

Les bases du Javascript

Version : Les bases du Javascript

Date : 7/10/2019

Auteur(s) : Florent TETART

Table des matières

1. Javascript : les bases	3
2. Le DOM : Document Object Model	19
3. Les événements	23

1. Javascript : les bases

Le Javascript est un langage qui a plusieurs caractéristiques :

- Interprété
- Typage faible
- Objet
- Dynamique
- Asynchrone
- Monothread

Interprété

Les instructions d'un programme en Javascript sont interprétées par le moteur Javascript au moment de l'exécution du programme. Cela permet le typage faible et la nature dynamique du langage.

Typage faible

Le type des variables n'est pas déclaré dans le programme. Il est déduit automatiquement lors de l'exécution. Le type d'une variable peut donc changer pendant d'exécution du programme.

Objet

A part les types simples : booléen, nombre et chaîne de caractère, tous les autres éléments du langage sont des objets. Un objet est conteneur d'une collection de propriétés. Chaque propriété, comme une variable, possède une étiquette (son nom) et une valeur qui peut-être un type simple (booléen, nombre, chaîne de caractère) ou un autre objet ce qui permet des structures complexes.

Dynamique

On peut à tout moment de sa vie ajouter dynamiquement des propriétés à un objet. C'est à dire ajouter des propriétés non déclarées au préalable dans le programme initial.

Asynchrone

Tous les appels à des entrées/sorties (requêtes ajax, timer, lecture de fichier, détection d'un changement d'état, ...) **sont non bloquants**. Le programme n'attend pas le retour des entrées/sorties pour continuer. Une fonction de rappel est programmée pour être appelée au moment de la fin de l'opération d'entrée/sortie.

Monothread

Javascript n'exécute qu'un programme à la fois. Les fonctions de rappel sont appelées quand toutes les instructions courantes sont terminées.



Le guide de référence sur MDN [\[https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference\]](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference)

Éléments de syntaxe

JavaScript emprunte la plupart des éléments de sa syntaxe à Java, C et C++.

Le Javascript est sensible à la casse (majuscules / minuscules).

Les instructions doivent être séparées par un ; ou par **un saut de ligne**. La bonne pratique est de toujours mettre un point virgule. **Les espaces, tabulations et sauts de ligne sont des caractères blancs** (ils sont ignorés).

Déclaration de variables

Toutes les variables doivent être déclarées la première fois avec un des mots clés **var**, **let** ou **const**.

var

Le mot clé **var** définit **une variable de portée globale** ou **de portée locale dans une fonction**.

let

Le mot clé **let** définit une variable **de portée locale dans une structure** (dans un if, while, une fonction)

const

Le mot-clé **const** définit une variable en lecture seule (une constante). Sa portée est similaire à **var**.



Déclarer des variables

- 1 Installer l'extension Live Server dans Visual Studio Code
- 2 Créer un dossier (projet) dans Visual Studio Code
- 3 Créer un fichier index.html dans le projet

```
1 <html>
2   <head>
3     <title>Les bases de Javascript</title>
4   </head>
5   <body>
6     <h1>Javascript : les bases, ouvrez la console avec F12</h1>
7     <script src="base.js"></script>
8   </body>
9 </html>
```

- 4 Créer un fichier base.js

base.js

```
1 var hw="Bonjour monde !";
```

```
2 console.log(hw);
```

5 Faire un clic droit sur le fichier index.html et cliquer sur  Open with live server. Dans le navigateur, ouvrez la console avec  F12.

Portée globale avec var

```
1 function testPortee() {  
2   console.log("valeur de hw",hw);  
3 }  
4  
5 var hw="Bonjour monde !";  
6 console.log(hw);  
7 testPortee();
```



La variable est définie dans la fonction !

Portée locale dans une fonction avec var

```
1 function testPortee() {  
2   var hw="Bonjour monde !";  
3   console.log(hw);  
4 }  
5  
6 testPortee();  
7 console.log("Valeur de hw", hw);
```



La variable n'est définie que dans la fonction au moment de l'appel.

Portée dans une structure avec let

```
1 function testPortee(lang) {  
2   if (lang=='fr') {  
3     let hw="Bonjour monde !";  
4     console.log(hw);  
5   } else {  
6     let hw="Hello world !";  
7     console.log(hw);  
8   }  
9   console.log(hw);  
10 }  
11  
12 testPortee('fr');
```



La variable hw n'est visible que dans la structure où elle est définie (ici if ou else).

Définition de constante

```
1 function testPortee() {  
2   console.log(hw);  
3   hw="Hello world !";  
4 }
```

```
5  
6 const hw="Bonjour monde !";  
7 testPortee();
```



La variable hw ne peut pas être assignée deux fois !

Les objets en Javascript

A part les types de base booléen, nombre et texte, toutes les structures du langage sont des objets.

Un objet est un conteneur de propriétés nommées. A chaque propriété on associe une valeur ou un autre objet. Des propriétés peuvent être ajoutées à tout moment sur n'importe quel objet.

Un objet est toujours affecté à au moins une variable ou à au moins une propriété d'un autre objet.



Les tableaux et les fonctions sont des objets.

Syntaxe pour créer un objet vide

On peut créer un objet vide de deux manières :

- `var o = new Object() ;`
- `var o = {} ;`



Les accolades représentent un objet.

