Programação Funcional e Declarativa em JavaScript

FAPESC – DESENVOLVEDORES PARA TECNOLOGIA DA INFORMAÇÃO

Herculano De Biasi
herculano.debiasi@unoesc.edu.br

Tópicos

- Estilo funcional/declarativo
- Funções de alta-ordem
  - `forEach()`
  - `map()`
  - `filter()`
  - `reduce()`

Estilo Funcional/Declarativo

- Programação funcional é um paradigma de programação que trata a computação como uma avaliação de funções matemáticas e que evita estados ou dados mutáveis
  - Ela enfatiza a aplicação de funções, em contraste da programação imperativa, que enfatiza mudanças no estado do programa

- Já a programação declarativa é um estilo de programação não imperativa, na qual os programas descrevem os resultados desejados sem listar explicitamente os comandos ou etapas que devem ser executados
  - Na programação declarativa a lógica de computação é expressa sem precisar descrever o seu fluxo de controle
  - Linguagens de programação funcionais e lógicas são caracterizadas por um estilo de programação declarativa

Funções de Alta-Ordem

- As funções abaixo são algumas das assim chamadas de alta-ordem (ou de ordem superior) pois elas recebem uma função (de *callback*) por parâmetro
  - `forEach()`
  - `map()`
  - `filter()`
  - `reduce()`
- As funções *callbacks* são também chamadas, neste caso, de predicados, porque descrevem o comportamento dos filtros e mapeamentos
- Essas funções podem ser encadeadas, ou seja, chamadas uma após a outra, o que é conhecimento como encadeamento de métodos

FUNÇÃO FOREACH ( )

- `forEach()` percorre um *array* e executa uma função *callback* a cada repetição

lista13 - forEach.js

```
1  const animais = ['gato', 'cachorro', 'passarinho'];
2  const sons = ['mia', 'late', 'canta'];
3
4  // for .. of (somente um array por vez)
5  for (animal of animais) {
6      console.log(animal);
7  }
8  console.log();
9
10 // for() clássico
11 for (let i=0; i<animais.length; i++) {
12     console.log(animais[i], sons[i]);
13 }
14 console.log();
15
16 // forEach() usando uma arrow function como função callback
17 animais.forEach( (animal, indice) => {
18     console.log(animal, sons[indice]);
19 });
20 console.log();
21
22 // Exemplo equivalente mas declarando a função separadamente
23 const funcao = (animal, indice) => { console.log(animal, sons[indice]); };
24 animais.forEach(funcao);
```

FUNÇÃO MAP ( )

- `map()` aplica uma função a cada um dos elementos de um *array*

lista13 - map.js

```
1  const numeros = [1, 2, 3, 4, 5];
2
3  const resultados = [];
4  for (numero of numeros) {
5      resultados.push(numero * numero);
6  }
7
8  console.log('numeros:', numeros);
9  console.log('resultado:', resultados);
10 console.log();
11
12 // Estilo funcional/declarativo
13 const resultado = numeros.map( numero => numero * numero );
14
15 console.log('numeros:', numeros);
16 console.log('resultado:', resultado);
17 console.log();
18
19 // Mapeamentos encadeados
20 const soma10 = num => num + 10;
21 const dobra = num => num * 2;
22 const emReais = num => `R$ ${parseFloat(num).toFixed(2).replace('.', ',')}`;
23
24 const res = numeros.map(soma10)
25                    .map(dobra)
26                    .map(emReais);
27 console.log(res);
```

UNOESC

FUNÇÃO `FILTER()`

- `filter()` processa uma lista/*array* e produz uma nova lista contendo somente os elementos que retornaram verdadeiro (`true`) para a condição de filtragem

lista13 - filter.js

```
1  const numeros = [42, 666, 90, 10, 50];
2
3  // Código no estilo imperativo
4  const resultados = [];
5  for (numero of numeros) {
6      if (numero >= 50) {
7          resultados.push(numero);
8      }
9  }
10 console.log(resultados);
11 console.log();
12
13 // Código no estilo declarativo
14 const resultado = numeros.filter(numero => numero >= 50);
15 console.log(resultado);
```

FUNÇÃO REDUCE ( )

- `reduce()` (também chamada de *fold, inject, accumulate* ou *aggregate*) recebe uma função *callback* e aplica-a a cada elemento de uma lista/*array* sendo que a saída de cada iteração alimenta a entrada da seguinte
  - A função anônima de *callback* será executada a cada iteração, e nela serão injetados a cada vez 2 argumentos: o acumulador e o valor atual
  - É responsabilidade do *callback* sempre retornar o acumulador, pois é assim que internamente cada execução irá se comunicar com a próxima, ou seja, a saída de cada iteração é parte da entrada da iteração seguinte
  - `reduce()` pode receber também um valor inicial para o acumulador

lista13 - reduce.js

```
1  const numeros = [5, 3, 2, 7];
2
3  // Estilo imperativo
4  let total = 0;
5  for (numero of numeros) {
6      console.log(total, numero);
7      total += numero;
8  }
9  console.log(total);
10 console.log();
11
12 // Estilo funcional/declarativo
13 const soma = numeros.reduce( (acumulador, atual) => {
14     console.log(acumulador, atual);
15     return acumulador + atual;
16 }, 0);
17
18 console.log(soma);
```

UNOESC