Javascript

Programa de atração de formação de talentos
Bradesco / Visionaire

Aula 04 – Desestruturação, datas e métodos de listas

Mark Joselli

mark.Joselli@pucpr.br

Funções puras

Uma função é considerada pura se obedecer a 2 critérios:

1. Dado uma entrada, ela produz sempre a **mesma** saída
2. Ela não produz efeitos colaterais
   - Não altera parâmetros de entrada
   - Nem propriedades de classe

Ou seja, não altera dados externos a função

Imutabilidade

- Funções puras garantem que os dados utilizados sejam **imutáveis**. E isso traz vantagens:
  - É fácil **rastrear** onde **modificações** em um dado ocorreram, ou criar um histórico
  - Auxilia bibliotecas a **detectarem** que as **mudanças** ocorreram, aumentando possibilidades de otimização
  - Aumenta as possibilidades de compartilhamento da informação

Por isso, damos preferência a funções que **copiam objetos** ao invés de simplesmente modificá-los.

Copiando e desestruturando objetos

- É possível utilizar o operador spread para copiar objetos ou arrays.
- Durante o processo, é possível incluir mais campos ou substituir os já existentes.

```
const cantor = {nome: "Vinícius de Moraes", estilo: "MPB"};
const copia = {...cantor, falecimento: 1980};


const lista = [1,2,3,4,5];
const copia = [0, ...lista, 6];
```

Copiando e desestruturando objetos

- A **desestruturação** permite copiar parte de um objeto ou lista para variáveis de maneira simples.
- É possível especificar valores padrão para as propriedades:

```
const album = {
    nome: "Roupa acústico",
    cantor: "Roupa Nova",
    ano: 2004,
    nota: 9
};
const {nome, ano, favorito = true} = album;
const [a,b,c, ...resto] = [1,2,3,4,5,6];
```

Copiando e desestruturando objetos

- É possível utilizar desestruturação diretamente nos **parâmetros de uma função**

```
function paginacao(lista, {limite = 20, pagina = 0} = {}) {
    //Realiza a paginação aqui
}
```

- Chamada:

```
const pagina = paginacao(albuns, {pagina: 2});
```

Copiando e desestruturando objetos

- É possível alterar o nome da variável durante a desestruturação:

```
let {nome, cantor: banda, favorito = true} = album;
console.log(banda);
```

Copiando e desestruturando objetos

- Também é possível desestruturar objetos e listas aninhados:

```
const album = {
    nome: "Roupa acústico",
    cantor: {
       nome: "Roupa Nova",
       estilos: ["Pop rock", "Soft rock"]
    },
    ano: 2004,
    nota: 9
};


const {nome, cantor: { estilos: [estilo] }, favorito = true} = album;
console.log(nome); //Roupa acústico
console.log(estilo); //Pop rock
```

Copiando e desestruturando objetos

- Também é possível desestruturar objetos e listas aninhados:

```
const album = {
    nome: "Roupa acústico",
    cantor: {
       nome: "Roupa Nova",
       estilos: ["Pop rock", "Soft rock"]
    },
    ano: 2004,
    nota: 9
};


const {nome, cantor: { estilos: [estilo] }, favorito = true} = album;
console.log(nome); //Roupa acústico
console.log(estilo); //Pop rock
```

Encadeamento opcional

- Fornece uma maneira fácil de acessar objetos sem que seja necessário testar se são nulos o tempo todo
- Caso qualquer elemento seja nulo, o retorno será nulo, ex:

```
let nome = album?.cantor?.nome;
```

Pode ser usado até em variáveis de funções:

```
obj.teste?.();
```

Null coalesce

- Similar ao || para atribuição de valores para objetos, mas funciona apenas se o lado esquerdo for nulo
- Não faz coersão automática para boolean ou pode ser combinado a outras expressões lógicas:

```
let nome = cantor.nome ?? "Nome não informado";
```

Object Date

Lidando com datas

- Para trabalhar com datas, o Javascript possui o objeto Date
- O exemplo abaixo, cria uma nova data com o horário atual:

```
const data = new Date();
```

Lidando com datas

- O objeto possui as seguintes funções:
  - `getDate`(): Retorna o dia do mês
  - `getDay`(): Retorna o dia da semana. 0 para domingo, 1 para segunda, etc.
  - `getMonth`(): Retorna o mês atual... Mas no intervalo de 0 até 11
  - `getFullYear`(): Retorna o ano atual
  - `getHours`(), `getMinutes`(), `getSeconds`(), `getMilliseconds`(): Retorna respectivamente a hora, minuto, segundo e os milissegundos da data atual.