Syntaxe des propriétés

Les propriétés sont manipulées de trois façon différentes :

- avec le caractère `.`
- avec la notation `[]`
- avec la notation **Json**

👁 Création d'une nouvelle propriété message (.)

```
o.message="Bonjour Monde !"
```

Dans ce cas, le nom de la propriété est directement écrite comme une variable. **Elle est ajoutée après la création de l'objet.**

👁 Création d'une nouvelle propriété message ([])

```
o["message"]="Bonjour monde !"
```

Dans ce cas, le nom de la propriété est une chaîne de caractère et peut contenir des caractères

interdits dans le nom d'une variable. **Elle est ajoutée après la création de l'objet.**

👁 Création d'une nouvelle propriété message (Json)

```
1 var o = {  
2   message : "Bonjour monde !"  
3 }
```

Dans ce cas, **les propriétés sont déclarées au moment de la création de l'objet.**

Objets imbriqués

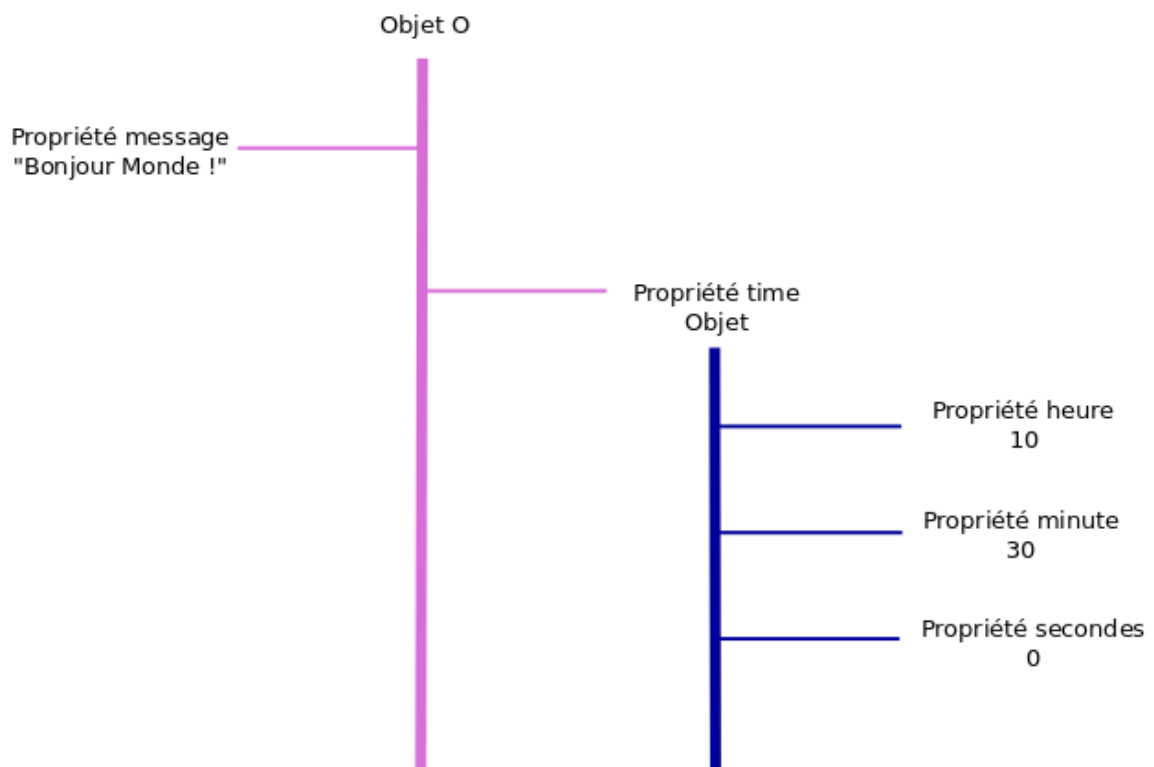
On peut affecter un objet à une propriété comme à une variable.

👁 Affectation d'une nouvelle propriété objet après création

```
1 o.time = {  
2   heure : 10,  
3   minute : 30,  
4   seconde : 0  
5 }
```

👁 à la création en Json

```
1 var o = {  
2   message : "Bonjour monde !",  
3   time : {  
4     heure : 10,  
5     minute : 30,  
6     secondes : 0  
7   }  
8 }
```



Appelle d'une propriété d'un objet imbriqué

Il suffit d'enchaîner les `.` ou les `[]`



```
console.log(o.time.heure) ;  
console.log(o["time"]["heure"]);  
console.log(o.time["heure"]);
```



Créez un objet `Personne` qui vous décrit : nom, prénom, formation. Détaillez les propriétés de la formation (niveau, intitulé, école, ...). Affichez l'objet avec `console.log`.

Les objets natifs de Javascript

L'objet Function

Les fonctions en Javascript sont des objets à part entière.

Il y a plusieurs manière de créer un objet fonction.

Fonction nommée

```
1 function afficheNom(prenom, nom) {  
2   console.log(prenom, nom);  
3 }
```

Crée un objet `Function` et l'affecte à la variable `afficheNom`

Fonction anonyme

```
1 var afficheNom=function(prenom, nom) {  
2   console.log(prenom, nom);  
3 }
```

Crée un objet `Function` et l'affecte à la variable `afficheNom`, la fonction est dite **anonyme**.

