JS 09 – Tableaux – Objets Map & Set

En JavaScript, les développeurs passent souvent beaucoup de temps à décider de la structure de données correcte à utiliser.   
En effet, le choix de la bonne structure de données peut faciliter la manipulation ultérieure de ces données, ce qui permet de gagner du temps et de rendre le code plus facile à comprendre.

Les deux structures de données prédominantes pour le stockage des collections de données sont Objets et Tableaux (un type d’objet).

Les développeurs utilisent des Objets pour stocker les paires clé/valeur et des Tableaux pour stocker les listes indexées.   
Toutefois, pour donner plus de flexibilité aux développeurs, la spécification ECMAScript 2015 a introduit deux nouveaux types d’objets itératifs :

* les *objets Map*, qui sont des collections ordonnées de paires clé/valeur.
* les *objets Set*, qui sont des collections de valeurs uniques.

Dans cet article, vous passerez en revue les objets Map et Set, ce qui les rend similaires ou différents des Objets et des Tableaux, les propriétés et méthodes dont ils disposent, et des exemples d’utilisations pratiques.

# Objets Map

Un objet Map est une collection de paires clé/valeur qui peut utiliser n’importe quel type de données comme clé et peut maintenir l’ordre de ses entrées. Les objets Map comportent des éléments d’Objets (une collection unique de paires clé/valeur) et de Tableaux (une collection ordonnée), mais sont plus proches des Objets sur le plan conceptuel.   
En effet, bien que la taille et l’ordre des entrées soient préservés comme un Tableau, les entrées elles-mêmes sont des paires clé/valeur comme les Objets.

Les objets Map peuvent être initialisés avec la nouvelle syntaxe Map() :

const map = new Map()

Cela nous donne un objet Map vide :

Output

Map(0) {}

## Ajout de valeurs à un objet Map

Vous pouvez ajouter des valeurs à un objet Map avec la méthode set(). Le premier argument sera la clé, et le second la valeur.

Ce qui suit ajoute trois paires clé/valeur à l’objet Map :

map.set('firstName', 'Luke')

map.set('lastName', 'Skywalker')

map.set('occupation', 'Jedi Knight')

Ici, nous commençons à voir comment les objets Map possèdent à la fois des éléments d’Objets et de Tableaux. Comme dans un Tableau, nous avons une collection indexée à zéro, et nous pouvons également voir combien d’éléments se trouvent par défaut dans l’objet Map. Les objets Map utilisent la syntaxe => pour signifier les paires clé/valeur en tant que key => value :

Output

Map(3)

0: {"firstName" => "Luke"}

1: {"lastName" => "Skywalker"}

2: {"occupation" => "Jedi Knight"}

Cet exemple ressemble à un objet ordinaire avec des clés en chaîne, mais nous pouvons utiliser n’importe quel type de données comme clé avec des objets Map.

Outre la définition manuelle de valeurs sur un objet Map, nous pouvons également initialiser un objet Map avec des valeurs existantes. Pour ce faire, nous utilisons un Tableau des Tableaux contenant deux éléments qui représentent chacun une paire clé/valeur ; ce tableau ressemble à ceci :

[ [ 'key1', 'value1'], ['key2', 'value2'] ]

En utilisant la syntaxe suivante, nous pouvons recréer le même objet Map :

const map = new Map([

['firstName', 'Luke'],

['lastName', 'Skywalker'],

['occupation', 'Jedi Knight'],

])

**Remarque** : cet exemple utilise des *virgules de fin de ligne*, également appelées "*dangling commas"*. Il s’agit d’une pratique de formatage JavaScript dans laquelle le dernier élément d’une série lors de la déclaration d’un ensemble de données comporte une virgule finale. Bien que ce choix de formatage puisse être utilisé pour des différences plus nettes et une manipulation plus facile du code, son utilisation ou non est une question de préférence. Pour plus d’informations sur les virgules de fin de ligne, consultez cet [article sur les virgules de fin de ligne](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Trailing_commas) dans les documents web de MDN.

Soit dit en passant, cette syntaxe est la même que celle qui résulte de l’appel de [Object.entries()](https://www.digitalocean.com/community/tutorials/how-to-use-object-methods-in-javascript" \l "objectentries()) sur un Objet. Cela permet de convertir un Objet en un objet Map, comme le montre le bloc de code suivant :

const luke = {

firstName: 'Luke',

lastName: 'Skywalker',

occupation: 'Jedi Knight',

}

const map = new Map(Object.entries(luke))

Vous pouvez également transformer un objet Map en un Objet ou un Tableau avec une seule ligne de code.

