



JavaScript

Métodos dos arrays

JS
JavaScript

parte 11

aprendendo Javascript

agenda

vamos voltar a falar sobre os arrays
e mostrar como é importante
explorar o potencial desse objeto
JavaScript.



o objeto global array

tem muitos métodos úteis e importantes, vamos conhecer o **forEach**

a documentação do ECMAScript 6 apresenta um conjunto muito rico de métodos para o objeto Array do JavaScript. O que veremos a seguir são alguns destes principais métodos e como eles podem ser bem úteis em detrimento a outros tipos de construção de código. O primeiro destes métodos é o `forEach` que já implementa uma iteração nativa no objeto sem necessidade de criar estruturas de repetição como `for`, `while` ou `do`.

```
arr.forEach(callback(currentValue [, index [, array]]) [, thisArg]);
```

o código abaixo percorre o array e exibe na console usando uma estrutura `for`.

```
const pets = ['Stella', 'Egídio', 'Luly', 'Thor'];
for (let i = 0; i < pets.length; i++) {
  console.log(`0 pet se chama ${pets[i]}`);
}
```

já nesta nova escrita, usamos o método `forEach` para iterar cada posição do array e exibi-la na console.

```
function nomePet(value) {
  console.log(`0 pet se chama ${value}`);
}
const pets = ['Stella', 'Egídio', 'Luly', 'Thor'];
pets.forEach(nomePet);
```

codificando em JavaScript

todos os códigos abaixo com **forEach** apresentam mesmo resultado

estruturas declarando `array` como variável

```
function nomePet(value) {  
  console.log(`0 pet se chama ${value}`);  
}  
const pets = ['Stella', 'Egídio', 'Luly', 'Thor'];  
pets.forEach(nomePet);
```

```
const pets = ['Stella', 'Egídio', 'Luly', 'Thor'];  
pets.forEach(value => console.log(`0 pet se chama ${value}`));
```

estruturas com `array` implícitos.

```
function nomePet(value) {  
  console.log(`0 pet se chama ${value}`);  
}  
  
['Stella', 'Egídio', 'Luly', 'Thor'].forEach(nomePet);
```

```
['Stella', 'Egídio', 'Luly', 'Thor'].forEach(value => {  
  console.log(`0 pet se chama ${value}`)});
```

o objeto global array

processar todo os elementos do array com **map**

quanto você precisar efetuar qualquer operação com todos os elementos de um array, pode também usar um método `map` que irá como `forEach` receber uma função de **callback** e vai retornar um **novo array** a partir deste processamento. Ele **não altera** o array original. Mas atenção, elementos com `undefined` não serão processados pelo método.

```
let newArr = arr.map(callback(currentValue [,index [,array]])[, thisArg]);
```


o objeto global array

processar todo os elementos do array com **map**

o código abaixo percorre o array e exibe na console os valores pares em dobro sem usar `for` ou `forEach`.

```
const dobraPares = (value) => {  
  if (value % 2 === 0) {  
    return value * 2;  
  } else {  
    return value;  
  }  
}  
  
let numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9];  
let paresDobrados = numeros.map(dobraPares);  
console.log(numeros);  
console.log(paresDobrados);
```

já neste código usamos uma função `Math.sqrt` para retornar o quadrado dos valores em outro array.

```
let numeros = [1, 4, 9, 16, 25];  
let raizQuadrada = numeros.map(Math.sqrt);  
  
console.log(numeros);  
console.log(raizQuadrada);
```

o objeto global array

filtrando seu array com `filter`

em várias situações você vai querer pegar o conteúdo de um array e aplicar filtros nele. O processo é muito simples e poderoso e certamente trará muita performance para seu programa já que tudo executa em memória. Como o método `map` o `filter` vai receber uma função de **callback** e vai retornar um **novo array**. Ele **não altera** o array original. A callback function deve retornar `true` para ocorrências filtradas e `false` para não filtrada.

```
let newArr = arr.filter(callback(currentValue [, index [, array]]) [, thisArg]);
```

o objeto global array

filtrando seu array com `filter`

o código abaixo percorre o array e exibe na console os valores pares usando `filter`.

```
const soPares = (value) => {  
  if (value % 2 === 0) {  
    return true;  
  } else {  
    return false;  
  }  
}  
  
let numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9];  
let pares = numeros.filter(soPares);  
console.log(numeros);  
console.log(pares);
```

mesmo código agora usando **arrow function** na construção do `filter`.

```
let numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9];  
let pares = numeros.filter(value => value % 2 === 0);  
console.log(numeros);  
console.log(pares);
```