Fonction fléchée

```
1 var afficheNom=(prenom, nom)=>{  
2   console.log(prenom, nom);  
3 }
```

Crée un objet pseudo `Function` et l'affecte à la variable `afficheNom`, la fonction est dite **fléchée**.
Ce n'est pas tout à fait la même chose qu'une fonction anonyme (voir plus tard)

Appel d'une fonction

Pour appeler une fonction (c'est à dire l'exécuter), il suffit d'appeler son nom suivi des `()` avec les arguments éventuels.



```
afficheNom("Florent", "TETART") ;
```



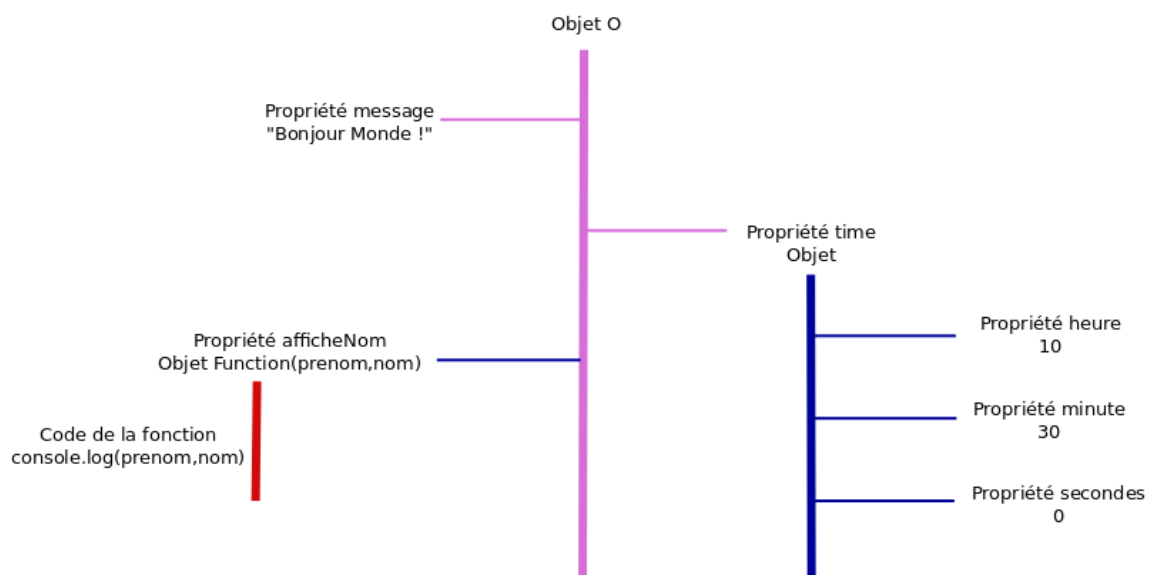
Une fonction peut-être déclarée dans une autre fonction, cela crée une variable locale qui contient l'objet Function associé. Elle ne sera donc pas visible dans l'espace global.



Avec la notation anonyme, on peut **affecter une fonction à une propriété d'un objet**.



```
1 var o = {
2   message : "Bonjour monde !",
3   time : {
4     heure : 10,
5     minute : 30,
6     secondes : 0
7   },
8   afficheNom : function(prenom, nom) {
9     console.log(prenom, nom);
10  }
11 }
```



On appelle la fonction en utilisant la propriété de l'objet.

```
o.afficheNom("Florent", "TETART") ;
```

Le mot clé this

Le mot clé **this** représente le contexte d'appel de la fonction. C'est à dire l'objet associé à l'exécution de la fonction.

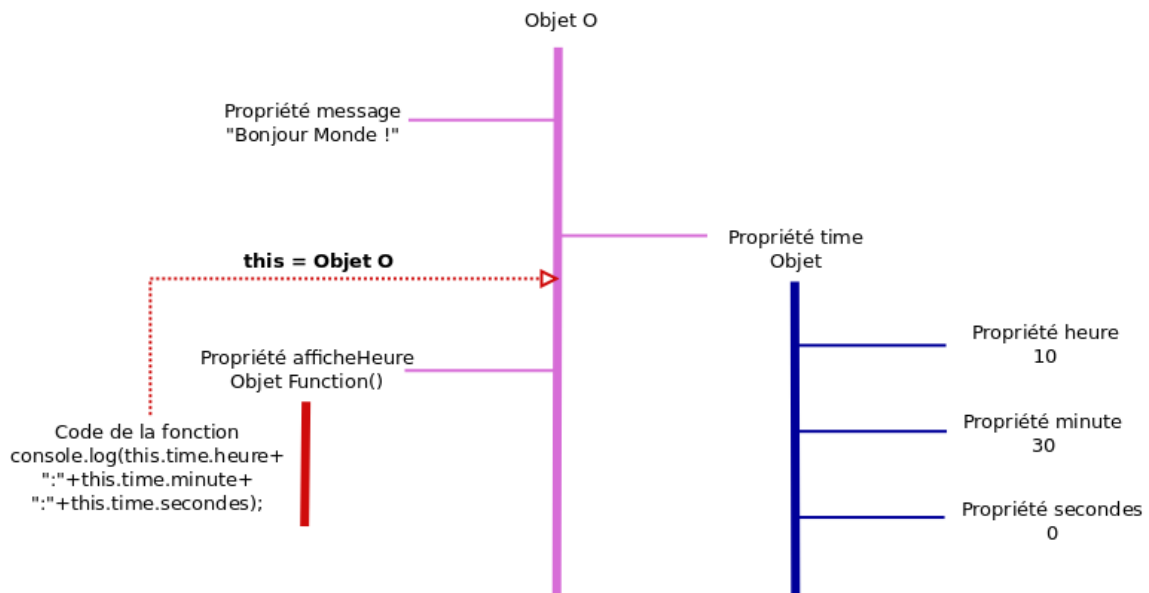


Si la fonction est appelée avec le nom de l'objet auquel elle appartient, **this** vaut cet objet.



```
1 var o = {
2   message : "Bonjour monde !",
3   time : {
4     heure : 10,
5     minute : 30,
6     secondes : 0
7   },
8   afficheHeure : function() {
9     console.log(this.time.heure+":"+this.time.minute+":"+this.time.secondes);
10  }
11 }
```

Si on appelle `o.afficheHeure()` ; **this** représente l'objet **o**. On parle alors de méthode de l'objet **o**.