Ce qui suit permet de convertir un objet Map en Objet :

const obj = Object.fromEntries(map)

Il en résultera la valeur suivante de obj :

Output

{firstName: "Luke", lastName: "Skywalker", occupation: "Jedi Knight"}

Maintenant, convertissons un objet Map en Tableau :

const arr = Array.from(map)

Il en résultera le tableau suivant pour arr :

Output

[ ['firstName', 'Luke'],

['lastName', 'Skywalker'],

['occupation', 'Jedi Knight'] ]

## Clés d’objet Map

Les objets Map acceptent tout type de données comme clé, et n’autorisent pas la duplication des valeurs de la clé. Nous pouvons le démontrer en créant un objet Map et en utilisant des valeurs non linéaires comme clés, ainsi qu’en attribuant deux valeurs à la même clé.

Tout d’abord, initialisons une carte avec des clés non ordonnées :

const map = new Map()

map.set('1', 'String one')

map.set(1, 'This will be overwritten')

map.set(1, 'Number one')

map.set(true, 'A Boolean')

Cet exemple remplacera la première clé de 1 par la suivante, et traitera "1" la chaîne et 1 le chiffre comme des clés uniques :

Output

0: {"1" => "String one"}

1: {1 => "Number one"}

2: {true => "A Boolean"}

Bien qu’il soit communément admis qu’un objet JavaScript ordinaire peut déjà traiter des nombres, des booléens et d’autres types de données primitives comme des clés, ce n’est en fait pas le cas, car les objets changent toutes les clés en chaînes de caractères.

À titre d’exemple, initialisez un objet avec une clé numérique et comparez la valeur d’une clé numérique 1 et d’une clé "1" en chaîne :

// Initialize an object with a numerical key

const obj = { 1: 'One' }

// The key is actually a string

obj[1] === obj['1'] // true

C’est pourquoi si vous essayez d’utiliser un objet comme clé, il imprimera la chaîne de caractères Objet à la place.

Par exemple, créez un Objet et utilisez-le ensuite comme clé d’un autre Objet :

// Create an object

const objAsKey = { foo: 'bar' }

// Use this object as the key of another object

const obj = {

[objAsKey]: 'What will happen?'

}

Il en résultera ce qui suit :

Output

{[object Object]: "What will happen?"}

Ce n’est pas le cas avec un objet Map. Essayez de créer un Objet et de le définir comme la clé d’un objet Map :

// Create an object

const objAsKey = { foo: 'bar' }

const map = new Map()

// Set this object as the key of a Map

map.set(objAsKey, 'What will happen?')

La key de l’élément Map est maintenant l’objet que nous avons créé.

Output

key: {foo: "bar"}

value: "What will happen?"

Il y a une chose importante à noter concernant l’utilisation d’un Objet ou d’un Tableau comme clé : l’objet Map utilise la référence à l’Objet pour comparer l’égalité, et non la valeur littérale de l’Objet. En JavaScript {} === {} renvoie false (faux), car les deux Objets ne sont pas les mêmes deux Objets, bien qu’ayant la même valeur (vide).

Cela signifie que l’ajout de deux Objets uniques ayant la même valeur créera un objet Map à deux entrées :

// Add two unique but similar objects as keys to a Map

map.set({}, 'One')

map.set({}, 'Two')

Il en résultera ce qui suit :

Output

Map(2) {{…} => "One", {…} => "Two"}

Mais en utilisant deux fois la même référence d’Objet, on crée un objet Map avec une seule entrée.

// Add the same exact object twice as keys to a Map

const obj = {}

map.set(obj, 'One')

map.set(obj, 'Two')

Ce qui donnera le résultat suivant :

Output

Map(1) {{…} => "Two"}

Le second set() met à jour exactement la même clé que le premier, de sorte que nous nous retrouvons avec un objet Map qui n’a qu’une seule valeur.

## Ajout et suppression d’éléments d’un objet Map

L’un des inconvénients du travail avec les Objets est qu’il peut être difficile de les énumérer, ou de travailler avec toutes les clés ou valeurs. La structure de l’objet Map, en revanche, possède de nombreuses propriétés intégrées qui rendent le travail avec leurs éléments plus direct.

Nous pouvons initialiser un nouvel objet Map pour démontrer les méthodes et propriétés suivantes : delete(), has(), get(), et size.

