20

Proxy Intercepta métodos?

Temos a seguinte declaração de classe:

```
class Pessoa {

    constructor(nome) {
        this._nome = nome;
    }

    get nome() {
        return this._nome;
    }

    set nome(nome) {
        this._nome = nome;
    }

    grita(frase) {
            return `${this._nome} grita ${frase}`;
    }
}
```

Criando uma instância e chamando o método `grita` :

```
let pessoa = new Pessoa('Barney');
pessoa.grita('Olá');
```

E se quisermos interceptar a chamada do método `grita` ? A má notícia é que toda proxy criada, por padrão, não esta preparada para interceptar métodos (getters e setters são exceções a este problema). Essa limitação ocorre porque sempre que um método de um objeto (que não deixa de ser uma propriedade que armazena uma função) é chamado, primeiro é realizado uma operação de leitura (get, do nosso handler da proxy) e depois os parâmetros são passados através de `Reflect.apply` . O problema é que, como o método é interceptado pelo `get` do handler passado para a proxy, não temos acesso aos seus parâmetros. E agora?

Uma solução é implementar o seguinte código:

```
let pessoa = new Proxy(new Pessoa('Barney'), {

        get(target, prop, receiver) {
            if(prop == 'grita' && typeof(target[prop]) == typeof(Function)) {
         // essa função retornada irá substituir o método 'grita' no proxy!!! Ou seja,
                return function() {
                    console.log(`Método chamado: ${prop}`);
                    // Quando usarmos Reflect.apply, Reflect.get e Reflect.set precisa
                    // arguments é uma variável implícita que dá acesso à todos os par
                    return Reflect.apply(target[prop], target, arguments);
                }
            }
```

```
            // só executa se não for função
            return Reflect.get(target, prop, receiver);
        }
    });

    pessoa.grita('Olá');
```

No código acima, verificamos se a propriedade que está sendo acessada é uma função através de `typeof(target[prop]) == typeof(Function))`. Se for, trocaremos o valor da propriedade (nosso método) por outra função, e, essa sim, executa nosso código antes de o método ser executado.

Sobre o código anterior, qual o papel de `arguments` ?

Selecione uma alternativa

A Não tem papel algum, ela não foi declarada no código e por isso teremos um erro.

B A variável `arguments` é uma variável implícita que pode ser acessada em métodos ou funções. Ele se comporta como um array onde cada posição equivale ao parâmetro que foi passado para o método ou função. Existe desde o ES5!

C A variável `arguments` é uma variável implícita que pode ser acessada em métodos ou funções. Ele se comporta como um array em que cada posição equivale ao parâmetro que foi passado para o método ou função. Esse recurso foi adotado a partir do ES6.