02

O padrão de projeto Proxy

Transcrição

Veremos qual é a solução que nos permite manter o modelo... Começaremos retirando o `_armadilha` de `ListaNegociacoes`:

```
class ListaNegociacoes {

  constructor() {

        this._negociacoes = [];
  }
  adiciona(negociacao) {
      this._negociacoes.push(negociacao);
  }

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

  esvazia() {
      this._negociacoes = [];
  }
}
```

Como removemos o `_armadilha`, o construtor de `NegociacaoController` deixará de funcionar e descobriremos uma forma de resolver problema da View. Temos ainda outro problema com a solução que usa o `_armadilha`: se quisermos monitorar os models `Mensagem` e `Negociacoes`, teremos que abrir a classe para alterar e colocar a armadilha - mas, não faremos isto.

Existe um famoso padrão de projeto chamado **Proxy**, que de forma resumida, é "um cara mentiroso". Vimos que não é bom inserirmos armadilhas na classe, porque estaremos perdendo a reutilização do modelo e teremos que repetir em todos os modelos do sistema. No entanto, o Proxy é idêntico ao objeto que queremos trabalhar, e teremos bastante dificuldade de diferenciá-los. Nós acessamos o Proxy como se ele fosse o objeto real, este último ficará escondido dentro do outro. Nós substituímos o objeto real, que só poderá ser acessado por meio do Proxy - que pode ou não ser executado em um código arbitrário se assim definirmos.

Observe que `ListaNegociacoes` tem o métodos `adiciona()` e `negociacoes()`, que também estarão presentes no Proxy.

```
class ListaNegociacoes {

  constructor(armadilha) {

        this._negociacoes = [];
        this._armadilha = armadilha;
  }
  adiciona(negociacao) {
      this._negociacoes.push(negociacao);
      this._armadilha(this);
```

```
    }

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

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

A diferença está em que quando chamarmos o `adiciona()`, o Proxy delegará a chamada do método para o objeto encapsulado por ele. Mas ainda não temos benefícios com esta mudança. A vantagem está que colocaremos as armadilhas **entre** a chamada do Proxy e o objeto real. Toda vez que acessamos o Proxy, executaremos um código antes de chamarmos um método ou propriedade correspondente ao objeto real.

A boa notícia é que não precisamos implementar esse padrão de projeto.

A partir da versão 2015 do ECMAScript, a própria linguagem já possui um recurso de Proxy. Então, implementaremos o padrão de projeto Proxy usando o ES6.