14

Function.apply vs Reflect.apply

Aprendemos a alterar o contexto de execução de uma função/método através de `Reflect.apply`. Vejamos um trecho de código que faz isso:

```
class ListaNegociacoes {

    constructor(contexto, armadilha) {
        this._negociacoes = [];
        this._armadilha = armadilha;
        this._contexto = contexto;
    }

    adiciona(negociacao) {
        this._negociacoes.push(negociacao);
        Reflect.apply(this._armadilha, this._contexto, [this]);
    }

    get negociacoes() {
        return [].concat(this._negociacoes);
    }

    esvazia() {
        this._negociacoes = [];
        Reflect.apply(this._armadilha, this._contexto, [this]);
    }
}
```

O artefato `Reflect` é algo novo do ES6, contudo poderíamos conseguir com ES5 o mesmo resultado, assim:

```
class ListaNegociacoes {

    constructor(contexto, armadilha) {
        this._negociacoes = [];
        this._armadilha = armadilha;
        this._contexto = contexto;
    }

    adiciona(negociacao) {
        this._negociacoes.push(negociacao);
        this._armadilha.apply(this._contexto, [this]);
    }

    get negociacoes() {
        return [].concat(this._negociacoes);
    }

    esvazia() {
        this._negociacoes = [];
        this._armadilha.apply(this._contexto, [this]);
    }
}
```

Veja que estamos chamando o método `apply`, mas diretamente na função atribuída à propriedade `this._armadilha`. O resultado é o mesmo quando usamos `Reflect.apply`.

Apesar de um pouco menos verbosa, a ideia é que operações como essa sejam feitas através de `Reflect`, pois é uma primeira tentativa da linguagem JavaScript de centralizar funcionalidades como vimos em um único lugar.

Achei interessante mostrar como esse procedimento é feito no ES5 para que o aluno entenda que o ES6, além de trazer muitas novidades, procura padronizar bastante coisa da própria linguagem.