De ES5 à ES6, qu'est-ce qui a changé?

Jérémie Amsellem - cours@lp1.eu

Sommaire

I - Premièrement, qu'est-ce qu'ES6?

Pour rappel, ECMAScript est la spécification de langage utilisée (entre autres) par le langage JavaScript.

ES6, ou **ECMAScript 6** (aussi appelé **ECMAScript 2015**) est la version d'ECMAScript actuellement utilisée par les dernières versions de Node.js, Firefox et Google Chrome.

Cette nouvelle version apporte beaucoup de nouveautés concernant les paradigmes d'ECMAScript empruntés de la POO et de la programmation fonctionnelle, notamment la possibilité de créer des classes en JavaScript !

II - Les mots clés let et const

ES6 Introduit deux nouveaux mots-clés permettant de définir des variables et des constantes : **let** et **const**.

let vise à remplacer var par une alternative plus "sûre" pour les développeurs et évitant les débordements involontaires dans des scopes parents.

const permet de définir des constantes, ce qui était auparavant impossible dans le langage.

III - Les fonctions flêchées

Seconde fonctionnalité primordiale :
la possibilité de définir des "arrow functions"
(aussi appelées lambdas dans d'autres langages).

Cela permet de prototyper des fonctions beaucoup plus courtes qu'avec la syntaxe habituelle :

```
function(params) {
  /* logique */
}
```

Les syntaxes existantes sont les suivantes :

```
(params) => {/* logique */}
param => {/* logique */}
param => /* logique */
```

IV - Les chaînes "templatées"

On peut désormais utiliser des templates dans des chaînes de caractères pour interpoler (remplacer dynamiquement) des variables dans du texte.

Pour définir une chaîne de caractères contenant des templates, il suffit d'utiliser des backquotes au lieux des simples et doubles quotes habituels.

On utilisera ensuite la syntaxe ${expression} pour indiquer le contenu à remplacer par une expression JavaScript.

Par exemple :

```
`Today's the ${new Date().toLocaleString()}`
```

V - Les Sets

Des nouvelles structures de données ont également fait leur apparition dans ES6, notamment les très utiles Sets et Maps.

Un **Set** permet de stocker de valeurs **uniques** de tous les types.

Par exemple :

```
const set = new Set()
set.add('test').add(42).add('hello').add({})
set.has(42) /* true */
set.delete(42)
set.has(42) /* false */
set.size /* 3 */
set.clear() /* Vide le Set */
```

VI - Les Maps

Les **Maps** permettent de stocker des variables en clé/valeur (comme un objet JavaScript classique en somme).

En revanche, celles-ci peuvent utiliser n'importe quel type d'objet comme clé.

Également, il est très simple d'obtenir la taille d'une Map, en utilisant l'attribut Map.size, contrairement aux objets JavaScript.

Exemple :

```
const obj = {}
const map = new Map()
map.set('hello', 42)
map.set(obj, true)
map.get(obj) /* true */
map.has('hello') /* true */
map.delete('hello')
map.size /* 1 */
```

VII - Les promesses (ou promises)

Les promesses sont une manière plus explicite et claire de définir un comportement asynchrone.

Une promesse est un objet à usage unique, c'est une portion de code qui sera exécutée de manière asynchrone et renverra dans un laps de temps indéterminé des informations à la fin de cette exécution.

On définit une promesse en instanciant un nouvel objet Promise et en y ajoutant notre comportement.

Une fois notre promesse définie, on peut la déclencher en appelant ses méthodes then et catch.

VIII - For...in

Deux nouveaux mot-clés permettant d'itérer plus facilement sur des tableaux et des objets ont également été ajoutés.

Ce sont les mots-clés **for ...in** et **for ...of**.

- for ... in permet d'itérer sur les clés d'un objet JavaScript :

```
for (const key in {key1: 1, key2: 2}) {
  console.log(key) // key1, key2
}
```

- for ... of permet d'itérer sur les éléments d'un tableau :

```
For (const item of [3, 2, 1]) {
  console.log(item) // 3, 2, 1
}
```

IX - Nouvelles méthodes de la classe String

- **String.includes** : Retourne true si la chaîne passée en paramètre de includes est contenue dans la String
- **String.repeat(n)** : Retourne la chaîne répétée n fois
- **String.startsWith** : Retourne true si la String commence par la chaîne passée en paramètre
- **String.endsWith** : Retourne true si la String termine par la chaîne passée en parametre

X - Nouvelles méthodes des tableaux

- **Array.forEach** : Permet d'itérer sur chacun des éléments d'un tableau
- **Array.filter** : Permet de filtrer un tableau JavaScript
- **Array.map** : Permet de transformer les valeurs d'un tableau
- **Array.reduce** : Permet de réduire un tableau à un seul nombre
- **Array.find** : Permet de trouver une valeur dans un tableau (et non son index)
- **Array.findIndex** : Permet de trouver l'index d'un élément dans un tableau

XI - Default

Comme dans de nombreux autres langages, il est possible de définir des paramètres par défaut dans une fonction dans le cas où ceux-ci ne seraient pas remplis.

Par exemple :

```
function createUser(name, age, location="France") {
  console.log(`${name / ${age} / ${location}`)
}
```

Attention, les paramètres par défaut sont toujours les derniers paramètres d'une fonction !

XII - Rest et Spread

Rest permet de stocker des paramètres en nombre indéfini dans un tableau et ce sans utiliser le mot clé arguments de JavaScript.

Par exemple

```
function test(name, age, ...extra) {
  console.log(extra) // Affiche tout ce qui est passé en plu
}
```

Spread, quand à lui permet d'utiliser un tableau en tant que paramètres pour une fonction.

Par exemple :

```
const array = ['Bob', 'TestMan', 'Paris']
function createUser(firstName, lastName, location) {
}
createUser(...array) // Équivaut à createUser(array[0], ai
```

XIII - Get et set

Autre nouvelle possibilité, cette fois-ci concernant les objets JavaScript, on peut désormais définir simplement nos propres setters et getters sur des attributs :

```
const obj = {
  get size() {
    //calcul de la taille de l'objet
  }
}
obj.size // Recalculé dynamiquement et gardé en mémoire
```