o p e r a d o r e s

THIAGO DELGADO PINTO
thiago_dp (at) yahoo (dot) com (dot) br

versão: 2020.01.20

acesso a propriedades

fornecem acesso a propriedades de objetos

notação de ponto

```
objeto.propriedade
```

notação de colchetes

```
objeto[ "propriedade" ]
```

acesso a propriedades – exemplo

```
var obj = {
  nome: "Ana",
  telefone: { ddd: "22", numero: "999887766" }
};

console.log( obj.nome ); // Ana
console.log( obj[ "nome" ] ); // Ana
console.log( obj.fake ); // undefined
console.log( obj[ "fake" ] ); // undefined
console.log( obj.telefone.ddd ); // 22
console.log( obj[ "telefone" ].ddd ); // 22
console.log( obj[ "telefone" ][ "ddd" ] ); // 22
console.log( obj.fake.bla ); // Error
```

criação de objetos

operador **new**

exemplo

```
function Telefone( ddd, numero ) {
  this.ddd = ddd;
  this.numero = numero;
}
var tel = new Telefone( '22', '33445566' );
console.log( tel );
```

acesso ao pai de uma classe ou objeto

operador **`super`**

ES6+

deve ser usado em construtores para invocar a implementação existente na classe pai, se houver

deve ser a primeira linha de código do construtor

pode ser usado em um método, caso se queira invocar uma implementação de método da classe pai

acesso ao pai de uma classe ou objeto

exemplo 1:

```
class Poligono {
  constructor( altura, largura ) {
    this.altura = altura; this.largura = largura;
    this.nome = "Polígono";
  }
  nome() { return this.nome; }
}
class Quadrado extends Poligono {
  constructor( largura ) {
    super( largura, largura );
    this.nome = "Quadrado";
  }
}
```

acesso ao pai de uma classe ou objeto

exemplo 2:

```
class Animal {
  constructor( nome ) {
     this.nome = nome;
     this.kmPorHora = 0;
  }
  correr( kmPorHora ) {
   this.kmPorHora = kmPorHora;
   console.log( `${this.nome} está
     correndo a ${kmPorHora}.`
   );
  }
  parar() {
    this.kmPorHora = 0;
    console.log( `${this.nome} está
      parado.` );
  }
}
```

```
class Coelho extends Animal {
  esconder() {
     console.log(
      `${this.nome} está escondido.` );
  }
  parar() {
    super.parar();
    this.esconder();
  }
}
```

extração de itens de array

operador **[ ]**

exemplos:

```
const [ n1, n2 ] = [ 1, 2, 3, 4, 5 ]; // n1=1, n2=2
const [ , n2, , , n5 ] = [ 1, 2, 3, 4, 5 ]; // n2=2, n5=5
```

extração de itens de array

outro exemplo:

```
// Espera um array e define os dois primeiros elementos
function imprimeDoisPrimeiros( [ n1, n2 ] ) {
  console.log( n1, n2 );
}

const numeros = [ 10, 20, 30, 40, 50 ];
imprimeDoisPrimeiros( numeros ); // 10, 20
```

extração de atributos de um objeto

operador **{ }**

exemplos:

```
const tel = { ddi: '+51', ddd: '22', num: '999887766' };
const { ddd } = tel; // extrai a propriedade "ddd"
console.log( ddd ); // 22
const { ddi, num } = tel;
console.log( num ); // 999887766
```

extração de atributos de um objeto

outro exemplo:

```
// Espera um objeto com a propriedade "num"
function imprimeNum( { num } ) {
  console.log( num );
}

const tel = { ddi: '+51', ddd: '22', num: '999887766' };
imprimeNum( tel );
```

espalhamento/resto (*spread/rest*)

operador **...**

ES6+

permite desestruturar um objeto ou um array → *spread*
permite indicar o restante de um objeto ou array → *rest*

pode ser usado dentro de

- chamadas de função
- declarações de arrays
- declarações de objetos

espalhamento/resto – exemplo 1

```
function soma( x, y ) {
  return x + y;
}

const valores = [ 10, 20 ];

console.log(
  soma( ...valores ) // idem a fornecer 10, 20
); // 30
```