Ajoutez une méthode `afficheInfos()` à votre objet `Personne` qui affiche votre nom, prénom et formation.

L'objet Array : manipuler des tableaux

Pour créer un objet Array, c'est à dire un tableau, deux syntaxes sont possibles.

Instanciation d'un tableau

```
var t = new Array("bonjour", 10, 24) ;
```

👁 Déclaration avec les crochets

```
| var t = ["bonjour", 10, 24] ;
```



Un tableau est une collection de valeurs auxquelles on accède par l'indice numérique de l'ordre dans le tableau. Pour accéder à un élément, on utilise la notation crochet. Les indices commencent à 0.



```
| console.log( t[2] ) ;  
| Affiche 24.
```

Méthodes de l'objet Array

L'objet Array possède des méthodes pour manipuler le tableau.

push()

La méthode push(valeur) permet d'ajouter un élément en dernier au tableau.



```
| t.push("monde") ;  
| console.log( t[3] ) affiche monde
```

L'objet Array sur MDN [https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Array]

Les objets natifs du navigateur

window

L'objet window représente la fenêtre du navigateur. C'est aussi l'objet qui contient l'espace global de Javascript.

Toute variable ou fonction déclarée dans l'espace global est une réalité une propriété de l'objet window.



```
| var nom = "CESR" ; est la même chose que window.nom="CESR" ;
```



```
| function plus(a,b) { return a+b ; } est la même chose que window.plus=function  
| (a,b) { return a+b ; }
```

document

L'objet document représente la page HTML en cours d'affichage dans le navigateur. C'est en manipulant les propriétés de l'objet document que l'on agit sur la page.



```
document.body.style.background="#FF0000" ;
```

HTMLElement

L'objet HTMLElement représente un élément (tag) de la page HTML en cours d'affichage.



```
h=document.getElementsByTagName("h1") ; //revoie un tableau de
HTMLElement
h[0].innerHTML="Bonjour CESR" ;
```

console

L'objet console représente l'environnement de débogage (acessible par F12).



```
console.log("CESR") ;
```

history

L'objet history représente l'historique de navigation.



```
history.go(-1) ;
revient en arrière d'une page dans l'historique.
```

Instanciation d'objet - prototype

Pour chaque objet que l'on doit créer, il faut déclarer les propriétés et les méthodes (fonctions affectées à des propriétés).

Il existe en Javascript un mécanisme qui permet de ne déclarer qu'une seule fois les méthodes pour des objets de même nature, c'est à dire d'une même classe.

Ce mécanisme combine l'instanciation et la notion de prototype.



Le principe est de déclarer une fonction qui représente la classe d'objet.

Dans la fonction on crée les propriétés qui seront affectées à l'instanciation.

Ensuite on affecte à la propriété `prototype` de l'objet fonction un objet qui contient les méthodes communes à toutes les instances.

L'instanciation se fait avec le mot clé `new`.

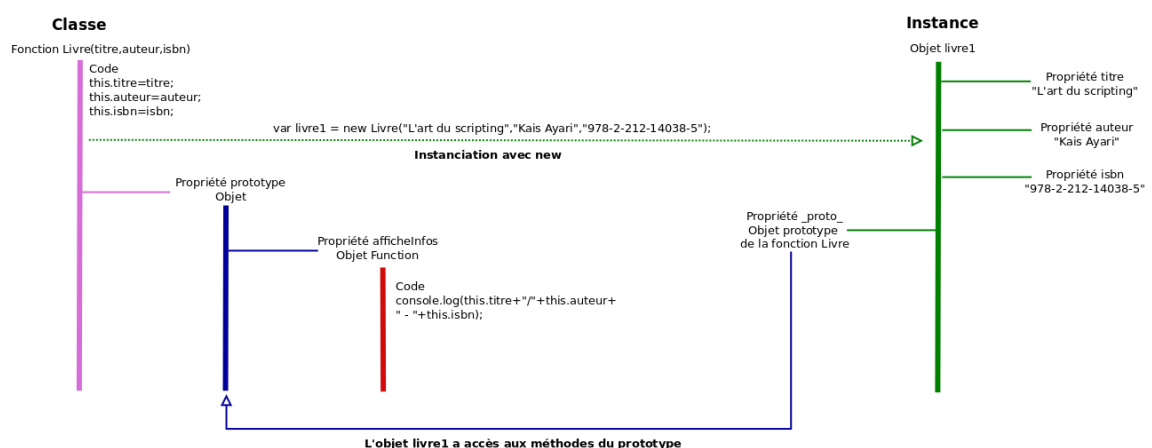


Le mot clé `new` crée un objet vide et l'associe à `this` puis exécute la fonction d'instanciation. La fonction d'instanciation crée les propriétés sur cet objet vide. Enfin, les méthodes dans l'objet `prototype` de la fonction sont rattachées au nouvel objet instance.

Une classe Livre

```
1 function Livre(titre, auteur, isbn) { //Fonction d'instanciation
2   this.titre=titre;
3   this.auteur=auteur;
4   this.isbn=isbn;
5 }
6
7 //Prototype
8 Livre.prototype={
9   afficheInfos : function() {
10    console.log(this.titre+ ' / ' +this.auteur+ ' - ' +this.isbn);
11  }
12 }
13
14 //Instanciation
15 var livre1 = new Livre("L'art du scripting", "Kais Ayari", "978-2-212-14038-5");
16 livre1.afficheInfos();
```

Affiche L'art du scripting / Kais Ayari - 978-2-212-14038-5.



