O básico do ECMAScript 6

um e-book ***produzido por:***
CodePrestige

Agradecimentos

Seja muito bem-vindo!

Há muitos e muitos anos que o JavaScript não ganhava melhorias relevantes, mas isso finalmente mudou com a chegada do ES6. Aqui você aprenderá o básico sobre as novidades que chegaram a linguagem e que são fundamentais para o entendimento das melhorias que já estão chegando com a ES7, ES8 e as próximas que virão.

Agora, sem mais delongas, boa leitura e bons códigos!

Diego Martins de Pinho
Cofundador da Code Prestige

Introdução

O profissional de tecnologia que deseja se manter relevante no mercado precisa estar antenado às mudanças. Uma das grandes mudanças nos últimos anos foi a chegada da sexta versão da especificação ECMAScript, responsável por descrever as funcionalidades da linguagem JavaScript.

Com ela, o JavaScript nunca esteve tão convidativo para iniciantes e confortável de se trabalhar. Aqui você aprenderá o básico sobre as principais mudanças e sairá sabendo quais aspectos deve aprofundar nos estudos.

Preparado para entrar de cabeça no futuro do JavaScript?

Sumário

- Métodos para Arrays
- Arrow Functions
- Valores padrões para parâmetros
- Operadores Rest e Spread
- Templates Literais
- Laço de repetição for...of
- Const e let
- Desestruturamento de objetos e Arrays
- Classes
- Geradores
- Aprenda mais sobre a ECMAScript 6

Métodos para Arrays

Agora temos uma série de métodos auxiliares para iterar Arrays, cada um com um objetivo diferente:

Antes

```
for (var i = 0; i < lista.length; i++) {
  // corpo da lógica que queremos
}
```

Agora

Objetivo
iterar todos os registros

```
lista.forEach(function(item) {
    // corpo
});
```

Objetivo
iterar todos os registros e fazer algo com eles

```
lista.map(function(item) {
    return // corpo;
});
```

Objetivo
filtrar a lista de acordo com algum critério

```
lista.filter(function(item) {
    return // corpo do critério;
});
```

Objetivo
encontrar um registro de acordo com um critério

```
lista.find(function(item) {
    return // corpo do critério;
});
```

Objetivo
verifica se todos os itens obedecem a um dado critério

```
lista.every(function(item) {
    return // corpo do critério;
});
```

Objetivo
verifica se pelo menos um item obedece a dado critério

```
lista.some(function(item) {
    return // corpo do critério;
});
```

Arrow Functions

Arrow Functions são uma notação para escrever funções. A definição de uma função desse tipo consiste de uma lista de parâmetros (...), seguido de uma *fat arrow* (=>) e o corpo da função.

Antes

```
var soma = function (a,b) {
  return a + b;
}
```

Agora

```
var soma = (a,b) => a + b;
```

Quando só houver um parâmetro, o uso dos parênteses é desnecessário

Quando só há uma linha, não é necessário o return nem as chaves!

O comportamento das *Arrow Functions* com o *this* (contexto de execução) é diferente do das funções normais. Cada função no JavaScript define seu próprio contexto de *this*, mas as *Arrow Functions* capturam o this do seu contexto delimitador. Isso evita que façamos gambiarras.

Valores padrões para parâmetros

O ECMAScript 6 agora nos permite atribuir valores padrões aos parâmetros de nossas funções.

Antes

```
function multiplicaPor(num, multi) {
  if(multi !== undefined) return num * multi;
  return num * 2; // valor padrão
}
```

Agora

```
function multiplicaPor(num, multi = 2) {
  return num * multi;
}
```

Se na chamada do método nenhum valor for especificado, o valor padrão é assumido!

Operadores Rest e Spread

Agora lidamos com os parâmetros de uma função muito mais tranquilamente utilizando os três pontos (...), que dependendo de sua localização, atua como operador rest ou spread.

Rest

Ele encapsula todos os parâmetros que recebe dentro de um Array

```
function somaTudo(...params) {
  return params.reduce((soma, param) => {
      return soma + param;
  }, 0);
}
```

Spread

```
var lista = [1,2,3,4,5];
somaTudo(...lista); // 15
```

Os itens são interpretados um a um e enviados para o método

Templates literais

Lidamos agora com Strings de maneira muito mais eficiente utilizando os templates literais. Basta utilizarmos a crase (`) e o ${ } para interpolar valores!

