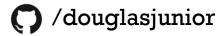
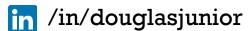
CRIAÇÃO DE OBJETOS EM JAVASCRIPT







ouglasjunior.me

massifrroma@gmail.com



Slides: https://git.io/vbU3N





- bind / this
- Object prototype
- new
- __proto__ vs prototype
- Object.create()
- class
- Referências





- Não existe um forma correta ou incorreta de se criar objetos em JavaScript.
- Diferente de linguagens como Python ou Java, não existe um site para você acessar e baixar o JavaScript.
- Existem ao menos 4 grandes motores de interpretação de JavaScript no mercado, criados por Google, Microsoft, Apple e Mozilla.
- O importante é entender como a orientação à objetos funciona em JavaScript, e então escolher o melhor para você, seu time ou sua empresa.



THIS

- Em JavaScript o this tem um comportamento diferente da maioria das linguagens orientadas à objeto.
- Na maioria dos casos o this representa o objeto que invocou a função, e não onde a função foi declarada.

THIS



```
const dog = {
   sound: "woof",
   talk: function() {
     console.log("Says " + this.sound);
   }
}

dog.talk(); // Says woof

const dogTalk = dog.talk;

dogTalk(); // Says undefined
```

Exemplo do this



BIND

- Seu objetivo principal é definir quem será o this quando a função for invocada.
- A função bind () cria uma nova função (função vinculada) com o mesmo corpo da função original.
- Ao chamar uma função vinculada, geralmente resulta na execução da função original.
- A função bind () também aceita argumentos padrões para fornecer à função de destinado quando vinculada à chamada.





Syntax

fun.bind(thisArg[, arg1[, arg2[, ...]]])

Parámetros

thisArg

O valor a ser passado com este parâmetro para a função de destino quando a função vinculada é chamada. O valor é ignorado se a função ligada é construída usando o new operador.

arg1, arg2, ...

Argumentos para preceder argumentos fornecidos para vincular a função ao invocar a função de destino.

BIND



```
const dog = {
  sound: "woof",
  talk: function() {
    console.log("Says " + this.sound);
  }
}

dog.talk(); // Says woof

const dogTalk = dog.talk.bind(dog);

dogTalk(); // Says woof
```

Exemplo do bind



PROTOTYPE

- A propriedade Object. prototype aponta para o Object protótipo do objeto em questão.
- Quase todos os objetos em JavaScript são instâncias de Object.
- Um objeto instanciado herda propriedades e funções do Object.prototype, embora essas propriedades possam ser sobrescritas.





```
const animal = {
  talk: function() {
    console.log('Says ' + this.sound);
animal.talk(); // Says undefined
const dog = {
  sound: 'Woof',
Object.setPrototypeOf(dog, animal);
dog.talk(); // Says Woof
```



NEW

- O operador new cria uma instância de um tipo de objeto definido pelo usuário ou de um dos tipos de objeto construídos que possuem uma função de construtor.
- Criar um objeto definido pelo usuário requer duas etapas:
 - 1. Definir o tipo de objeto escrevendo uma função.
 - 2. Criar uma instância do objeto com new.
- Para definir um tipo de objeto, crie uma função para o tipo de objeto que especifica seu nome e propriedades. Um objeto pode ter uma propriedade que é ele próprio outro objeto.





```
function Person(name) {
   this.name = name;
}
Person.prototype.sayYourName = function() {
   console.log("My name is " + this.name);
}

const myPerson = new Person("Douglas");

myPerson.sayYourName(); // My name is Douglas
```

Exemplo do operador new



PROTO VS PROTOTYPE

- A propriedade __proto__ de Object.prototype é uma propriedade de acesso (uma função getter e uma função setter) que expõe o interior [[Prototype]] (um objeto ou nulo) do objeto através do qual ele é acessado.
- O uso de __proto__ é controverso e foi desencorajado. Nunca foi originalmente incluído na especificação do EcmaScript, mas os navegadores modernos decidiram implementá-lo de qualquer maneira.
- Em outras palavras, __proto__ aponta para o objeto herdado.