Chaque instance a donc les mêmes propriétés **avec des valeurs différentes** et partage les même méthodes via le prototype dans la propriété `_proto_`.



Créez une classe `Personne` instanciable.



Il existe une syntaxe raccourcie pour tout déclarer en une fois. Mais le mécanisme reste le même.

```
1 class Livre {
2   constructor(titre, auteur, isbn) { //Fonction d'instanciation
3     //Fonction d'instanciation
4     this.titre=titre;
5     this.auteur=auteur;
6     this.isbn=isbn;
7   }
8
9   afficheInfos() { //Méthode du prototype
10    console.log(this.titre+' / '+this.auteur+' - '+this.isbn);
11  }
12 }
13
14 //Instanciation
15 var livre1 = new Livre("L'art du scripting", "Kais Ayari", "978-2-212-14038-5");
16 livre1.afficheInfos();
```

Asynchronisme de Javascript

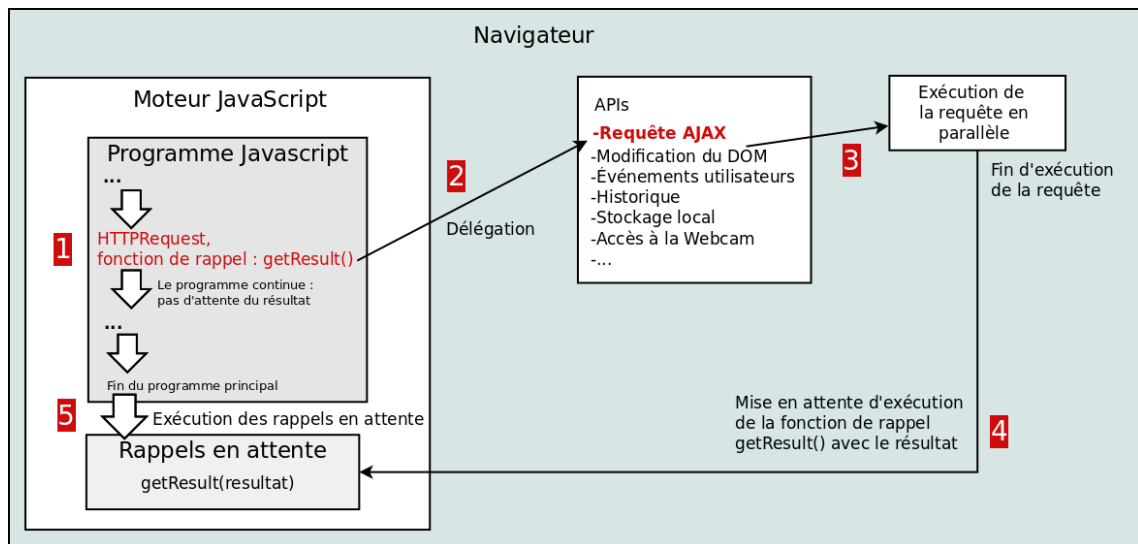
Lorsque le Javascript exécute une opération d'entrée / sortie (par exemple une requête Ajax), il délègue l'exécution de cette opération au système dans lequel le JavaScript s'exécute (le navigateur par exemple) et le moteur Javascript continue à exécuter les instructions suivantes.



Le moteur du Javascript n'attend pas la fin de l'opération de l'entrée / sortie avant de continuer. Autrement dit, le résultat de l'entrée / sortie est traité quand il est disponible plus tard : **de manière asynchrone avec le programme principal.**

Fonction de rappel

A la place d'un traitement synchrone, une fonction de rappel doit être fournie. Quand l'opération d'entrée sortie se termine, la fonction de rappel est mise en attente d'exécution. Quand le Javascript a fini d'exécuter les instructions du programme principal, si une fonction de rappel est en attente elle est exécutée.



Le Javascript est asynchrone pour ne pas bloquer le navigateur quand on attend un résultat d'une entrée / sortie !

Monothread

Le moteur Javascript n'exécute qu'un programme à la fois. Il n'y a pas de parallélisme.

Test asynchrone et monothread

Test asynchrone !

- 1 Créer un fichier index_async.html dans le projet

```
1 <html>
2   <head>
3     <title>Asynchrone</title>
4   </head>
5   <body>
6     <h1>Test asynchrone</h1>
7     <script src="js_async.js"></script>
8   </body>
9 </html>
```

- 2 Créer un fichier js_async.js

```
1 setTimeout(function() {
2   console.log("2 secondes après !");
3 }, 2000);
4 console.log("suite du programme !");
```

Test monothread !

- 1 Modifier le fichier js_async.js

```
1 setTimeout(function() {
```

```

2 console.log("2 secondes après !");
3 let i=0;
4 //On perd du temps !
5 while (i<100000) {
6     i++;
7 }
8 },2000);
9 setTimeout(function() {
10     console.log("3 secondes après !");
11 },3000);
12 console.log("suite du programme !");

```

2 Augmentez le compteur !

Multipliez par 10 le nombre de tours dans la boucle while jusqu'à ce que le message 3 secondes n'apparaisse plus au bout de 3s !!



Dans toutes les fonctions de rappel, `this` représente l'objet qui a déclenché le rappel (ici le timer). Vous pouvez le vérifier en faisant un `console.log(this)`.



Lorsque l'on va utiliser une méthode d'un objet comme fonction de rappel d'un appel asynchrone, `this` ne vaut pas l'objet qui porte la méthode mais l'objet qui a généré le rappel.

Pour conserver l'objet d'origine dans `this` il faut utiliser les fonctions fléchées.

