



BACHARELADO EM SISTEMAS DE INFORMAÇÃO
CEFET-RJ NOVA FRIBURGO

Programação de Clientes Web

PROF. THIAGO DELGADO PINTO

Utilidades de Arrays

versão: 2023.09.26



Licença Creative Commons 4

agenda

introdução

includes (ES6)

some

every

find

findIndex

filter

map

forEach

from (ES6)

operações de conjunto (bônus)

introdução

este material é complementar ao de *Array*

veremos métodos úteis, **combinados ao uso de *Arrow Functions***
estilo similar à programação funcional

várias construções se mostram bastante produtivas
uso é recomendado

includes

`includes(elemento: any, fromIndex: index = 0): boolean`

verifica se um elemento está presente no *array*

exemplo:

```
[ 'a', 'b', 'c' ].includes( 'b' ) // true  
[ 'a', 'b', 'c' ].includes( 'd' ) // false
```

some

```
some(  
  callback: (element?: any, index?: number, arr?: Array)=> boolean,  
  thisArg?: any  
): boolean
```

testa se **um** dos elementos passa no teste (*callback*)

exemplo:

```
[ 'a', 'b', 'c' ].some( e => 'b' === e ) // true  
[ 'a', 'b', 'c' ].some( e => 'd' === e ) // false  
[ 'a', 'b', 'c' ].some(  
  (e,i) => 'b' === e && i > 1 ) // false
```

every

```
every(  
  callback: (element?: any, index?: number, arr?: Array)=> boolean,  
  thisArg?: any  
): boolean
```

testa se **todos** os elementos passam no teste (*callback*)

exemplo:

```
[ 'a', 'b', 'c' ].every( e => 1 === e.length ) // true
```

findIndex

```
findIndex(  
  callback: (element?: any, index?: number, arr?: Array)=> boolean,  
  thisArg?: any  
): number
```

retorna o índice do primeiro elemento que passar no teste (*callback*)

exemplo:

```
[ 'a', 'b', 'c' ].findIndex( e => 'b' === e ) // 1  
[ 'a', 'b', 'c' ].findIndex( e => 'd' === e ) // -1  
[ 'a', 'b', 'c' ].findIndex(  
  (e,i) => 'b' === e && i > 1 ) // -1
```

```
find(  
  callback: (element?: any, index?: number, arr?: Array)=> boolean,  
  thisArg?: any  
): any
```

retorna o primeiro elemento que passar no teste (*callback*)
ou **undefined**, caso contrário

exemplo:

```
[ 'a', 'b', 'c' ].find( e => 'b' === e ) // b  
[ 'a', 'b', 'c' ].find( e => 'd' === e ) // undefined
```



```
[ { nome: 'Ana', altura: 1.75 },  
  { nome: 'Bia', altura: 1.40 },  
  { nome: 'Carlos', altura: 1.72 },  
  { nome: 'Darlan', altura: 1.68 },  
].find( e => e.altura >= 1.70 )  
// { nome: 'Ana', altura: 1.75 }
```

```
[ { nome: 'Ana', altura: 1.75 },  
  { nome: 'Bia', altura: 1.40 },  
  { nome: 'Carlos', altura: 1.72 },  
  { nome: 'Darlan', altura: 1.68 },  
].find( e => e.nome.startsWith( 'C' ) )  
// { nome: 'Carlos', altura: 1.72 }
```

```
filter(  
  callback: (element?: any, index?: number, arr?: Array)=> boolean,  
  thisArg?: any  
): Array
```

filtra o array para retornar apenas os elementos que passam no teste (*callback*)

exemplo:

```
[ 'a', 'b', 'c' ].filter( e => 'b' !== e ) // [ 'a', 'c' ]  
[ 'a', 'b', 'c' ].filter( e => 'b' === e ) // [ 'b' ]  
[ 'a', 'b', 'c' ].filter( e => 'd' === e ) // []
```

```
[ { nome: 'Ana', altura: 1.75 },  
  { nome: 'Bia', altura: 1.40 },  
  { nome: 'Carlos', altura: 1.72 },  
  { nome: 'Darlan', altura: 1.68 },  
].filter( e => e.altura >= 1.70 )  
// [ { nome: 'Ana', altura: 1.75 }, { nome: 'Carlos', altura: 1.72 } ]
```

```
[ { nome: 'Ana', altura: 1.75 },  
  { nome: 'Bia', altura: 1.40 },  
  { nome: 'Carlos', altura: 1.72 },  
  { nome: 'Darlan', altura: 1.68 },  
].filter( e => e.nome.contains( 'n' ) )  
// [ { nome: 'Ana', altura: 1.75 }, { nome: 'Darlan', altura: 1.68 } ]
```