PROTO VS PROTOTYPE

```
const animal = {
  talk: function() {
    console.log('Says ' + this.sound);
const dog = {
  sound: 'Woof',
Object.setPrototypeOf(dog, animal);
console.log(dog.__proto__) // { talk: [Function: talk] }
console.log(dog.__proto__ === animal) // true
```



OBJECT.CREATE()

• O método Object.create() cria um novo objeto com o objeto e as propriedades do prototype especificado.

OBJECT.CREATE()



Sintaxe

Object.create(proto[, propertiesObject])

Parâmetros

proto

O objeto que deve ser o protótipo do objeto recém-criado.

propertiesObject

Opcional. Se especificado e não undefined, um objeto cuja as propriedades próprias enumeráveis (isto é, aquelas propriedades definidas sobre si mesmo, e não propriedades enumeráveis ao longo da sua cadeia protótipa) especificam os nomes das propriedades a serem adicionadas ao objeto recém-criado, com os nomes das propriedades correspondentes. Essas propriedades correspondem ao segundo argumento de Object.defineProperties().

Retorno

Um novo objeto com o protótipo de objeto e propriedades especificadas.

Exceções

16





```
const animal = {
  talk: function() {
    console.log('Says' + this.sound);
  }
}

const dog = Object.create(animal);
dog.sound = 'Woof';

dog.talk(); // Says Woof
```

Exemplo de Object.create()



CLASS

- A expressão class é uma forma de definir uma classe no ECMAScript 2015.
- Semelhante às expressões de função, as expressões de classe podem ser nomeadas ou sem nome. Se nomeado, o nome da classe é local apenas para o corpo da classe.
- As classes de JavaScript usam herança baseada em prototype.

CLASS

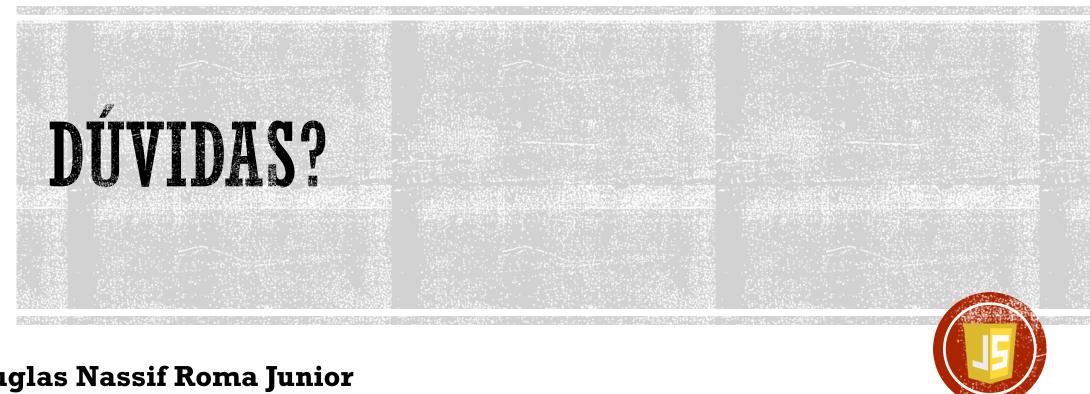


```
class Animal {
  talk() {
    console.log('Says ' + this.sound);
class Dog extends Animal {
  constructor() {
    super();
    this.sound = 'Woof';
const dog = new Dog();
dog.talk(); // Says Woof
```





MDN Web Docs - https://developer.mozilla.org/



Douglas Nassif Roma Junior

- /douglasjunior
- /in/douglasjunior
- douglasjunior.me
- massifrroma@gmail.com

Slides: https://git.io/vbU3N