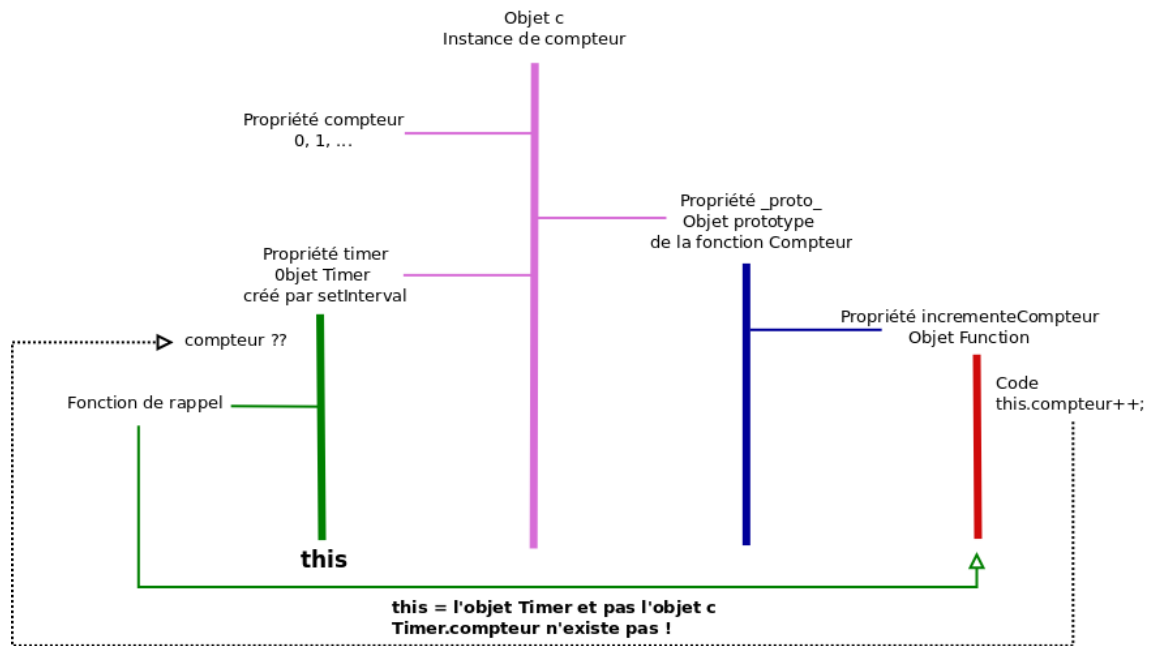


```

1 class Compteur {
2     constructor(duree) {
3         this.duree=duree;
4         this.compteur=0;
5         this.timer=setInterval(this.incrementeCompteur,this.duree); //Ne marche
6     }
7
8     incrementeCompteur() {
9         this.compteur+=1;
10        console.log(this.compteur);
11    }
12
13    stopCompteur() {
14        clearInterval(this.timer);
15    }
16 }
17
18 var c=new Compteur(1000);

```

`this.incrementCompteur` est la fonction passée à `setInterval` qui sera rappelée tous les `duree` en ms. Au moment du rappel, dans la fonction `incrementeCompteur`, `this` vaut l'objet timer qui a été créé par `setInterval` pour faire le rappel et non l'instance de `Compteur`.

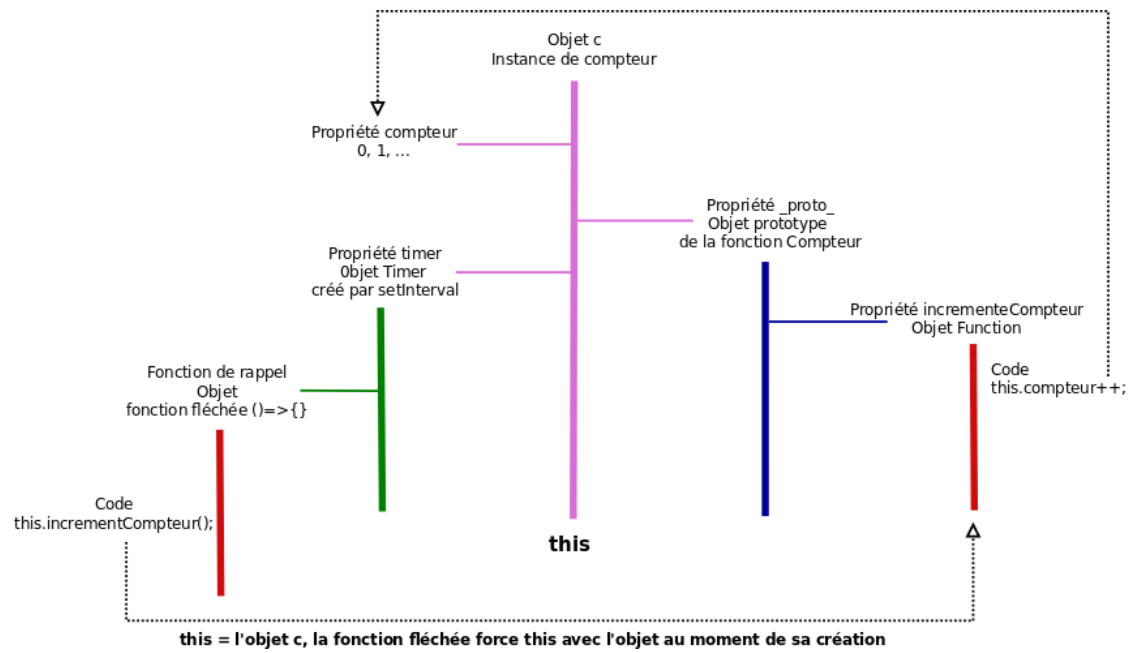


```

1 class Compteur {
2   constructor(duree) {
3     this.duree=duree;
4     this.compteur=0;
5     this.timer=setInterval(()=>{ this.incrementeCompteur() },this.duree);
6     //Marche !
7   }
8   incrementeCompteur() {
9     this.compteur+=1;
10    console.log(this.compteur);
11  }
12
13  stopCompteur() {
14    clearInterval(this.timer);
15  }
16 }
17
18 var t
19 var c=new Compteur(1000);

```

La fonction fléchée `()=>{ this.incrementeCompteur() }` est la fonction qui sera appelée par `setInterval` tous les `duree` en ms. Au moment du rappel, la fonction fléchée conserve dans `this` la valeur qu'il avait au moment de la création de la fonction fléchée, soit l'instance de `Compteur`.