map

```
map(  
  callback: (element?: any, index?: number, arr?: Array)=> any,  
  thisArg?: any  
): Array
```

chama uma função (callback) para cada elemento do array e constrói um novo array com os retornos das chamadas

exemplo:

```
[ 'a', 'b', 'c' ].map( e => e + '1' )  
// [ 'a1', 'b1', 'c1' ]
```

```
[ { nome: 'Ana', altura: 1.75 },  
  { nome: 'Bia', altura: 1.40 },  
  { nome: 'Carlos', altura: 1.72 },  
  { nome: 'Darlan', altura: 1.68 },  
].map( e => e.nome + ' mede ' + e.altura + 'm' )  
// [ 'Ana mede 1.75m', 'Bia mede 1.40m', 'Carlos mede 1.72m', 'Darlan mede 1.68m' ]  
  
[  
  'Ana mede 1.75m',  
  'Bia mede 1.40m',  
  'Carlos mede 1.72m',  
  'Darlan mede 1.68m',  
].map( e => {  
  const partes = e.split( ' mede ' );  
  return { nome: partes[0], altura: Number(partes[1].substring(0,4))};  
} );
```

combinando funções

```
[ { nome: 'Ana', altura: 1.75 },  
  { nome: 'Bia', altura: 1.40 },  
  { nome: 'Carlos', altura: 1.72 },  
  { nome: 'Darlan', altura: 1.68 },  
].filter( e => e.altura >= 1.70 )  
.map( e => e.nome + ' mede ' + altura + 'm' )  
// [ 'Ana mede 1.75m', 'Carlos mede 1.72m' ]
```

```
[ { nome: 'Ana', altura: 1.75 },  
  { nome: 'Bia', altura: 1.40 },  
  { nome: 'Carlos', altura: 1.72 },  
  { nome: 'Darlan', altura: 1.68 },  
].map( e => e.nome + ' mede ' + altura + 'm' )  
.find( e => e.contains( '1.68m' ) )  
// 'Darlan mede 1.68m'
```

forEach

```
forEach(  
  callback: (element?: any, index?: number, arr?: Array)=> any,  
  thisArg?: any  
): Array
```

chama uma função (*callback*) para cada elemento do array

exemplo:

```
[ 'a', 'b', 'c' ].forEach( e => console.log( e ) );  
// irá imprimir os elementos
```

map *versus* forEach

utilize **map()** se você quiser gerar um novo array

utilize **forEach()** se você apenas quiser executar algo com os elementos do array, sem retorná-lo


```
static from(  
  iterable: object,  
  mapFn?: (element?: any, index?: number, arr?: Array)=> any,  
  thisArg?: any  
): Array
```

cria um *array* a partir de um objeto iterável

exemplo:

```
Array.from( new Set( [ 'a', 'a', 'b', 'c' ] ) );  
// [ 'a', 'b', 'c' ]  
Array.from( new Map( [ ['a', 1], ['b', 2], ['c', 3] ] ) );  
// [ ['a', 1], ['b', 2], ['c', 3] ]
```

1. Crie uma função *entreMinMax* que receba um array, um valor mínimo e um valor máximo e retorne um novo array contendo apenas os elementos entre o mínimo e o máximo indicados, incluindo eles. Exemplo:

```
entreMinMax( [ 1, 2, 3, 4, 5, 6 ], 2, 5 ) // [ 2, 3, 4, 5 ]
```

2. Crie uma função *multiplicar*, que receba um array e um número e que retorne um novo array contendo os números do array recebido multiplicados por esse número. Exemplo:

```
multiplicar( [ 1, 2, 3, 4 ], 2 ) // [ 2, 4, 6, 8 ]
```