Lidando com datas

- Você também pode passar os valores de ano, mês, dia, hora, minuto, segundo e milissegundo no construtor.
- A data também possui os “sets” correspondentes.
  - Todos eles tem a capacidade de corrigir o objeto data caso um valor fora de um intervalo seja fornecido
  - Isso permite somar e subtrair valores de datas facilmente
- Exemplo:

```
const data = new Date(2021, 8, 15); //15/09/2021
data.setDate(data.getDate() + 50); //04/11/2021
```

Lidando com datas

- A data possui o método getTime(). Ele retorna a quantidade de milissegundos transcorridos desde 01/01/1970
- Este valor também pode ser utilizado para inicializar uma data
- Além disso, é útil utilizar esta função para:
  - Comparar duas datas
  - Calcular intervalos de tempo

Imprimindo datas

- Caso você imprima uma data, ela será exibida no formato UTC (o Brasil está no horário BRT, ou seja, 3 horas a frente). Exemplo:

```
2022-09-28T12:04:58.351Z
```

- Use os métodos toLocaleString(), toLocaleDateString(), toLocaleTimeString() para converte-la em um formato mais legível. Você pode passar por parâmetro a região (ex. "pt-br")

```
28/09/2022 09:06:31
```

Nesta aula, iremos explorar as funções que permitem o **processamento de listas** no Javascript

forEach

- O comando que veremos é o `forEach` recebe uma função que é executada uma vez para cada elemento dentro da lista
- Esta função deve aceitar **3 parâmetros**:
  1. O elemento sendo percorrido
  2. O índice do elemento
  3. A lista que está sendo percorrida
- Porém, lembre-se que no Javascript **não precisamos** fornecer funções que utilizem todos eles

```
[1,2,3,4].forEach(console.log);
```

filter

- O comando `filter` retorna uma cópia da lista, contendo apenas os elementos que atendam a um determinado critério
- Cada elemento é submetido a uma função, que deve retornar verdadeiro caso ele deva ser incluído no resultado
- Exemplo:

```
[1,2,3,4,5,6]
    .filter(x => x % 2 === 0)
    .forEach(x => console.log(x));
```

flat

- O comando `flat` irá atuar com uma lista que tenha outras listas dentro. Ele retira os elementos de dentro das listas internas, colocando-os na lista principal.
- A profundidade dessa operação pode ser passada por parâmetro, por padrão a profundidade 1 é usada.

Exemplo:

```
const elementos = [1, [2,3], [4], [[5]], 6];
console.log(elementos.flat());
//imprime 1,2,3,4,[5],6

console.log(elementos.flat(2));
//imprime 1,2,3,4,5,6
```

map

- O comando `map` executa uma função de transformação para cada elemento da lista
- Esta função substitui o elemento pelo que for retornado, criando uma nova lista

```
[1,2,3,4,5,6]
    .map(x => x * 10)
    .forEach(x => console.log(x));
```

flatMap

O `flatMap` combina um comando `map` com um comando `flat(1)` de maneira eficiente

```
const coordenadas = [
    {x: 12.5, y: 17.2},
    {x: 27.4, y: 13.0},
    {x: 14.7, y: 12.0},
];
console.log(coordenadas.map(c => Object.values(c)));
//[ [ 12.5, 17.2 ], [ 27.4, 13 ], [ 14.7, 12 ] ]

console.log(coordenadas.flatMap(c => Object.values(c)));
//[ 12.5, 17.2, 27.4, 13, 14.7, 12 ]
```

reduce

- Executa uma função de redução que recebe os parâmetros:
  - O valor retornado pela função de redução no passo anterior (acumulador)
  - O valor atual
  - O índice atual
  - A lista sendo processada
- Além da função de redução, pode receber como segundo parâmetro o valor inicial. Se não for fornecido, usa o primeiro valor da lista.
- A função de redução deve retornar o próximo valor do acumulador. Por exemplo, para somar todos os elementos de uma lista:

```
const soma = [1, 2, 3, 4]
    .reduce((a, e) => a + e, 0);
console.log("Soma:", soma);
```

Equivale a

```
let a = 0;
for (const e of [1,2,3,4]) {
  a = a + e;
}
```

Sort

Ordena **a própria lista**. Recebe como parâmetro uma função de comparação que recebe como parâmetro dois valores a e b:

- Retornar um valor positivo se a vem antes que b
- Negativo se a vem depois de b
- Zero, se a e b devem ficar na mesma ordem

Cuidado: Por padrão, a função sort ordena transformando os elementos em texto

Concat

Combina o valor de duas ou mais listas em uma nova lista.

```
const num1 = [1,2,3];
const num2 = [4,5,6];
const num3 = [7,8,9];
// [1, 2, 3, 4, 5, 6, 7, 8, 9]; num1, num2, num3 não se modificam
const nums = num1.concat(num2, num3);
```

Hoje o mesmo resultado pode ser obtido com o spread operator:

```
const nums = [...num1, ...num2, ...num3];
```

Relembrando...