espalhamento/resto – exemplo 2

```
function soma() {
  let s = 0;
  for ( const a of arguments ) {
    s += a;
  }
  return s;
}

const valores = [ 10, 20, 30, 40, 50, 60 ]; // 210

console.log(
  soma( 5, ...valores, 70 )
); // 285
```

espalhamento/resto – exemplo 3

```
function soma( ...numeros ) {
  let s = 0;
  for ( const a of numeros ) {
    s += a;
  }
  return s;
}

const valores = [ 10, 20, 30, 40, 50, 60 ]; // 210

console.log(
  soma( 5, ...valores, 70 )
); // 285
```

espalhamento/resto – exemplo 4

```
function soma( n1, n2, ...numeros ) {
  let s = n1 + n2;
  for ( const n of numeros ) {
    s += n;
  }
  return s;
}

soma( 10, 20, 30 ); // 60
```

espalhamento/resto – exemplo 5

```
const gerentes = [ 'Maria', 'João' ];
const atendentes = [ 'Ana', 'Bia', 'Carlos' ];
const todos = [ ...gerentes, ...atendentes ];

let t = [ ...gerentes ]; // copia os gerentes
t.push( ...atendentes ); // acrescenta atendentes
```

espalhamento/resto – exemplo 6

```
const vogais = [ 'a', 'e', 'i', 'o', 'u' ];
const [ a, e, ...outras ] = vogais;
console.log( a ); // 'a'
console.log( e ); // 'e'
console.log( outras ); // [ 'i', 'o', 'u' ]

const cmyk = [ 'ciano', 'magenta', 'amarelo', 'preto' ];
const [ ...cmy, k ] = cores;
console.log( cmy ); // [ 'ciano', 'magenta', 'amarelo' ]
console.log( k ); // 'preto'
```

espalhamento/resto – exemplo 7

```
// Espera um array
function f( [ n1, n2, ...outros ] ) {
  console.log( n1, n2 );
  console.log( outros );
}

const numeros = [ 10, 20, 30, 40, 50 ];
f( numeros );
// 10 20
// [ 30, 40, 50 ]
```

espalhamento/resto – exemplo 8

```
const obj1 = {
  nome: 'Coca-cola Lata',
  estoque: 100,
  preco: 5.00
};

let obj2 = { ...obj1 }; // Copia atributos
obj1.nome = 'Pepsi Lata';

const obj3 = { ...obj1, nome: 'Fanta Lata' };
```

espalhamento/resto – exemplo 9

```
// Espera um objeto como "nome" e "preco"
// e guarda outros atributos em um objeto outros
function imprimeProduto( { nome, preco, ...outros } ) {
  console.log( nome, 'R$', preco );
  console.log( outros );
}

const obj = {
  nome: 'Coca-cola Lata', estoque: 100, preco: 5.00
};
imprimeProduto( obj );
// Coca-cola Lata R$ 5
// { estoque: 100 }
```

exercícios

1. Crie uma função *concatenar* que retorne um array resultante da concatenação de dois arrays recebidos como argumento. Use o operador de espalhamento na solução.

2. Modifique a função *concatenar* para que receba um número indefinido de arrays – sem usar *arguments* – e retorne a concatenação de todos eles.

3. Crie uma função *imprimir* que receba um array desmembrado e que imprima o primeiro e o terceiro valores desse array, sem acessar seus índices.

exercícios

4. Crie uma função *imprimirApos2* que receba um array desmembrado e que imprima todos os elementos após o segundo.

5. Crie uma função *imprimeEmail* que receba um objeto desmembrado com uma propriedade e-mail e a imprima.

6. Crie uma função *clonar* que retorne o clone de um objeto fornecido como argumento.

exercícios

7. Modifique a função *clonar* para que receba um segundo objeto como argumento. Se esse segundo objeto for fornecido, o clone retornado deve ter suas propriedades sobrescritas com as do objeto. Exemplo: clonar({ nome: "Ana", idade: 7 }, { idade:8 }) deve retornar { nome: "Ana", idade: 8 }.

8. Crie uma função cloneSemElemento que receba um array e um valor desse array. A função deve retornar um clone do array, com todos os valores, exceto aquele recebido como argumento. Na solução, utilize apenas os operadores vistos (não use funções de array).

referências usadas

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

fim

Versão 1: 2020.01.20