Dans tous les rappels asynchrones d'une méthode d'un objet, pour conserver l'instance d'origine, il faut utiliser une fonction fléchée.

2. Le DOM : Document Object Model

En Javascript, la page HTML courante du navigateur est modélisée par une série d'objets hiérarchisés. Cette modélisation est normalisée et s'appelle **Document Object Model** ou **DOM**.

Document

Le modèle est composé d'un objet document qui représente l'ensemble de la page HTML courante.

Nœuds

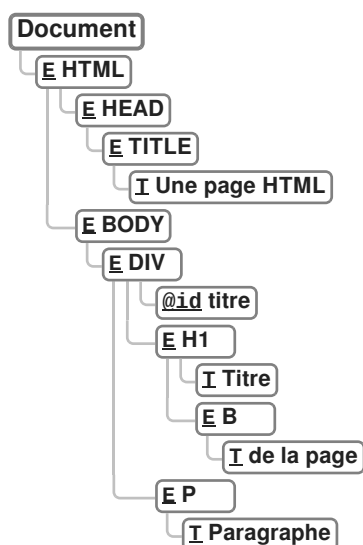
Le document contient un ensemble d'objets nœuds hiérarchisés qui représentent les balises, attributs et textes présents dans le code HTML.

Les nœuds sont de plusieurs type :

- Type Élément (notation E) : une balise HTML
- Type Attribut (notation @) : un attribut d'une balise
- Type Texte (notation I) : un texte à l'intérieur d'une balise HTML

👁 Page HTML et son arbre DOM

```
1 <html>
2   <head>
3     <title>Une page HTML</title>
4   </head>
5   <body>
6     <div id='titre'>
7       <h1>Titre <b>de la page</b></h1>
8       <p>Paragraphe.</p>
9     </div>
10  </body>
11 </html>
```



Propriétés et méthodes de l'objet document

Méthodes	Description	Exemple
document.getElementById('identifiant')	Renvoie un objet nœud de type Élément correspondant à la balise portant l'attribut id='identifiant'	var divTitre=document.getElementById('titre') : renvoie l'objet ElementHTML DIV
document.getElementsByTagName('nom de balise')	Renvoie un tableau de nœuds de type Élément correspondant à toutes les balises de même nom dans le document (quelque soit leur profondeur)	document.getElementsByTagName('p') : renvoie [ElementHTML P]
document.createElement('nom de balise')	Crée un nouveau nœud de type Élément (une balise HTML) portant le nom passé en paramètre.	var p2 = document.createElement('p') : renvoie un nouvel objet ElementHTML P
document.createTextNode('texte')	Crée un nouveau nœud de type Texte portant le texte passé en paramètre.	var t2 = document.createTextNode('Un deuxième paragraphe') renvoie un nouvel objet Text Un deuxième paragraphe
Propriétés	Description	Exemple
documentElement	Le nœud racine du document (autrement dit l'objet correspondant à la balise html)	document.documentElement renvoie l'objet ElementHTML HTML
body	Le nœud body de la page HTML.	document.body renvoie l'objet ElementHTML BODY

Propriétés et méthodes des objets Node (objet ancêtre de tous les objets nœuds)

Méthodes	Description	Exemple
appendChild(objet nœud)	Ajoute un nœud enfant au nœud courant (une sous balise ou un nœud texte)	divTitre.appendChild(p2) ;
removeChild(objet nœud)	Supprime de la liste des enfants l'objet nœud.	divTitre.removeChild(p2) ;
hasChildNodes()	Renvoie vrai ou faux selon que le nœud a des enfants ou pas	divTitre.hasChildNodes() : true
setAttribute('nom attribut', 'valeur')	Ajoute ou modifie un attribut sur un nœud de type élément.	p2.setAttribute('id', 'p2') ;
getAttribute("nom d'attribut")	Récupère la valeur d'un attribut sur un nœud de type élément.	p2.getAttribute('id') ;
Propriétés	Description	Exemple
nodeType	Type de nœud : ● 1 : Élément ● 3 : Nœud texte	divTitre.nodeType -> 1
nodeName	Nom du nœud pour les nœuds de type Élément	divTitre.nodeName -> DIV
nodeValue	Le texte d'un nœud Text	t2.nodeValue -> 'texte'
firstChild	Renvoie le premier nœud enfant du nœud Élément	h1.firstChild -> T Titre
parentNode	Renvoie le nœud parent d'un élément ou d'un nœud texte	p2.parentNode -> E DIV@id='titre'
children	Renvoie un tableau des éléments enfants (pas les nœuds texte) de l'élément.	divTitre.children -> [E H1, E P]
innerHTML	Le contenu sous forme HTML d'un nœud élément	divTitre.innerHTML -> ' Titre de la page Paragraphe. '
id	l'identifiant unique de l'élément	divTitre.id -> titre
style	L'objet style associé au nœud élément	divTitre.style.background = '#888888' ;



Ajout d'un paragraphe au div titre :

```
1 var divTitre = document.getElementById('titre');
```

```
2 var p2 = document.createElement('p');  
3 var t2 = document.createTextNode('Un deuxième paragraphe');  
4 p2.appendChild(t2);  
5 divTitre.appendChild(p2);  
6  
7 console.log(divTitre.innerHTML);
```



Ce code renvoie :

```
1 <h1>Titre <b>de la page</b></h1>  
2 <p>Paragraphe.</p>  
3 <p>Un deuxième paragraphe</p>
```

3. Les événements

Le navigateur gère un certain nombre d'événements que l'on peut capter en javascript.

