ES6 : boucles, itérations et générateurs

Avec ES6 on a une nouvelle instruction de boucle : **for-of**. On dispose aussi de l'itération. Faisons le point dans ce chapitre.

La boucle for-of

Avec ES5 on a déjà plusieurs possibilités pour faire des boucles :

- for,
- do…while,
- while,
- for…in,
- forEach.

*Que nous apporte de plus **for-of** ?*

L'instruction **for-of** fonctionne sur des objets itérables (**Array**, **Map**, **Set**, **String**) et est destinée à remplacer **for-in** et **forEach**.

La syntaxe est simple :

```
let tableau = [1, 2];
for (let element of tableau) {
    console.log(element);  // 1  2
}
```

*On peut utiliser **break** et **continue** dans cette boucle.*

A l'intérieur de la boucle on dispose de la clé et de la valeur, voici un exemple avec un **map** :

```
let map = new Map([['prenom', 'Pierre'], ['nom', 'Durand']]);
for (let [key, value] of map) {
    console.log(`clé : ${key}, valeur : ${value}`);
}
```

*On utilise ici des choses qu'on a vues dans les précédents chapitres : **map**, destructuration et littéral de gabarit.*

Les itérations

On a ici deux notions importantes :

- Un itérable est une structure de données dont les éléments sont accessibles ,
- un itérateur est un pointeur qui permet de sélectionner un élément d'un l'itérable (avec la méthode **next**).

Quelle sont les structures de données itérables ?

On a essentiellement :

- Array,
- String,
- Map,
- Set.

On peut les considérer comme des sources de données consommables. D'un autre côté on a des consommateurs comme la boucle **for-of** qu'on a vue ci-dessus. Mais on a aussi vu dans ce cours **Array.from**, l'opérateur **…**, les constructeurs **Set** et **Map**.

Vous n'avez pas besoin de créer un itérateurs pour ces collections, JavaScript les a déjà prévues :

- **entries()** : retourne un itérateur pour des clés/valeurs,
- **values()** : retourne un itérateur pour des valeurs,
- **keys()** : retourne un itérateur pour des clés.

Voici un exemple avec un **map** :

```
let map = new Map();
map.set('prenom', 'Pierre');
map.set('nom', 'Durand');
for (let element of map.entries()) {
  console.log(element);
}
for (let element of map.values()) {
  console.log(element);
}
for (let element of map.keys()) {
```

```
  console.log(element);
}
```

Ce qui produit :

```
prenom,Pierre
nom,Durand
Pierre
Durand
prenom
nom
```

*Mais on a pas utilisé ces méthodes quand on a vu la boucle **for-of** au début de ce chapitre !*

C’est exact parce qu’il y a un itérateur par défaut pour simplifier la syntaxe. Pour les tableaux et ensembles c’est **values** et pour les map c’est **entries**.

Les générateurs

Un générateur est une fonction qui peut être mise en pause et relancée et qui sert essentiellement à créer des itérateurs.

Un objet n’est pas itérable alors il faut prévoir un itérateur si on veut parcourir ses propriétés. C’est là qu’un générateur nous permet de faire cela très facilement :

```
function* Proprietes(objet) {
    let proprietes = Reflect.ownKeys(objet);
    for (let propriete of proprietes) {
        yield [propriete, objet[propriete]];
    }
}
let identite = { prenom: 'Pierre', nom: 'Durand' };
for (let [key, value] of Proprietes(identite)) {
    console.log(`${key}: ${value}`);
}
```

Ce qui donne :

```
prenom: Pierre
nom: Durand
```

Remarquez deux choses :

- l’astérisque présent après **function** pour spécifier que c’est un générateur,
- le mot clé **yield** qui renvoie l’élément pointé et met la fonction en pause.

Donc chaque fois qu’on appelle la fonction un nouvel élément est retourné jusqu’à ce qu’il n’y en ait plus, on a créé un itérateur !

En résumé

- ES6 ajoute l’instruction **for-of** pour les structures de données itérables.
- Un itérateur est un pointeur pour un itérable.
- JavaScript expose des itérateurs pour Array, Set, Maps…
- Pour les objets on peut créer un itérable avec un générateur.