# Map

O objeto Map contém pares de chave-valor e lembra a ordem original da inserção das chaves. Qualquer valor (objetos e valores primitivos) podem ser usados como chave ou valor.

* Cria um novo array
* Não modifica o array original
* Realiza operações em ordem

## Descrição

Um objeto Map itera seus elementos na order da inserção - um loop for...of retorna um array de [key, value] para cada iteração

Sintaxe:

**array.map(callback,thisArg);**

**calback(item, index, array);**

**Callback –** função a ser executada em cada elemento

**thisArg (opcional):** valor de ‘this’ dentro da função call-back

# Filter

O objeto Filter cria um novo vetor baseado no original que atendem às condições do filtro.

* Cria um novo array
* Não modifica o array original

Sintaxe:

**array.filter(call-back, thisArg);**

**Callback –** função a ser executada em cada elemento

**thisArg (opcional):** valor de ‘this’ dentro da função callBack



## Igualdade de chaves

A igualdade de chaves é baseada no algoritimo sameValueZero.

O NaN é considerado o mesmo que o NaN (apesar de NaN !== NaN) e todos os outros valores são considerados de acordo com a semântica do operador ===.

Na especificação atual do ECMAScript, -0 e +0 são considerados iguais, embora não tenha sido nos esboços anteriores. Veja "Equivalência de valor entre -0 e 0" em Compatibilidade com navegadores para mais detalhes;

## Objetos vs. Maps

Um Object é similar o Map - ambos permitem que valores sejam definidos a chaves, retornar esses valores, remover as chaves, e detectar se algo está armazenado na chave. Por esta razão (e porque não existem outras alternativas construídas), o Objeto tem sido usado como Map historicamente.

Porém, existem diferenças importantes que fazem o Map ser preferido em alguns casos:

|  | **Map** | **Objeto** |
| --- | --- | --- |
| **Chaves acidentais** | O Map não contém nenhuma chave por padrão. Ele só contém o que é definido explicitamente nele. | Um Objeto tem um prototype, então ele contém por padrão algumas chaves que podem conflitar com suas próprias chaves se você não for cuidadoso.  **Note:** Apartir do ES5, isso pode ser ignorado com o [Object.create(null)](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Object/create), mas isso raramente é feito. |
| **Tipos das chaves** | As chaves do Map podem ser qualquer valor (incluindo funções, objetos, ou qualquer outro tipo primitivo). | A chave de um Objeto deve ser uma [String](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/String) ou um [Symbol](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Symbol). |
| **Ordem das chaves** | As chaves dentro do Map são ordenadas de forma simples, de maneira direta: O objeto Map itera suas entradas, chaves, e valor na ordem em que foram inseridas. | Embora as chaves de um Objeto comum hoje sejam ordenadas, nem sempre esse foi o caso, e a ordem é complexa. Como resultado é melhor não confiar na ordem das propriedades.  A ordem foi definida primeiramente para suas próprias propriedades apenas no ECMAScript 2015; no ECMAScript 2020 a ordem definida é por propriedades herdadas também. Veja o [OrdinaryOwnPropertyKeys](https://tc39.es/ecma262/" \l "sec-ordinaryownpropertykeys) e [numerateObjectProperties](https://tc39.es/ecma262/" \l "sec-enumerate-object-properties)operações de especificações abstraídas. Mas note que nenhum mecanismo itera **todas** as propriedades do objeto; cada um dos vários mecanismos incluem diferentes subconjuntos de propriedades. ([for-in](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/for...in) incluí apenas propriedades enumeráveis com chaves do tipo string [Object.keys](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Object/keys) inclui apenas chaves próprias e enumeráveis do tipo string; [Object.getOwnPropertyNames](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Object/getOwnPropertyNames) incluí o próprio, e propriedades com chaves do tipo string, mesmo que não enumeráveis [Object.getOwnPropertySymbols](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Object/getOwnPropertySymbols) faz o mesmo só que para propriedades Símbolos como chave, etc.) |
| **Tamanho** | O número de items dentro de um Map é facilmente retornado pela propriedade [size](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Map/size) | O número de items dentro de um Objeto deve ser determinado manualmente |
| **Iteração** | Um Map é [iterável](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Iteration_protocols), então ele pode ser diretamente iterável | O Objeto não implementa o [protocolo de iteração](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Iteration_protocols#the_iterable_protocol), e os objetos não podem ser iterados diretamente usando o [for...of](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/for...of) (por padrão).  **Nota:**   * Um objeto pode implementar o protocolo de iteração, ou pode ser iterado usando o [Object.keys](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Object/keys)ou[Object.entries](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Object/entries). * A declaração [for...in](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/for...in) permite que você itere pela propriedades enumeráveis de um objeto. |
| **Performance** | Perfoma melhor em cenários envolvendo adições e remoções frequentes em pares chave-valor. | Não é otimizado para adições e remoções frequentes de pares chave-valor. |
| **Serialização e análise sintática** | Não há suporte nativo para a serialização ou análise sintática.  (Mas você pode construir sua própria serialização e conversão para o Map usando o [JSON. stringify()](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify) como o argumento *replacer*, e usando o [JSON.parse()](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse) com o argumento *reviver*. Veja a questão no Stack Overflow [How do you JSON.stringify an ES6 Map?](https://stackoverflow.com/q/29085197/)). | Suporte nativo para serialização de [Objeto](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Object) para JSON, usando [JSON.stringify()](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify).  Suporte nativo para conversão de JSON para [Objeto](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Object) usando [JSON.parse()](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse). |

