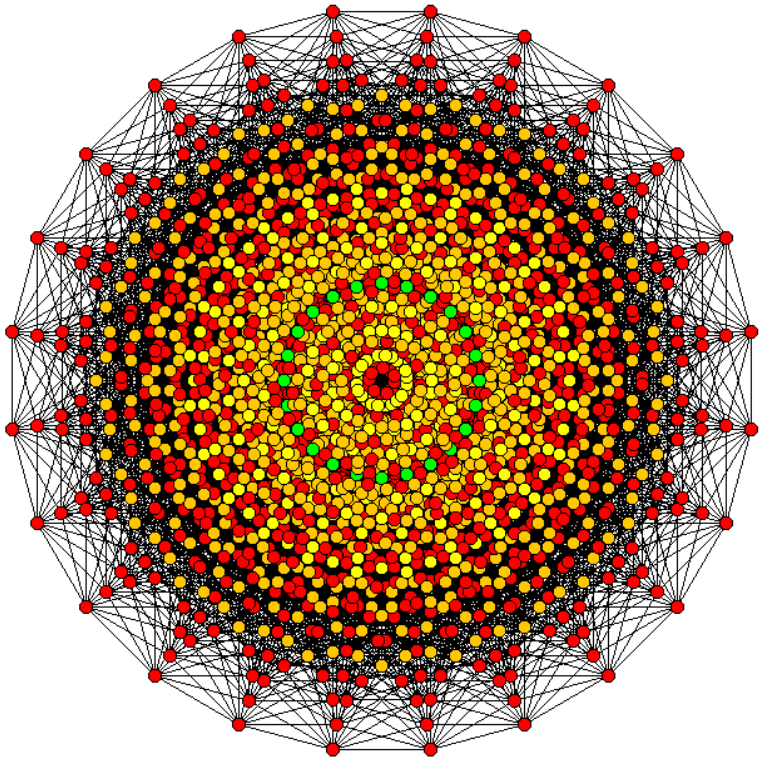


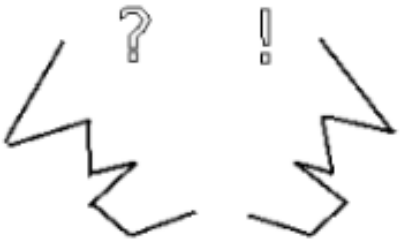
Tableau en TS/JS

O. Capuozzo

Version 1.0, 2020-12-07

Table des matieres

Présentation



Support de cours sur une introduction à la structure de tableau en TS/JS

Tableau et itération

Introduction

`Array` est la structure qui permet de représenter une liste de valeurs.

Par *valeur* on entend les types primitifs et autres (tableau compris).

Déclaration

Deux syntaxes possibles :

La plus courante :

```
let list: number[] = [1, 2, 3];
```

La forme utilisant la syntaxe de type générique `Array<elemType>`

```
let list: Array<number> = [1, 2, 3];
```

Il est fortement recommandé d'utiliser des déclarations de liste **typée** dans le but de mettre en évidence des problèmes de conception. Exemples :

```
let list: number[] = [1, 2, , 3];  // ERROR car `undefined` n'est pas un number
```

```
let list: number[] = [1, 2, NaN, 3];  // OK (`NaN` est un nombre qui n'en n'est pas un...)
```

La methode `Array.of()` permet de créer une nouvelle instance d'objet Array à partir d'un nombre variable d'arguments, quelque soit leur nombre ou leur type.

```
let list: number[] = Array.of(1, 2, 3);
```

Opérations de type non-mutateur

Ce sont les opérations les plus courantes, elles **n'altère pas** la collection sur laquelle elle agit. On les qualifie de ***fonctions pures***.

- Rechercher un élément (par exemple le premier) : `indexOf()`, `lastIndexOf()`
- Opérations permettant de rechercher un sous-ensemble d'éléments (par exemple tous sauf le premier et le dernier) : `slice()`, `filter()`
- Déterminer une information calculée : `reduce()`
- Prédicat : Vérifier une proposition (par exemple "est une liste de nombres impairs") : `every()`, `some()`
- Nouvelle version du tableau : `map()`

Vous trouverez la présentation des fonctions sur les tableaux à cette adresse : Documentation en ligne developer.mozilla : Array

Mutateurs

On appelle *mutateurs* des méthodes qui ont le pouvoir de **modifier l'état** de l'instance du tableau sur lequel elles sont appelées.

À utiliser avec prudence.

