Seção 06: Javascript Objetos e prototypes (avançado)

* Aula 70 Revisando Obetos

const pessoa = {

    nome: 'Luiz',

    sobrenome: 'Otávio'

};

console.log(pessoa['nome']);

console.log(pessoa.sobrenome);

forma de chamar

const pessoa = {

    nome: 'Luiz',

    sobrenome: 'Otávio',

    idade: 30

};

// formas de chamar um atributo

console.log(pessoa['nome']);

console.log(pessoa.sobrenome);

// chave dinamica

const chave = 'idade';

console.log(pessoa[chave]);

console.log('#############################################################');

// new ArraY(); ou []

const pessoa1 = new Object();

pessoa1.nome = 'Luiz';

pessoa1.sobrenome = 'Otávio';

pessoa1.falarNome = function(){

    return `${this.nome} está falando seu nome.`;

};

pessoa1.idade = 30;

pessoa1.getDataNascimento = function() {

    const dataAtual = new Date();

    return dataAtual.getFullYear() - this.idade;

};

const pessoa2 = {

    nome: 'Luiz',

    sobrenome: 'Otávio'

};

// delete pessoa2.nome;

// console.log(pessoa1);

// console.log(pessoa2);

// pessoa1.falarNome()

// console.log(pessoa1.getDataNascimento());

for (let chave in pessoa1) {

    console.log(chave, pessoa1[chave]);

}

console.log('#############################################################');

// factory functions / Constructor functions // Classes

// Factory function

function criaPessoa(nome, sobrenome) {

    return {

        nome,

        sobrenome,

        nomeCompleto() {

            return `${this.nome} ${this.sobrenome}`

        }

    };

};

const p1 = criaPessoa('Luiz', 'Otávio');

console.log(p1.nomeCompleto());

// Constructor functions

function Pessoa(nome, sobrenome) {

    this.nome = nome,

    this.sobrenome = sobrenome;

    Object.freeze(this);  // Nenhum objeto poderar ser alterado seu valor ou criado

}

// {} <- this -> retornando esse this

const pe1 = new Pessoa('Maria', 'Joaquina');

Object.freeze(pe1);   // para travar o objeto

pe1.nome = 'Monica';  // iria aterar o objeto caso não estivesse travado

const pe2 = new Pessoa('João', 'Pedro');

console.log(pe1);

console.log(pe2);

* Aula 71 Object.defineProperty() e Object.defineProperties()

// defineProperty - defineProperties

function Produto(nome, preco, estoque) {

    this.nome = nome;

    this.preco = preco;

    Object.defineProperty(this, 'estoque', {

        enumerable: true,     // mostra a chave

        value: estoque,       // valores que vai receber

        writable: false,      // controla se o valor pode ou não ser alterado

        configurable: false   // se pode reconfigurar a propriedade (ex: delete p1.estoque)

    });

}

const p1 = new Produto('Camiseta', 20, 3);

console.log(p1);

console.log(Object.keys(p1)); [ 'nome', 'preco', 'estoque' ]

para mostrar as chaves do seu objeto

Se a opção de enumerable for false, essa chave não aparecerá.

 Object.defineProperty(this, 'estoque', {

        enumerable: false,     // mostra a chave

    });

console.log(Object.keys(p1)); [ 'nome', 'preco' ]

essa foi a defineProperty, que serve para pegar so um valor.

Agora vamos ver a defineProperties para pegar mais de uma propriedade

function Produto2(nome, preco, estoque) {

    Object.defineProperties(this, {

        nome: {

            enumerable: true,

            value: nome,

            writable: true,

            configurable: true

        },

        preco: {

            enumerable: false,

            value: preco,

            writable: true,

            configurable: true

        },

        estoque: {

            enumerable: true,

            value: estoque,

            writable: true,

            configurable: true

        }

    });

}

const p2 = new Produto2('Camiseta', 20, 3);

console.log(p2);

você vai interando com cada chave do seu objeto particularmente, um a um

* Aula 72 Getters e Setters

// defineProperty -> Getters e Setters

// Get - Obter valor

// Set - setar esse valor

function Produto(nome, preco, estoque) {

    this.nome = nome;

    this.preco = preco;

    let estoquePrivado = estoque

    Object.defineProperty(this, 'estoque', {

        enumerable: true,

        configurable: false,

        get: function() {    // Pegar e mostrar o valor

            return estoquePrivado;