## Definindo propriedades no objeto

Definir propriedades no objeto também funciona em objetos Map, e pode causar um confusão considerável.

Portanto, isso aparenta funcionar de certa forma:

**const wrongMap = new Map()**

**wrongMap['bla'] = 'blaa'**

**wrongMap['bla2'] = 'blaaa2'**

**console.log(wrongMap) // Map { bla: 'blaa', bla2: 'blaaa2' }**

Mas esse jeito de definir propriedades não interage com a estrura de dados do Map. Dessa forma é usada a implementação genérica do objeto. O valor 'bla' não é armazenado no Map para queries. Outras operações nos dados irão falhar.

**wrongMap.has('bla') // false**

**wrongMap.delete('bla') // false**

**console.log(wrongMap) // Map { bla: 'blaa', bla2: 'blaaa2' }**

**Copy to Clipboard**

A maneira correta para armazenar dados dentro do Map é através do set(key,value)

**const contacts = new Map()**

**contacts.set('Jessie', {phone: "213-555-1234", address: "123 N 1st Ave"})**

**contacts.has('Jessie') // true**

**contacts.get('Hilary') // undefined**

**contacts.set('Hilary', {phone: "617-555-4321", address: "321 S 2nd St"})**

**contacts.get('Jessie') // {phone: "213-555-1234", address: "123 N 1st Ave"}**

**contacts.delete('Raymond') // false**

**contacts.delete('Jessie') // true**

**console.log(contacts.size) //**

**Diferença entre Map e Objetos**

**MAP**

* Maps podem ter chaves de qualquer tipo
* Maps possuem a propriedade length
* Maps são mais fáceis de interar
* Utilizado quando o valor das chaves é desconhecido
* Os valores tem o mesmo tipo

# Set

São estruturas que armazenam apenas **valores únicos**. Os valores nunca se repetem.

## Métodos do Set()

## Add();

## Has();

## Delete();

Exemplo:

const mySet = new Set();

mySet.add(1);

mySet.add(5);

mySet.has(1);

//true pois foi adicionado o número 1 dentro do array

mySet.has(3);

//false pois foi não adicionado o número 3 dentro do array

mySet.delete(5);

//apaga o número 5 de dentro do array

**SET vs Array**

* Possui valores únicos;
* Em vez da propriedade length, consulta-se o número de registros pela propriedade size;
* Não possui os métodos map, filter, reduce etc.

# Reduce

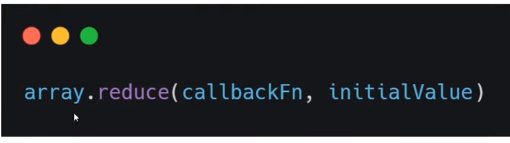
O objeto Reduce executa uma função em todos os elementos do array, retornando um valor único, podendo ser um array ou não.

Sintaxe:

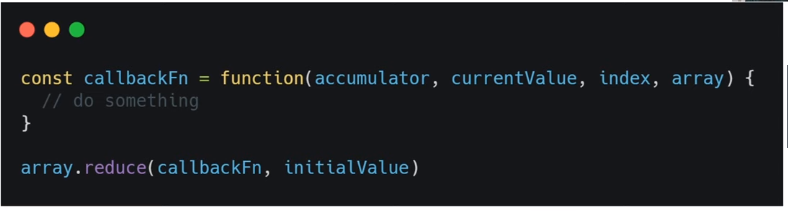
**array.reduce(callbackFn, initialValue);**

Callback – função a ser executada a partir do acumulador.

InitialValue – valor sobre o qual o retorno final irá atuar (opcional)



Exemplo:



**Accumulator / prevValue:** acumulador de todas as chamadas de callbackFn

**currentValue:** elemento atual sendo acessado pela função

neste exemplo funciona como uma função recursiva.