// Initialize a new Map

const map = new Map([

['animal', 'otter'],

['shape', 'triangle'],

['city', 'New York'],

['country', 'Bulgaria'],

])

Utilisez la méthode has() pour vérifier l’existence d’un élément dans un objet Map. has() renverra un booléen.

// Check if a key exists in a Map

map.has('shark') // false

map.has('country') // true

Utilisez la méthode get() pour récupérer une valeur par clé.

// Get an item from a Map

map.get('animal') // "otter"

L’un des avantages particuliers des objets Map par rapport aux Objets est que vous pouvez trouver la taille d’un objet Map à tout moment, comme vous le feriez avec un Tableau. Vous pouvez obtenir le nombre d’articles dans un objet Map avec la propriété size. Cela implique moins d’étapes que la conversion d’un Objet en Tableau pour en déterminer la longueur.

// Get the count of items in a Map

map.size // 4

Utilisez la méthode delete() pour supprimer un élément d’un objet Map par clé. La méthode retournera un booléen-true (vrai) si un élément existait et a été supprimé, et false (faux) s’il ne correspondait à aucun élément.

// Delete an item from a Map by key

map.delete('city') // true

Il en résultera la carte suivante :

Output

Map(3) {"animal" => "otter", "shape" => "triangle", "country" => "Bulgaria"}

Enfin, un objet Map peut être débarrassé de toutes ses valeurs avec map.clear().

// Empty a Map

map.clear()

Il en résultera :

Output

Map(0) {}

## Clés, Valeurs et Entrées pour objets Map

Les objets peuvent récupérer des clés, des valeurs et des entrées en utilisant les propriétés du constructeur d'Objet. Les objets Map, en revanche, disposent de méthodes prototypes qui nous permettent d’obtenir directement les clés, les valeurs et les entrées de l’instance de l’objet Map.

Les méthodes keys(), values(), et entries() renvoient toutes un MapIterator, qui est similaire à un Tableau en ce sens que vous pouvez utiliser for...of pour boucler les valeurs.

Voici un autre exemple d’objet Map, que nous pouvons utiliser pour démontrer ces méthodes :

const map = new Map([

[1970, 'bell bottoms'],

[1980, 'leg warmers'],

[1990, 'flannel'],

])

La méthode keys() renvoie les clés :

map.keys()

Output

MapIterator {1970, 1980, 1990}

La méthode values() renvoie les valeurs :

map.values()

Output

MapIterator {"bell bottoms", "leg warmers", "flannel"}

La méthode entries() renvoie un tableau de paires clé/valeur :

map.entries()

Output

MapIterator {1970 => "bell bottoms", 1980 => "leg warmers", 1990 => "flannel"}

## Itération avec objet Map

Map dispose d’une méthode forEach intégrée, similaire à celle d’un Tableau, pour l’itération intégrée. Cependant, il y a une petite différence dans ce qu’ils répètent. Le rappel du forEach d’un objet Map se fait par la value, la key et l’objet Map lui-même, tandis que la version Tableau se fait par l’élément, l’index et le tableau lui-même.

// Map

Map.prototype.forEach((value, key, map) = () => {})

// Array

Array.prototype.forEach((item, index, array) = () => {})

C’est un grand avantage pour les objets Map par rapport aux Objets, car les Objets doivent être convertis avec keys(), values() ou entries(), et il n’y a pas de moyen simple de récupérer les propriétés d’un Objet sans le convertir.

Pour le démontrer, parcourons notre objet Map et enregistrons les paires clé/valeur sur la console :

// Log the keys and values of the Map with forEach

map.forEach((value, key) => {

console.log(`${key}: ${value}`)

})

Il en résultera :

Output

1970: bell bottoms

1980: leg warmers

1990: flannel

Comme une boucle for...of se réitère sur des itérables comme objet Map et Tableau, nous pouvons obtenir exactement le même résultat en déstructurant l’ensemble des éléments de l’objet Map :

// Destructure the key and value out of the Map item

for (const [key, value] of map) {

// Log the keys and values of the Map with for...of

console.log(`${key}: ${value}`)