        },

        set: function(valor) {

            if(typeof valor !== 'number') {

                console.log('Bad value');

                return

            }

            estoquePrivado = valor;

        }

    });

}

const p1 = new Produto('Camisa', 20, 3);

//console.log(p1);

p1.estoque = 500;

console.log(p1.estoque);

agora com function factory

// Function Factory get e set

function criaProduto(nome) {

    return {

        get nome(){

            return nome;

        },

        set nome(valor){

            nome = valor;

        }

    };

}

const p2 = criaProduto('camiseta');

console.log(p2.nome);

* Aula 73 Metodos uteis para objetos

/\*

Object.values               // Mostra somente seus valores

Object.entries              // Mostra as chaves e os valores

Object.getOwnPropertyDescriptor(o, 'prop') // Mostras suas propriedades

Object.assign(des, any)     // Copiar um objeto

...(spread)                 // Espalha os valores do objeto

// Já vimos

Object.keys (retorna as chaves)

Object.freeze (congela o objeto)

Object.defineProperties (define várias propriedade)

Object.defineProperty (define uma propriedade)

\*/

const produto = {nome: 'Produto', preco: 1.8};

const caneca = {...produto, material: 'porcelana'};

console.log(produto);

console.log(caneca);

modelo que copia uma array.

const produto = {nome: 'Produto', preco: 1.8};

//const caneca = {...produto};

const caneca = Object.assign({}, produto, {material: 'porcelana'});

caneca.nome = 'Outro nome';

caneca.preco = 2.5;

outra forma de copiar os valores de um objeto e criando novas chaves

console.log(Object.getOwnPropertyDescriptor(produto, 'nome'));

{

  value: 'Produto',

  writable: true,

  enumerable: true,

  configurable: true

}

Mostra as propriedades da chaves.

console.log(Object.entries(produto));

[ [ 'nome', 'Produto' ], [ 'preco', 1.8 ] ]

mostra as chaves e os valores

// Desestruturando um objeto

for(let valor of Object.entries(produto)){

    console.log(valor[0], valor[1]);

}

nome Produto

preco 1.8

Para desestruturar um objeto

* Aula 74 Prototypes

// Construtora -> molde (classe)

function Pessoa(nome, sobrenome) {

    this.nome = nome;

    this.sobrenome = sobrenome;

    this.nomeCompleto = () => `${this.nome} ${this.sobrenome}`;

}

// instância

const pessoa1 = new Pessoa('Luiz', 'O.') // <- Pessoa = Função construtora

const pessoa2 = new Pessoa('Maria', 'A.') // <- Pessoa = Função construtora

console.log(pessoa1);

console.log(pessoa2);

Pessoa {

  nome: 'Luiz',

  sobrenome: 'O.',

  nomeCompleto: [Function (anonymous)]

}

Pessoa {

  nome: 'Maria',

  sobrenome: 'A.',

  nomeCompleto: [Function (anonymous)]

}

Para cada pessoa que for criada, ele ira criar nome, sobrenome e nomeCompleto.

Vimos que nomeCompleto se repete em toda vez que for chamado, para resolver isso, utilizamos o prototype.

Pessoa.prototype.estouAqui = 'Estou dentro do prototype';

E criado dentro do \_\_proto\_\_, o método estouAqui. Agora todos os objetos que vierem de Pessoa, teram esse método de estouAqui.

Pessoa.prototype === pessoa1.\_\_proto\_\_

function Pessoa(nome, sobrenome) {

    this.nome = nome;

    this.sobrenome = sobrenome;

    this.nomeCompleto = () => 'ORIGINAL ' +this.nome +' ' +this.sobrenome;

}

Pessoa.prototype.nomeCompleto = function() {

    return this.nome + ' ' + this.sobrenome;

};

const pessoa1 = new Pessoa('Luiz', 'O.')

console.log(pessoa1.nomeCompleto());

Aqui estou dizendo que Pessoa terá um método em seu prototype chamado de nomeCompleto, caso o objeto que foi criado tenha em seus atributos o mesmo nome que o de seu pai o proto, o que valerá será o do filho, do objeto que foi criado.