Não se esqueça dos comandos que vistos em aulas anteriores:

- `length`: Retorna o tamanho do array
- `push` / `unshift`: Adiciona um item ao final / início
- `pop` / `shift`: Remove um item do final / inicio
- `splice(pos, n)`: Remove *n* elementos a partir do índice *pos*
- `slice(inicio, fim)`: Copia o array da posição *inicio* até *fim*. Caso o fim não seja fornecido copia até o final. Caso nenhum seja fornecido, copia o array todo.
- `indexOf`: Retorna o índice de um elemento

Atividades

- Para as próximas atividades, considere a seguinte lista (disponível no canvas):

```
const albuns = [
    {nome: "Mais", cantor: "Marisa monte", ano: 1991, nota: 8.5},
    {nome: "A tempestade", cantor: "Legião Urbana", ano: 1996, nota: 9.5},
    {nome: "Domingo", cantor: "Titãs", ano: 1995, nota: 7.5},
    {nome: "Ouro de Minas", cantor: "Roupa Nova", ano: 2001, nota: 8},
    {nome: "Como estão vocês", cantor: "Titãs", ano: 2003, nota: 7},
    {nome: "Troco Likes", cantor: "Thiago Iorc", ano: 2015, nota: 4.5},
    {nome: "Dois", cantor: "Legião Urbana", ano: 1986, nota: 10},
    {nome: "Radiola", cantor: "Skank", ano: 2004, nota: 6.5},
    {nome: "Roupa acústico", cantor: "Roupa Nova", ano: 2004, nota: 9},
    {nome: "Umbilical", cantor: "Thiago Iorc", ano: 2011, nota: 3.5},
    {nome: "Barulhinho bom", cantor: "Marisa monte", ano: 1996, nota: 7.5}
];
```

Atividades

1. Gere uma lista de objetos com o nome e ano de todos os álbuns da Marisa Monte
2. Gere uma lista contendo o nome de todos os cantores e álbuns. A lista deve conter só os textos, não objetos.
3. Calcule a média da nota dos discos anteriores ao ano de 2005
4. Gere uma lista contendo o nome de todos os cantores, sem repetições
5. Gere uma lista contendo a quantidade de álbuns que cada cantor possui

Atividades

- Substitua o nome do cantor da lista anterior pelo seu objeto correspondente, presente na lista abaixo:

```
const cantores = [
    {nome: "Vinícius de Moraes", estilo: "MPB"},
    {nome: "Rita Lee", estilo: "Rock"},
    {nome: "Marisa monte", estilo: "MPB"},
    {nome: "Legião Urbana", estilo: "Rock"},
    {nome: "Titãs", estilo: "Rock"},
    {nome: "Roupa Nova", estilo: "Pop rock"},
    {nome: "Thiago Iorc", estilo: "Nova MPB"},
    {nome: "Skank", estilo: "Pop rock"}
];
```

Atividades

6. Crie a função justDate que recebe uma data e retorna a mesma data, mas com os campos de tempo zerados
7. Crie as funções de comparação de datas: before e after. Adicione um parâmetro opcional `inclusive` com valor padrão false que permite considerar também a própria data.
8. Crie a função between que recebe uma data, uma data de inicio, outra de fim e uma terceira data. Teste se a data está no meio desse intervalo. Adicione um objeto desestruturado opcional no quarto parâmetro para permitir os parâmetros opcionais inclusiveStart e inclusiveEnd
9. Crie uma função que recebe uma data inicial, um número *n* e um intervalo de tempo. Ela deve retornar uma lista, contando as *n* próximas datas considerando esse intervalo de tempo.

Atividades

10. Crie uma lista de álbuns ordenada por ano. Não altere a lista original de álbuns
11. Crie a função `paginador` que recebe uma lista e um tamanho de página. Ela deve retonar outra função que quando chamada com um número de página, retorne apenas os elementos daquela página

```
let pagina = paginador(albuns, 3);
console.log(pagina(1));
```

Atividades

12. Crie a função `media`, que recebe uma lista e opcionalmente um nome de campo.
    - Caso o nome de campo seja fornecido, calcule a média dos valores desse campo
    - Caso não seja, faça a média utilizando os próprios elementos da lista

- Exemplo:

```
let avg = media(albuns, "nota");
let avg2 = media([1,2,3,4,5]);
```