Attention à `length`, car contrairement à java, cet attribut est un mutateur ! Il est accessible en écriture (modifie la longueur du tableau)

```
let tab = [1, 2, 3];
tab.length = 2;  // réduit le nombre d'éléments de la liste
console.log(JSON.stringify(tab)); // [1, 2]
```

Ces méthodes modifient le tableau avec lesquelles elles sont appelées. Exemple.

```
let tab = [1, 2, 3];
tab.fill(4);
console.log(JSON.stringify(tab)); // [4, 4, 4]
```

Extraits de quelques unes de ces fonctions :

`Array.prototype.fill()`
: Cette méthode remplie tous les éléments du tableau avec une même valeur, éventuellement entre un indice de début et un indice de fin.

`Array.prototype.pop()`

Cette méthode supprime le dernier élément du tableau et retourne cet élément.

`Array.prototype.push()`
: Cette méthode ajoute un ou plusieurs éléments à la fin du tableau et retourne la nouvelle longueur du tableau.

`Array.prototype.reverse()`
: Cette méthode renverse l'ordre des éléments du tableau - le premier élément devient le dernier, et le dernier devient le premier.

`Array.prototype.shift()`
: Cette méthode supprime le premier élément du tableau et retourne cet élément.

`Array.prototype.sort()`
: Cette méthode trie en place les éléments du tableau et retourne le tableau.

`Array.prototype.splice()`
: Cette méthode permet d'ajouter ou de retirer des éléments du tableau.

`Array.prototype.unshift()`
: Cette méthode permet d'ajouter un ou plusieurs éléments au début du tableau et renvoie la nouvelle longueur du tableau.

`Array.prototype.copyWithin()`
: Cette méthode copie une série d'éléments d'un tableau dans le tableau courant.

Voir des exemples ici : https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Array

Différentes façons de boucler sur un tableau

JavaScript propose plusieurs méthodes d'itération. Nous en exploitons ici les plus populaires.

Plusieurs méthodes utilisent des fonctions comme argument. Ces fonctions sont utilisées afin de traiter, d'une façon ou d'une autre, l'un après l'autre tout élément du tableau.

L'exemple ci-dessous montre **9 façons différentes** (!) de résoudre la question : "*Combien y a-t-il de personnages maîtres référencés dans la liste ?*".

Listing 1. exemple en TypeScript

```
// Déclaration d'un type à partir d'une énumération de valeurs possibles.
// Très pratique et impossible à faire avec une Interface.
type Force =
  typeof Personnage.FORCE_ZERO
  | typeof Personnage.FORCE_PETITE
  | typeof Personnage.FORCE_MOYENNE
  | typeof Personnage.FORCE_GRANDE;

class Personnage {
 static readonly FORCE_PETITE = 20;
 static readonly FORCE_MOYENNE = 50;
```

```
 static readonly FORCE_GRANDE = 80;
 static readonly FORCE_ZERO = 0;

 private force : Force;

public constructor(forceInitiale : Force = Personnage.FORCE_ZERO){
  this.force = forceInitiale;
}
public getForce() : Force {
  return this.force;
}

// on remarquera ici le typage fort pratique du paramètre !
// Ainsi le développeur n'a pas à tester de la validitié ou non de la
// valeur du paramètre reçu, car forcément acceptable.
public setForce(force : Force) : void {
  if  (this.force !== force) {
     this.force = force;
  }
}

public isMaster()  : boolean {
  return this.force === Personnage.FORCE_GRANDE;
}

}

console.clear();

let p1: Personnage = new Personnage(80);
let p2: Personnage = new Personnage(20);
let p3: Personnage = new Personnage(80);

let ptab: Personnage[] = [p1, p2, p3];
// ou  Array.of(p1, p2, p3);

//console.log(ptab.length); // 3

///// for classique

let cpt : number = 0;
for (let i=0; i < ptab.length; i++) {
   if (ptab[i].isMaster()) cpt++;
}
console.log("Version classique : Le nombre de maîtres est " + cpt);

///// for ... of

cpt = 0;
for (const p of ptab) {
  if (p.isMaster()) cpt++;
```

```
}
console.log("Version for .. of   : Le nombre de maîtres est " + cpt);

///// foreach V1 : la variable p représente l'élément courant du tableau,
// avec fonction anonyme ayant comme paramètre un élément du tableau

cpt = 0;
ptab.forEach(function (p) {if (p.isMaster()) cpt++;});
console.log("Version foreach V1 : Le nombre de maîtres est " + cpt);

///// foreach V2 avec fonction fléchée (arrow function) -- écriture plus concise

cpt = 0;
ptab.forEach(p => {if (p.isMaster()) cpt++;});
console.log("Version foreach V2  : Le nombre de maîtres est " + cpt);

// autre version, pas forcément mieux...
cpt = 0;
ptab.forEach(p => cpt = p.isMaster() ? cpt+1: cpt);
console.log("Version foreach V2Bis  : Le nombre de maîtres est " + cpt);
```