}

## Propriétés et méthodes Map

Le tableau suivant présente une liste des propriétés et des méthodes Map pour une consultation rapide :

| **Propriétés/Méthodes** | **Description** | **Retours** |
| --- | --- | --- |
| [set(key, value)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/set) | Ajoute une paire clé/valeur à une carte | Objet Map |
| [delete(key)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/delete) | Supprime une paire clé/valeur d’un objet Map par clé | Booléen |
| [get(key)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/get) | Retourne une valeur par clé | valeur |
| [has(key)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/has) | Recherche la présence d’un élément dans un objet Map par clé | Booléen |
| [clear()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/clear) | Supprime tous les éléments d’un objet Map | S/O |
| [keys()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/keys) | Retourne toutes les clés dans un objet Map | Objet MapIterator |
| [values()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/values) | Retourne toutes les valeurs dans un objet Map | Objet MapIterator |
| [entries()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/entries) | Retourne toutes les clés et valeurs dans un objet Map sous forme de [key, valeur] | Objet MapIterator |
| [forEach()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/forEach) | Se réitère dans l’objet Map dans l’ordre d’insertion | S/O |
| [size](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map/size) | Retourne le nombre d’items dans un objet Map | Number |

## Quand utiliser Map

En résumé, les objets Map sont similaires aux Objets dans la mesure où ils contiennent des paires clé/valeur, mais les objets Map présentent plusieurs avantages par rapport aux Objets :

* **Taille** - Les objets Map ont une propriété size, alors que les Objets n’ont pas de moyen intégré pour récupérer leur taille.
* **Itération** - Les objets Map sont directement itérables alors que les Objets ne le sont pas.
* **Flexibilité** - Les objets Map peuvent avoir n’importe quel type de données (primitives ou Objets) comme clé pour une valeur, alors que les Objets ne peuvent avoir que des chaînes de caractères.
* **Ordonné** - Les objets Map conservent leur ordre d’insertion, alors que les Objets n’ont pas d’ordre garanti.

En raison de ces facteurs, les objets Map constituent une structure de données puissante à prendre en compte. Cependant, les Objets présentent également des avantages importants :

* **JSON** - Les Objets fonctionnent parfaitement avec JSON.parse() et JSON.stringify(), deux fonctions essentielles pour travailler avec [JSON](https://www.digitalocean.com/community/tutorials/how-to-work-with-json-in-javascript), un format de données commun que de nombreuses REST API traitent.
* **Travailler avec un seul élément** - En travaillant avec une valeur connue dans un Objet, vous pouvez y accéder directement avec la clé sans avoir besoin d’utiliser une méthode, telle que get() d’objet Map.

Cette liste vous aidera à décider si un objet Map ou un Objet est la bonne structure de données pour votre cas d’utilisation.

# Objet Set

Un objet Set est une collection de valeurs uniques. Contrairement à un objet Map, un objet Set est conceptuellement plus proche d’un Tableau que d’un Objet, puisqu’il s’agit d’une liste de valeurs et non de paires clé/valeur. Toutefois, l’objet Set ne remplace pas les Tableaux, mais constitue plutôt un complément pour fournir un soutien supplémentaire pour le travail avec des données dupliquées.

Vous pouvez initialiser les objets Set avec la nouvelle syntaxe Set().

const set = new Set()

Cela nous donne un objet Set vide :

Output

Set(0) {}

Les éléments peuvent être ajoutés à un objet Set avec la méthode add(). (À ne pas confondre avec la méthode set() disponible pour Map, bien qu’elles soient similaires).

// Add items to a Set

set.add('Beethoven')

set.add('Mozart')

set.add('Chopin')

Comme les objets Set ne peuvent contenir que des valeurs uniques, toute tentative d’ajouter une valeur qui existe déjà sera ignorée.

set.add('Chopin') // Set will still contain 3 unique values

**Note** : la même comparaison d’égalité qui s’applique aux clés Map s’applique aux items Set. Deux objets qui ont la même valeur mais ne partagent pas la même référence ne seront pas considérés comme égaux.

Vous pouvez également initialiser les objets Set avec un Tableau de valeurs. S’il y a des valeurs en double dans le tableau, elles seront supprimées de l’objet Set.

// Initialize a Set from an Array

const set = new Set(['Beethoven', 'Mozart', 'Chopin', 'Chopin'])

Output

Set(3) {"Beethoven", "Mozart", "Chopin"}

Inversement, un objet Set peut être converti en un Tableau avec une ligne de code :

const arr = [...set]

Output

(3) ["Beethoven", "Mozart", "Chopin"]

Set possède un grand nombre de méthodes et de propriétés identiques à celles de Map, notamment delete(), has(), clear() et size.