3) Considere a lista de contatos a seguir:

```
[ { nome: 'Ana', sobrenome: 'Souza', tel: { ddd: '22', numero: '999887766' } },  
  { nome: 'Beto', sobrenome: 'Costa', tel: { ddd: '21', numero: '999776655' } },  
  { nome: 'Bia', sobrenome: 'Andrade', tel: { ddd: '21', numero: '988554433' } },  
  { nome: 'Carla', sobrenome: 'Silva', tel: { ddd: '24', numero: '998606060' } },  
  { nome: 'Carlos', sobrenome: 'Rocha', tel: { ddd: '22', numero: '988223344' } },  
]
```

Utilize funções vistas anteriormente nas seguintes soluções:

a) Exiba apenas os contatos com DDD 22.

b) Transforme a lista original em uma lista de strings, contendo o nome e o telefone concatenados, conforme o exemplo: ['Ana Souza - (22) 999887766', ...]

c) Transforme a lista original em outra lista de contatos, conforme o exemplo:

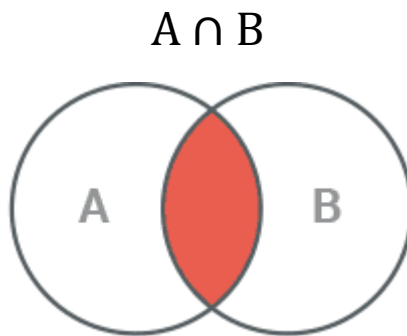
```
[ { nomeCompleto: 'Souza, Ana', telefone: '(22) 999887766' }, ... ]
```

d) Da lista criada no item anterior, exiba somente os contatos com sobrenome iniciando com a letra "S".

operações de conjunto

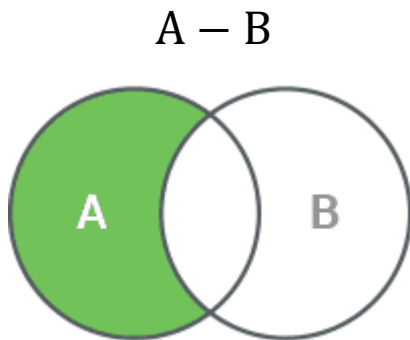
interseção

```
const intersection =  
  a.filter( item => b.includes( item ) );
```



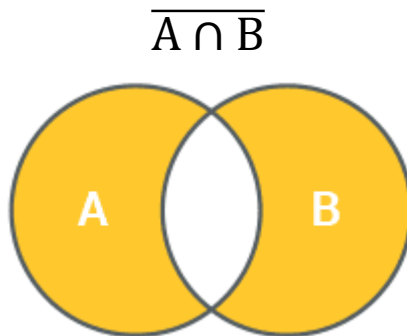
diferença

```
const difference =  
  a.filter( item => ! b.includes( item ) );
```



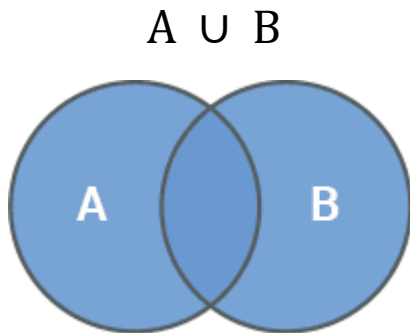
diferença simétrica

```
const symmetricDifference =  
  a.filter( item => ! b.includes( item ) )  
  .concat(  
    b.filter( item => ! a.includes( item ) )  
  );
```



união

```
// const union = a.concat( b );  
// const union = Array.from( new Set( a.concat( b ) ) );  
  
// const union = [ ...a, ...b ];  
const union = [ ...( new Set( [ ...a, ...b ] ) ) ];
```



👉 O operador de espalhamento (*spread*) é bem mais rápido que método *concat*. Use-o.

referências

MDN. *Array – JavaScript*. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array

fim

2023.09.26 - Adiciona `forEach()` e `comparative` com `map()`.

2022.01.21 - Melhora legibilidade dos exemplos com operações de conjunto. Coloca slide "Agenda" logo após capa. Ajusta rodapés.

2020.10.26 - Adiciona exercícios.

2019.10.08 - Versão inicial.

