



BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Programação de Clientes Web

A R R O W F U N C T I O N S

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

versão: 2019.10.08

notação de seta para funções

introduzida no ES6

permite criar funções anônimas

possui sintaxe mais concisa

há diferenças de comportamento quanto ao uso de

this

arguments

new

notação com corpo

// Notação comum

```
const soma1 = function( x, y ) {  
    return x + y;  
};  
console.log( soma1( 10, 20 ) ); // 30
```

// Notação de seta com corpo

```
const soma2 = ( x, y ) => {  
    return x + y;  
};  
console.log( soma2( 10, 20 ) ); // 30
```

Sai "function".

Entra o operador "=>",
que denota uma função.

notação *sem* corpo

// Notação comum

```
const soma1 = function( x, y ) {  
  return x + y;  
};  
console.log( soma1( 10, 20 ) ); // 30
```

// Notação de seta sem corpo

```
const soma2 = ( x, y ) => x + y;  
console.log( soma2( 10, 20 ) ); // 30
```

Sai "**function**", as
chaves e **return**.

Entra o operador "**=>**",
que denota uma função.

Se não há corpo, é
admitida uma **única**
instrução, que é
retornada.

recaptulando...

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

```
( x, y ) => { return x + y; }
```

```
( x, y ) => x + y
```

outro exemplo

// Normal

```
const dobrar1 = function( valor ) { return 2 * valor; };
```

// Seta com corpo

```
const dobrar2 = valor => { return 2 * valor; }
```

// Seta sem corpo

```
const dobrar3 = valor => 2 * valor;
```



Único argumento não
requer parênteses!

É a **seta** (**=>**) que faz ambos os lados
– parâmetro(s) e conteúdo –
formar uma função.

```
const dobrar = valor => 2 * valor;
```

mais um exemplo

```
// Normal
```

```
const saudacao1 = function() { return 'Oi'; };
```

```
// Seta com corpo
```

```
const saudacao2 = () => { return 'Oi'; };
```

```
// Seta sem corpo
```

```
const saudacao3 = () => 'Oi';
```


retorno de objeto em notação sem corpo

colocar parênteses para retornar objetos
quando a notação sem corpo for usada

```
telefone => ( { "numero": telefone } )
```

para o interpretador não achar que é o corpo da função!

escopo e notação de seta

uma declaração com notação de seta aponta para o escopo na qual foi declarada

ao usar **this** internamente, esse referenciará o escopo **externo**

exemplo

em um método com notação de seta, o **this** referencia o escopo externo e não o objeto atual

método com notação de seta

com a notação de seta, **não há** amarração (*binding*) do método ao **this**

```
let pessoa = {  
  nome: "Ana",  
  getNome1() { return this.nome; },  
  getNome2: function() { return this.nome; },  
  getNome3: () => { return this.nome; }  
};  
  
console.log( pessoa.getNome1() ); // "Ana"  
console.log( pessoa.getNome2() ); // "Ana"  
console.log( pessoa.getNome3() ); // undefined
```

o **this** de **getNome3** faz referência ao **escopo externo**
no exemplo acima, em um navegador, seria o objeto **window**

bind não é aplicável

```
let pessoa = {  
  nome: "Ana",  
  getNome1() { return this.nome; },  
  getNome2: function() { return this.nome; },  
  getNome3: () => { return this.nome; }  
};  
  
// Tenta amarrar ao objeto pessoa, para que ele vire o this.  
// Porém, isso não funciona!  
pessoa.getNome3.bind( pessoa );  
  
console.log( pessoa.getNome1() ); // "Ana"  
console.log( pessoa.getNome2() ); // "Ana"  
console.log( pessoa.getNome3() ); // undefined
```

uso em eventos que usam o this

```
var button = document.getElementById( 'ok' );  
button.addEventListener( 'click', () => {  
    this.classList.toggle( 'pressionado' );  
} );
```

acima, **this** *não* será amarrado ao **botão**, mas ao escopo **externo**
o que ocasionará um erro (TypeError) ao executar

uso em funções com argumentos dinâmicos

```
let tela = {  
  mostrar1() {  
    console.log( ... arguments );  
  },  
  mostrar2: () => {  
    console.log( ... arguments ); // Erro  
  }  
};
```

```
console.log( tela.mostrar1( 100, 200, 300 ) );  
// 100, 200, 300  
console.log( tela.mostrar2( 100, 200, 300 ) );  
// Uncaught ReferenceError: arguments is not defined
```

uso em funções que se queira dar new

```
let PessoaA = function( _nome ) {  
  this.nome = function() { return _nome; }  
};
```

```
let PessoaB = ( _nome ) => {  
  this.nome = function() { return _nome; }  
};
```

```
let p1 = new PessoaA( "Ana" );  
console.log( p1.nome() ); // Ana
```

```
let p2 = new PessoaB( "Ana" );  
// ^ Uncaught TypeError: PessoaB is not a constructor  
console.log( p2.nome() );
```

quando **não** usar

em **métodos** que fazem uso do **this**

em **funções** que fazem uso do **this**, como *callbacks* de eventos

em **funções** que possuam **argumentos dinâmicos**

em **funções** que se queira dar **new**



CEFET/RJ – CAMPUS NOVA FRIBURGO, RJ
BACHARELADO EM SISTEMAS DE INFORMAÇÃO
<http://bsi.cefet-rj.br>

fim

Versão 1: 2019.10.08



Licença Creative Commons 4

ESTE MATERIAL PERTENCE AO PROFESSOR THIAGO DELGADO PINTO
E ESTÁ DISPONÍVEL SOB A LICENÇA CREATIVE COMMONS VERSÃO 4.
AO SE BASEAR EM QUALQUER CONTEÚDO OU FORMATO DELE, POR
GENTILEZA, CITE-O.