o objeto global array

procurando no array com **find**

em algum momento você vai querer fazer buscas no array para certificar que ele contém algo que você precisa para seu algoritmo. O processo é novamente muito simples. O método `find` vai retornar o conteúdo do array, quando ele for encontrado ou `undefined` quando a procura falhar.

```
let result = arr.find(callback(currentValue [,index [,array]])[, thisArg]);
```

o objeto global array

efetuando busca no array com **find**

o código abaixo percorre o array e busca pelo nome Stella exibindo o valor na console usando `find`.



```
const pessoas = ['Alan', 'Stella', 'Laurinha'];  
// Busca para saber se existe Stella no array  
const result = pessoas.find(value => value === 'Stella');  
console.log(result);
```


o objeto global array

procurando a posição no array com `findIndex`


agora você quer procurar por um alvo, mas quer saber em qual posição ele foi encontrado. O método `findIndex` vai retornar o índice de onde o conteúdo foi encontrado no array. Quando ele não for encontrado vai retornar `-1`.

```
let result = arr.findIndex(callback(currentValue [, index [, array]]) [, thisArg]);
```

o objeto global array

efetuando busca no array com `findIndex`

o código abaixo percorre o array e busca pelo nome Stella e retorna o valor 1 usando `findIndex`.



```
const pessoas = ['Alan', 'Stella', 'Laurinha'];  
const result = pessoas.findIndex(value => value === 'Stella');  
console.log(result); // 1
```


combinando métodos

dá para combinar vários métodos como `filter` e `map` em uma instrução

vamos imaginar que nosso programa deveria pegar um array de inteiros, e elevar ao cubo todos os números ímpares contidos nele. Nós poderíamos primeiro aplicar um `filter` no array gerando um array só com os números ímpares e posteriormente aplicar o `map` ou `forEach` para elevar ao cubo cada um dos elementos. Mas no exemplo a seguir, vamos fazer tudo isto combinando os dois métodos.

versão 1

```
function soImpares(value) {
  if (value % 2 > 0) {
    return true;
  } else {
    return false;
  }
}

const numeros = [ 1, 2, 3, 4, 5, 6, 7];
let impares = numeros.filter(soImpares);
console.log(`Todos os números ${numeros}`);
console.log(`Apenas os ímpares ${impares}`);
console.log(`Ímpares ao cubo`);
impares.map(function exhibe(value) {
  console.log(`${value} ao cubo: ${Math.pow(value, 03)}`);
});
```

versão 2

```
console.log(`Ímpares ao cubo`);
[ 1, 2, 3, 4, 5, 6, 7]
  .filter(value => value % 2 > 0)
  .map(value => console.log(`${value} ao cubo: ${Math.pow(value, 03)}`));
```

command chaining 🙌

método acumulador

usando **reduce** para acumular e retornar algo

normalmente `reduce` é o método mais complicado de se entender, e digo, é pura crença porque em nada tem de complicado para compreender. Diferente de `map` ou `filter` ele não retorna outro array, mas retorna uma coisa como um `number`, uma `string`, um `array` e porque não um objeto.

```
let res = arr.reduce(callback(acumulador, valorAtual [,index [,array]]) [, valorInicial]);
```


método acumulador

usando **reduce** para acumular e retornar algo

o código abaixo percorre o array e exibe na console o valor da soma dos elementos usando `for`.

```
const numeros = [1, 3, 4, 56, 78, 90, 43, 61, 98];
let total = 0;

for (let i = 0; i < numeros.length; i++) {
  total += numeros[i];
}

console.log(`O total do array é ${total}`);
```

o código abaixo usa o método `reduce` para acumular o valor dos elementos do array em uma variável **total**.

```
const numeros = [1, 3, 4, 56, 78, 90, 43, 61, 98];
const total = numeros.reduce(function (acumulador, valor) {
  return acumulador += valor;
}, 0);

console.log(`O total do array é ${total}`);
```



VAMOS PRATICAR

programming

revendo a prática

princípio básico

1. aprendemos o que **callback** functions em JavaScript são a base para a chamada de vários métodos do objeto array.
2. que há uma alternativa melhor, mais rápida e melhor escrita para uso do **for** em arrays chamada **forEach**.
3. o método **map** assim como o **forEach** me permite iterar com todos os elementos do meu array criando um **novo array** a partir dele.
4. com o método **filter** é possível fazer uma seleção muitíssimo inteligente dos dados dos nossos arrays.
5. com **reduce** é possível iterar e criar verdadeiros somatórios ou novos objetos.