Le problème avec les différentes versions présentées ci-dessus, est qu'elles dépendent d'une variable dont la portée dépasse celle de la structure itérative (une portée plus "globale"). En effet, ces implémentations **modifient la variable `cpt` déclarée en dehors de leur contexte**. Donc leur résultat dépendra d'une part de la liste sur laquelle elles travaillent (ça c'est normal), **et dautre part** de la valeur de la variable `cpt` au moment de l'appel. Aïe... Peut mieux faire !

C'est ce que l'on appelle un ***effet de bord*** : leur usage change l'état du sytème qui les a activés (modificiation d'une variable globale), mettant à mal l'aspect prédictif d'un programme.

Afin d'éviter les effets de bords, nous allons nous intéresser à deux autres solutions : `filter` et `reduce`.

Listing 2. exemple filter

```
///// filter : méthode de Array

// filter : retourne un tableau composé des éléments du tableau courant (ici ptab)
// dont les éléments vérifient la proposition passée en argument de filter.
// Il ne nous reste plus qu'à interroger le nombre d'éléments de ce tableau
cpt = ptab.filter(p => p.isMaster()).length;
console.log("Version filter  : Le nombre de maîtres est " + cpt);
```

Critique : Solution élégante (claire et concise), mais qui passe par la création d'une nouvelle instance de tableau.

La solution suivante (**reduce**) ne produit pas de tableau intermédiaire.

Listing 3. exemple reduce

```
///// reduce : méthode de Array

/*
reduce : prend en argument une fonction « reducer » dont l'objectif est de retourner
une valeur, sur la base d'un algorithme dont elle a le secret.

Le plus souvent, cet algorithme se base sur les 2 premiers paramètres de la fonction
reduce :

- premier paramètre (appelé couramment accumulateur) correspond à la valeur retournée
par le précédent appel à la fonction sur l'élément précédent (à l'étape initiale ce
sera  soit la valeur du premier élément de la liste, soit la valeur par défaut passée
au lancement de reduce, comme c'est le cas ici.)

- deuxième paramètre représente l'élément en cours de traitement de la liste

Remarque : la fonction «reducer» peut recevoir jusqu'à 4 paramètres (index de l'él
ément courant et référence au tableau en question)

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Array/re
duce
 */

// reduce : prend en arguent une fonction qui effectue une "réduction",
// ici met à jour un accumulateur conditionné par le fait que
// le personnage courant est un maître.
const reducerMastersCount = function (accumulator: number, p: Personnage): number {
  return p.isMaster() ? accumulator+1 : accumulator
};
cpt = ptab.reduce(reducerMastersCount, 0);
console.log("Version reduce V1 : Le nombre de maîtres est " + cpt);

// La syntaxe condensée de fonction anonyme (arrow function) est
// possible ici car le corps de la fonction peut se résumer
// en une seule expression (on en profite pour renommer un parametre)
const reducerMastersCountV2 = (nbMaster: number, p: Personnage): number => p.
isMaster() ? nbMaster+1 : nbMaster;
cpt = ptab.reduce(reducerMastersCountV2, 0);
console.log("Version reduce V2  : Le nombre de maîtres est " + cpt);
```

La méthode `reduce` est une fonction efficace (en terme de consommation mémoire), très générique et riche en paramètres. Le choix de nommage de ses paramètres doit être soigné pour gagner en compréhension.

Il est possible de mettre à jour un `array`, ou un objet, passé en premier argument de `reduce`. Le code suivant est équivalent à la fonction `filter`, mais réalisé avec `reduce`, **juste pour la démonstration**, car autant utiliser `filter`.

Listing 4. pour l'exemple - filter vs reduce

```
const reducerVFilter =
  function (pfilters : Personnage[], p : Personnage) : Personnage[] {
    if (p.isMaster()) {
      pfilters.push(p);
    }
    return pfilters
  };

cpt = ptab.reduce(reducerVFilter, []).length;
console.log("Version reduce-filter : Le nombre de maîtres est " + cpt);
```

On remarquera la valeur initiale (tableau vide), au lancement de `reduce`.