Descubra Quando usar Map Filter e Reduce

Nos dias atuais, onde programadores são disputados por diversas empresas, ter um bom código é essencial. Os métodos **map**, **filter** e **reduce**, são exemplos de como podemos deixar o nosso código mais elegante e consistente.

A utilização destes métodos está muito ligada à **manipulação de arrays**. Quando falamos de map e filter, iremos iterar sobre um array e **devolver um outro array após as alterações** predeterminadas por nós. O método reduce trabalha de uma maneira diferente, **retornando um valor que pode ser array, boolean, objeto, etc.**

Tendo isso em conta, vamos entender melhor como funciona e quando utilizar cada um deles. Mais uma vez, utilizaremos aqui a linguagem **Javascript** como exemplo, porém estes métodos estão presentes em diversas linguagens como o Python e o C#.

Vamos começar? Bom, antes de iniciarmos, vamos utilizar como base para todos os exemplos um array de números, sendo composto por:

```
const array = [2, 4, 5, 6, 9, 12]
```

Map()

A função map é talvez **a mais utilizada das três** que iremos abordar. Ela funciona de forma a **iterar sobre cada elemento** de um array e utilizar uma função de callback que deve ser utilizada para comparar estes elementos a uma determinada situação. Complicado? Vamos de exemplo!

Imagine que você quer usar o nosso array inicial e alterar ele, de forma a mostrar o dobro de todos os números que ele contém. Podemos fazer isso utilizando o método map ou utilizando um loop for. Veja a diferença.

Método Trivial

```
 const array = [2, 4, 5, 6, 9, 12]
 let novo_array = []
 for(let i=0;i<array.length;i++){
   novo_array.push(array[i] * 2)
}
```

No método trivial, tudo que iremos fazer é utilizar um loop para iterar sobre todos os elementos do array e realizar manipulações para depois montarmos um novo array.

Perceba que neste método, utilizamos muitas variáveis que são desnecessárias e que tornam o nosso código pouco refinado e menos legível. Vamos ver como ficaria com o map?

Método map e seu funcionamento

```
const array = [2, 4, 5, 6, 9, 12]
const novo_array = numeros.map(
function dobro (elemento){
return elemento * 2
}
)
```

Perceba que com o método map, tudo fica mais simples e legível. Mas, o que de fato essa função faz?

A função map se utiliza de uma callback function como argumento e **sempre retorna um array.** Em outras palavras, nós sempre teremos uma função (com **retorno obrigatório, ou seja, callback**) como parâmetro e uma nova variável

do tipo array como retorno. No nosso exemplo acima, retornamos o valor de saída do map para a const novo_array e utilizamos a função dobro como argumento.

O que chamamos de elemento no nosso código é justamente o que o nome diz, ou seja, **cada elemento do nosso array inicial** está sendo multiplicado por 2 e retornado pela função. Dentro da nossa função poderíamos dar o nome que quisermos para elemento e, quase que por padrão, nós utilizamos a letra "**e"** para designar o elemento de um array dentro do map.

Perceba por fim, que o retorno da nossa função map deve ser um array, que aqui chamamos de novo_array, **de tamanho igual** (e sempre será igual) ao array inicial. No entanto, agora temos um array formado pelo dobro de cada um dos elementos do array inicial. O output será:

```
[ 4, 8, 10, 12, 18, 24 ]
```

Utilizando **arrow functions e retorno imediato,** teríamos um código ainda mais limpo. Indico que **estude sobre estas duas sintaxes**, mas só depois de terminar de ler o post, okay?

Código com arrow functions e retorno Imediato

```
const novo_array=array.map( elemento => elemento * 2  )
```

Filter()

O método filter funciona de uma forma muito similar ao map. Ambos irão retornar um array, utilizam uma callback function e iteram sobre cada um dos elementos do array inicial. As diferenças, no entanto, estão no retorno do método.

Vimos que no método map, o nosso retorno sempre seria um array de tamanho igual ao do array inicial, **diferente de como ocorre com o filter**. Como queremos filtrar o nosso array de acordo com uma determinada regra (predeterminada em nossa função de callback), **nem sempre o novo array será do mesmo tamanho, podendo muitas vezes ser vazio ou unitário.**

Método trivial

Vamos imaginar que queremos filtrar o nosso array e armazenar em um novo array apenas os números que são múltiplos de 2. Teríamos o seguinte código:

```
 const array = [2, 4, 5, 6, 9, 12]
 let multiplos = []
 for(let i=0;i<array.length;i++){
   if(array[i] % 2 === 0){
    multiplos.push(array[i])
}
}
```

Método filter e seu funcionamento

```
let multiplos=array.filter(function multiplosDeDois
(elemento){
   return elemento % 2 === 0
})
```

Neste caso, temos um retorno e logo após ele uma expressão. Perceba que isto é completamente possível, pois se elemento/2=0, ele irá retornar true, caso contrário, será false.

Você neste momento deve estar se perguntando: mas por que true ou false? Não deveria retornar o elemento? Bom, o filter funciona de uma forma diferente. Aqui, quando o elemento é true para a expressão, significa que **aquele elemento pode ser inserido em nosso novo array**, caso seja false, **não adicionaremos** ele ao novo array de retorno.

Reduce()

O método reduce é sem dúvidas o mais diferenciado dentre os três. Com ele teremos como retorno uma variável de **tipo a ser definido** e é muito utilizado para criar novos arrays ou novas organizações para objetos.

Um exemplo muito claro do reduce é o **filtro de pesquisas.** Quando, ao pesquisar um hotel para as férias, utilizamos o “filtrar por preços”, é um reduce que irá agir para te entregar os novos dados.

Vamos a implementação. Imaginemos agora que queremos retornar a soma de todos os elementos do array inicial. Vejamos como fazer de forma trivial e de forma elegante, com reduce.

Método Trivial

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

Método Reduce e funcionamento

```
const somaReduce = numeros.reduce((soma, array) =>  soma + array, 0)
```

Neste exemplo, **já utilizamos as arrow functions e retorno imediato do ES6.** O reduce trabalha com uma callback function que recebe 2 parâmetros, uma **variável acumuladora**, que será **modificada ao longo das iterações** e o array ou objeto que iremos iterar.

Como vemos no código, o reduce recebe, logo após a nossa callback function, **o valor inicial para a nossa variável acumuladora**, que no nosso exemplo se chama soma e inicia em 0.

A variável acumuladora **pode ser de qualquer tipo**: string, inteiro, objeto. Por conta disso, não podemos afirmar qual será o tipo de variável que o reduce irá retornar para nós, afinal, tudo depende do que estamos iterando.

Conclusão

Map, filter e reduce são funções que muito se assemelham ao **paradigma funcional** de programação. Tornam o nosso código mais limpo, legível e sofisticado. Estude bastante como utilizá-los e **refatore alguns de seus códigos antigos, agora utilizando estes métodos.**

O mercado de trabalho de tecnologia tem avançado muito e cada vez mais são exigidos programadores com **boas práticas de codificação.** Cabe agora a você implementar novas soluções utilizando o poder destes 3 métodos.