacks

- Programmation d'actions asynchrones
- Fonction passée en argument à une autre fonction pour exécution quand l'action est finie
- “Event listener callback”

```
function onClickHandlerForBtn3() {  
    btn3.innerText = "I have also been clicked";  
    console.log("onClickHandlerForBtn3::click");  
}  
btn3.addEventListener("click", onClickHandlerForBtn3);
```



## Callbacks

### 🕒 DEMO : Création d'une callback

Appel d'une callback après avoir bouclé un million de fois

- Template literals [\[37.\]](#)





## Création d'une callback & appel

```
function runBigLoop(callback) {  
  const t0 = performance.now();  
  for (let index = 0; index < LOOP_ITERATIONS; index++) {  
    // do nothing  
  }  
  const t1 = performance.now();  
  const timeInSec = Math.round(t1 - t0);  
  callback(LOOP_ITERATIONS, timeInSec);  
}  
  
function isLooped(iterations,time) {  
  console.log(`duration to loop ${iterations} times : ${time} ms`, );  
  alert(`duration to loop ${iterations} times : ${time} ms`);  
}  
  
runBigLoop(isLooped);
```



## Event object

- Automatiquement passé à la callback d'un écouteur d'événements

```
<div class="message" id="msgBox"></div>
```



```
msg.addEventListener("mouseout", function (e) {  
  console.log("msg::mouseout: div id :" + e.target.id);  
  msg.innerText = "You have left the div tag";  
});
```





## Stopper une gestion d'événements par défaut

### 🕒 e.preventDefault()

```
const onSubmit = (e) => {  
  console.log("onSubmit::");  
  // Prevent the default behaviour of a form (data sent to URL specified in  
    action parameter)  
  e.preventDefault();  
}  
  
myForm.addEventListener("submit", onSubmit);
```



## DOM et gestion d'événements

- **DEMO : Gestion de mouseover et du DOM**  
MàJ du texte d'une DIV quand la souris passe dessus.



## Gestion d'un timer

- **setTimeout(f,t) et clearTimeout()** [\[32.\]](#) :  
exécution d'une fonction (**f**) à l'expiration d'un  
timer (après **t** ms)

```
btn1.addEventListener("click", delayedAlert);
btn2.addEventListener("click", clearAlert);
function delayedAlert() {
    timeoutID = window.setTimeout(
        window.alert, 2 * 1000, "That was really slow!");
}
function clearAlert() {
    window.clearTimeout(timeoutID);
}
```





## Introduction à la gestion d'événements

◎ **setInterval(f,t)** & **clearInterval()** : exécution de **f** tous les **t** ms sauf si **clearInterval()**

```
btn1.addEventListener("click", myStopFunction);
let myVar = setInterval(myTimer, 1000);
function myTimer() {
  let d = new Date();
  let t = d.toLocaleTimeString();
  document.getElementById("demo").innerHTML = t;
}
function myStopFunction() {
  clearInterval(myVar);
}
```