// Delete an item

set.delete('Beethoven') // true

// Check for the existence of an item

set.has('Beethoven') // false

// Clear a Set

set.clear()

// Check the size of a Set

set.size // 0

Notez que Set n’a pas de moyen d’accéder à une valeur par une clé ou un index, comme Map.get(key) ou arr[index].

## Clés, Valeurs et Entrées pour objets Set

Map et Set ont tous deux des méthodes keys(), values(), et entries() qui renvoient un itérateur. Cependant, si chacune de ces méthodes a un but distinct dans Map, les objets Set n’ont pas de clés, et les clés sont donc un alias pour les valeurs. Cela signifie que keys() et values() renverront toutes deux le même itérateur, et que entries() renverra la valeur deux fois. Il est plus logique de n’utiliser que des values() avec Set, car les deux autres méthodes existent pour la cohérence et la compatibilité croisée avec Map.

const set = new Set([1, 2, 3])

// Get the values of a set

set.values()

Output

SetIterator {1, 2, 3}

## Itération avec Set

Comme Map, Set a une méthode intégrée forEach(). Comme les objets Set n’ont pas de clés, le premier et le second paramètre du rappel forEach() renvoient la même valeur, il n’y a donc pas de cas d’utilisation en dehors de la compatibilité avec Map. Les paramètres de forEach() sont (value, key, set).

forEach() et for...of peuvent tous deux être utilisés sur Set. Tout d’abord, examinons l’itération forEach() :

const set = new Set(['hi', 'hello', 'good day'])

// Iterate a Set with forEach

set.forEach((value) => console.log(value))

Ensuite, nous pouvons écrire la version for...of :

// Iterate a Set with for...of

for (const value of set) {

console.log(value);

}

Ces deux stratégies donneront les résultats suivants :

Output

hi

hello

good day

## Propriétés et méthodes de Set

Le tableau suivant présente une liste des propriétés et des méthodes de Set pour une consultation rapide :

| **Propriétés/Méthodes** | **Description** | **Retours** |
| --- | --- | --- |
| [add(value)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/add) | Ajoute un nouvel élément à un objet Set | Objet Set |
| [delete(value)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/delete) | Supprime l’élément spécifié d’un objet Set | Booléen |
| [has()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/has) | Recherche la présence d’un item dans un objet Set | Booléen |
| [clear()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/clear) | Supprime tous les éléments d’un objet Set | S/O |
| [keys()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/values) | Renvoie toutes les valeurs d’un objet Set (identique à values()) | Objet SetIterator |
| [values()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/values) | Renvoie toutes les valeurs d’un objet Set (identique à values() | Objet SetIterator |
| [entries()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/entries) | Retourne toutes les valeurs d’un objet Set comme [value, value] | Objet SetIterator |
| [forEach()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/forEach) | Se réitère dans l’objet Set dans l’ordre d’insertion | S/O |
| [size](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set/size) | Retourne le nombre d’items de l’objet Set | Number |

## Quand utiliser Set

Set est un complément utile à votre boîte à outils JavaScript, en particulier pour travailler avec des valeurs doubles dans les données.

En une seule ligne, nous pouvons créer un nouveau Tableau sans valeurs doubles à partir d’un Tableau qui a des valeurs doubles.

const uniqueArray = [ ...new Set([1, 1, 2, 2, 2, 3])] // (3) [1, 2, 3]

Il en résultera :

Output

(3) [1, 2, 3]

Set peut être utilisé pour trouver l’union, l’intersection et la différence entre deux ensembles de données. Toutefois, les Tableaux présentent un avantage significatif par rapport aux objets Set pour la manipulation supplémentaire des données en raison des méthodes sort(), map(), filter() et reduce(), ainsi que de la compatibilité directe avec les méthodes JSON.

# Conclusion

Dans cette partie, vous avez appris qu’un objet Map est une collection de paires de clés/valeurs ordonnées, et qu’un objet Set est une collection de valeurs uniques. Ces deux structures de données ajoutent des capacités supplémentaires à JavaScript et simplifient les tâches courantes telles que la recherche de la longueur d’une collection de paires clé/valeur et la suppression des éléments en double d’un ensemble de données, respectivement. D’autre part, les Objets et les Tableaux ont été traditionnellement utilisés pour le stockage et la manipulation de données en JavaScript, et sont directement compatibles avec JSON, ce qui continue à en faire les structures de données les plus essentielles, notamment pour travailler avec les REST API. Les objets Map et Set sont principalement utiles comme structures de données de soutien pour les Objets et les Tableaux.

# Ressources complémentaires

<https://fr.javascript.info/map-set>