12

Revisando Reflect.apply

Esse é mais um exercício de revisão. Não é necessário respondê-lo, no entanto praticar esse código vale a pena.

Temos dois objetos criados de maneira literal por uma questão de brevidade, mas que poderiam ser instâncias de uma classe:

```
let objeto1 = {

    nome: 'Bob'
};

let objeto2 = {

    nome: 'Leo'
}
```

Temos a seguinte função:

```
function exibeNome() {

    alert(this.nome);
}
```

O que acontecerá se chamarmos `exibeNome` ? O resultado será `undefined` , porque o `this` da função, ou seja, seu contexto não possui a propriedade `nome` .

Agora, que tal chamarmos a função `exibeNome` , mas indicando que seu contexto de execução será `objeto1` ? Vejamos:

```
Reflect.apply(exibeNome, objeto1, []); // exibe 'Bob'
```

O resultado será o alerta sendo exibido com o texto `Bob` . Podemos executar a função agora fazendo com que o seu `this` (contexto) seja `objeto2` :

```
Reflect.apply(exibeNome, objeto2, []); // exibe 'Leo'
```

Como `Reflect.apply` funciona? O primeiro parâmetro é o método ou função que desejamos invocar. O segundo parâmetro é o contexto que o método ou função adotará, ou seja, o valor que será assumido pelo `this` . Por fim, o último parâmetro é um array que contém todos os parâmetros que o método passado como primeiro parâmetro receberá. Como ele não recebe parâmetro nenhum, passamos um array vazio.

Vamos alterar nossa função para receber dois parâmetros. O primeiro será um prefixo que será adicionando no nome e o último um sufixo:

```
function exibeNome(prefixo, sufixo) {
```

```
        alert(prefixo + this.nome + sufixo);
    }
```

Agora, vamos chamar o método através de `Reflect.apply` :

```
Reflect.apply(exibeNome, objeto1, ['(', ')']); // exibe '(Bob)'
```

Veja que agora estamos passando dois parâmetros para o método.