Então primeiro ele ira procurar no objeto que foi criado, caso não encontre ira procurar no pai o \_\_photo\_\_.

function Pessoa(nome, sobrenome) {

    this.nome = nome;

    this.sobrenome = sobrenome;

}

Pessoa.prototype.nomeCompleto = function() {

    return this.nome + ' ' + this.sobrenome;

};

const pessoa1 = new Pessoa('Luiz', 'O.')

const pessoa2 = new Pessoa('Maria', 'A.')

console.log(pessoa1);

console.log(pessoa2);

console.log(pessoa1.nomeCompleto());

console.log(pessoa2.nomeCompleto());

Pessoa { nome: 'Luiz', sobrenome: 'O.' }

Pessoa { nome: 'Maria', sobrenome: 'A.' }

Luiz O.

Maria A.

Dessa maneira foi resolvida a questão de repetição, deixando os valores que são variável como atributos, e os valores que se repetem foram jogado em um so lugar no prototype principal.

* Aula 75 Manipulando prototypes

// new Object -> Object.prototype

const objA = {

    chaveA: 'A'

    // \_\_proto\_\_: Object.prototype

};

const objB = {

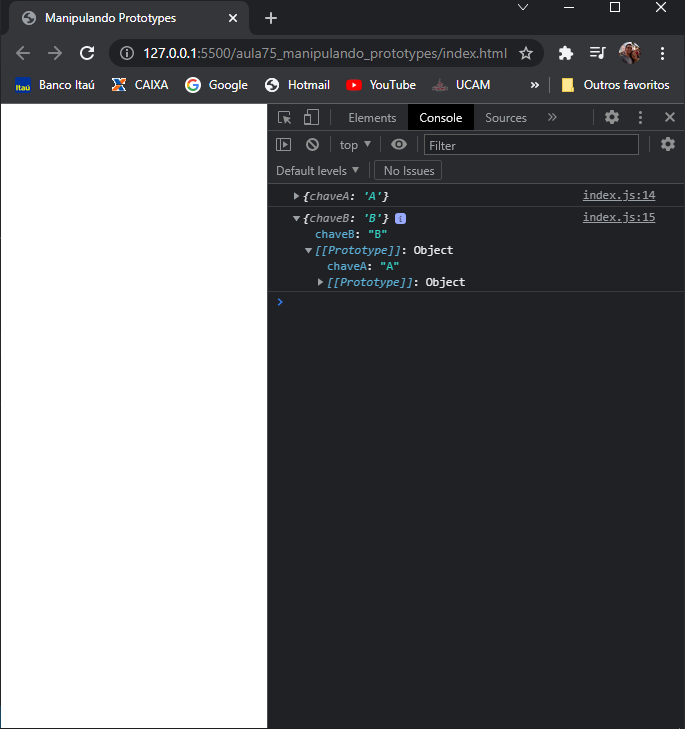
    chaveA: 'B'

    // \_\_proto\_\_: objA

};

Object.setPrototypeOf(objB, objA);

Código para que o prototype do objB seja lincando ao do objA, feito isso, objB terá seus atributos normais e dentro de seu \_\_proto\_\_, terá os atributos de objA



Com isso você vai relacionando seus prototypes com diversos outros protos

function Produto(nome, preco) {

    this.nome = nome;

    this.preco = preco;

}

Produto.prototype.desconto = function(percentual) {

    this.preco = this.preco - (this.preco \* (percentual / 100));

};

Produto.prototype.aumento = function(percentual) {

    this.preco = this.preco + (this.preco \* (percentual / 100));

};

const p1 = new Produto('camiseta', 50);

// p1.desconto(100);

p1.aumento(100)

console.log(p1); Produto { nome: 'camiseta', preco: 100 }

modelo de código, que recebe o método que estão em Produtos.

const p1 = new Produto('camiseta', 50);

// Literal

const p2 = {

    nome: 'Caneca',

    preco: 15

};

Object.setPrototypeOf(p2, Produto.prototype);

Setando o p2, que e um objeto literal a ter um o prototype de Produtos

const p3 = Object.create(Produto.prototype);

aqui estamos criando um novo objeto e setando o seu prototype para ser o de Produto

const p3 = Object.create(Produto.prototype, {

    preco: {

        writable:true,

        configurable: true,

        enumerable: true,

        value: 99

    },

    tamanho: {

        writable:true,

        configurable: true,

        enumerable: true,

        value: 42

    }

});

Aqui