Les événements sont captables sur tous les éléments du DOM ainsi que sur l'objet document et l'objet window

Les événements ont un nom normalisé dans tous les navigateurs

Nom	Description
click	Un click sur un élément
mousedown	Un appui sur l'un des boutons de la souris
mouseup	Un lâché sur l'un des boutons de la souris
mousemove	Un déplacement de la souris
keypress	Une touche frappée au clavier
keydown	Une touche enfoncée au clavier
keyup	Une touche relâchée au clavier
change	Un changement dans la valeur d'un champ de formulaire
submit	Au moment de la soumission d'un formulaire
load	Après le chargement complet de la page

Liste de tous les événements de Firefox : <https://developer.mozilla.org/fr/docs/Web/Events>

Captation simple

Les éléments ont des propriétés qui permettent d'associer un objet fonction à un événement sur cet élément. Les propriétés dépendent de l'élément.



Les propriétés portent le préfixe **on**



```
<button id='bouton1'>Cliquez moi</button>
```



```
document.getElementById('bouton1').onclick=function(e) {  
  console.log(e) ;  
}
```

```
| }
```

Une fonction de rappel d'un événement prend en argument un objet de type Event.

L'objet Event possède des propriétés qui dépendent de l'événement.

Modèle complet du gestionnaire d'événement

Les événements sont diffusés à partir de la fenêtre window en deux phases consécutives.

La phase capture

L'événement part de l'objet window et est ensuite transmis à document puis à tous les éléments enfants jusqu'à la cible de l'événement.

La phase bubbling

Après la phase capture, l'événement repart de l'élément cible et est transmis à l'ensemble des éléments parents.



On peut capter un événement dans l'une ou l'autre des phases. Si un événement est capté dans la phase capture, on peut l'empêcher de continuer à être propagé aux enfants. À l'inverse dans la phase bubbling on peut l'empêcher d'être transmis aux parents.



Le captage simple par les méthodes **on** se fait toujours dans la phase bubbling. C'est donc la méthode de l'élément cible qui sera appelée en premier.



```
1 <div id='d1'>
2   <div id='d2'>
3     <button id='b1'>Cliquez-moi !</button>
4   </div>
5 </div>
6 <script>
7   function click(e) {
8     console.log(e);
9   }
10  document.getElementById('d1').onclick=click;
11  document.getElementById('d2').onclick=click;
12  document.getElementById('b1').onclick=click;
13 </script>
```

stopPropagation()

Pour empêcher un événement de se propager, dans la fonction de rappel il suffit d'appeler la méthode stopPropagation() de l'événement.



```

1 <div id='d1'>
2   <div id='d2'>
3     <button id='b1'>Cliquez-moi !</button>
4   </div>
5 </div>
6 <script>
7   function click(e) {
8     console.log(e);
9     e.stopPropagation();
10  }
11  document.getElementById('d1').onclick=click;
12  document.getElementById('d2').onclick=click;
13  document.getElementById('b1').onclick=click;
14 </script>

```

Captage en phase capture et fonctions multiples

Pour capter un événement on peut utiliser une méthode universelle que les méthodes simples : `addEventListener()`.

Cette méthode prend 3 arguments :

- une chaîne de caractère représentant l'événement à capter : ex : 'click'
- la fonction de rappel à exécuter : ex : `click`
- la phase : `true` pour la phase capture ou `false` pour la phase bubbling



Capture d'événement en phase capture

```

1 <div id='d1'>
2   <div id='d2'>
3     <button id='b1'>Cliquez-moi !</button>
4   </div>
5 </div>
6 <script>
7   function click(e) {
8     console.log(e);
9   }
10  document.getElementById('d1').addEventListener('click', click, true);
11  document.getElementById('d2').addEventListener('click', click, true);
12  document.getElementById('b1').addEventListener('click', click, true);
13 </script>

```



Capture d'événement en phase bubbling

```

1 <div id='d1'>
2   <div id='d2'>
3     <button id='b1'>Cliquez-moi !</button>
4   </div>
5 </div>
6 <script>
7   function click(e) {
8     console.log(e);
9   }
10  document.getElementById('d1').addEventListener('click', click, false);

```

```
11 document.getElementById('d2').addEventListener('click',click,false);  
12 document.getElementById('b1').addEventListener('click',click,false);  
13 </script>
```



Certains événements sont gérés aussi par le navigateur, par exemple des raccourcis claviers, le menu contextuel sur click droit ou la sélection de texte (mousedown + mousemove).



On peut empêcher le comportement par défaut par la méthode de l'événement `preventDefault()`

Blocage d'un événement pour toute la page

Pour bloquer un événement sur toute la page, il suffit de le capter en phase capture sur l'objet `window` et d'empêcher sa propagation.

👁 Blocage du menu contextuel pour toute la page

```
1 window.addEventListener('contextmenu',function(e) {  
2     e.preventDefault();  
3 },true) ;  
4
```