Antes

```
var nome = “Luiz”;
var sobrenome =
“Augusto”;

console.log(“Bom dia, ” +
nome + “ ” + sobrenome);

// Bom dia, Luiz Augusto
```

Agora

```
var nome = “Luiz”;
var sobrenome =
“Augusto”;

console.log(`Bom dia,
${nome} ${sobrenome}`);

// Bom dia, Luiz Augusto
```

O valor da variável (ou expressão) é interpretado e jogado dentro da String

Laço de repetição for...of

Com este tipo de laço, ficou muito mais fácil iterar objetos que sejam iteráveis:

Antes

```
var lista = [1,2,3,4,5];
for(var i = 0; i < lista.length; i++) {
  console.log(lista[i]);
}
```

Agora

```
var lista = [1,2,3,4,5];
for(var numero of lista) {
  console.log(numero);
}
```

A cada iteração, o item da lista é recuperado e jogado para a variável

Const e Let

Com o ECMAScript 6, não declaramos mais nossas variáveis com o **var,** agora utilizamos as palavras chave: **const** e **let**. Esse novo tipo contém escopo de bloco, ao invés de escopo de função como tínhamos antigamente.

let

```
var a = 2;
{
  let a = 3;
  console.log(a); // 3
}
console.log(a); // 2
```

Como o escopo é de bloco, as variáveis declaradas neste espaço não são vistas fora

const

```
const nome = “Luiz”;

// vamos tentar alterar
nome = “Kiko”;

// TypeError
```

Os valores deste tipo de variáveis não podem ser alterados!

Desestruturamento de objetos e arrays

Desestrututamento ajuda a evitar a necessidade de variáveis temporárias quando lidamos com objetos e arrays.

Considerando...

```
var livro = {
  titulo: “ECMAScript 6”,
  autor: “Diego Pinho”,
  editora: “Casa do Código”,
  ISBN: “128301212”
}
```

Antes

```
var titulo = livro.titulo;
var autor = livro.autor;
```

Agora

```
let {titulo,autor} = livro;
```

As propriedades com estes nomes são extraídas e variáveis locais com o mesmo nome é criada

Classes

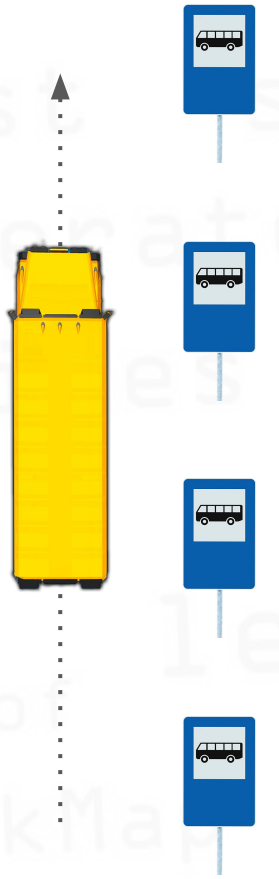
A herança por prototipagem sempre foi bem confuso de entender, mas agora o JavaScript tem suporte à classes assim como já conhecemos em outras linguagens de programação. As classes são sintaxe de açúcar, ou seja, por debaixo dos panos, tudo continua como antes, mas temos um nova abstração de alto nível para trabalhar.

```
class Fiat extends Carro {

  constructor(cor, modelo) {
    super(cor,modelo);
  }

  andar() {
    console.log(“vrum vrum”);
  }

  static mostrarModelo() {
    console.log(this.modelo);
  }
}
```

Temos os métodos estáticos que podem ser invocados sem a necessidade de instanciar a classe!

Geradores

Funções geradoras são um novo tipo de função que nos permite gerar quantos valores forem necessários, ou seja, podemos executá-la, interromper a sua execução, e retomá-la.



O asterisco descreve uma função como geradora

O yield é a palavra chave que define o ponto de parada da função

```
function* percorrerLinha477() {
  console.log('Passei pela rua 1');
  yield 'Parada 1';
  console.log('Passei pela rua 2');
  yield 'Parada 2';
  console.log('Passei pela rua 3');
  yield 'Parada 3';
  console.log('Passei pela rua 4');
  yield 'Fim da linha';
}

const linha = percorrerLinha477();
for (let parada of linha) {
  console.log(parada);
}
```

```
Passei pela rua 1
Parada 1
Passei pela rua 2
Parada 2
Passei pela rua 3
Parada 3
Passei pela rua 4
Fim da linha
```

Aprenda mais sobre o ECMAScript 6!

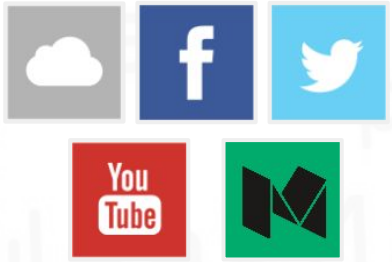
Gostou das novidades? E olha que isso é só o começo! Confira o nosso livro e o nosso curso para aprender com detalhes como funcionam essas melhorias e entre de cabeça no futuro do JavaScript!

Somente no livro/curso:

- Fundamentação teórica
- Aplicação em casos reais do dia a dia do desenvolvedor
- Exercícios práticos e teóricos para fixação
- Maps, WeakMaps, Set e Weaksets
- Iteradores e iteráveis
- Melhorias em objetos literais
- Modularização
- Operações assíncronas com Promises
- Metaprogramação com proxies
- E muito mais!

CodePrestige

Ensino de programação à distância



/CodePrestige

Confira outros e-books, vídeos e cursos nas nossas redes sociais!