01

Padrão de Projeto Factory

Transcrição

Você pode fazer o download (https://github.com/alura-cursos/javascript-avancado-ii/archive/aula2.zip) completo do projeto até aqui e continuar seus estudos.

Vamos voltar a trabalhar com a Proxy. Esta é uma solução interessante que não polui o modelo original, com o código para atualizar a View. Mas criamos uma forma bastante verbosa... Estamos trabalhando com dez propriedades da `Controller` com as quais queremos criar uma Proxy. Podemos criar um padrão de projeto chamado `Factory` que consiste em uma classe ser especializada em criar determinado tipo de objeto. Em seguida, vamos gerar o arquivo `ProxyFactory.js` , que ficará dentro da pasta `services` :

```
class ProxyFactory {

  static createProxy(objeto, props, acao) {
  }
}
```

Não é uma regra em si, mas podemos invocar um método estático da classe para não ter que instanciá-la. Depois, moveremos de `NegociacaoController` , o trecho referente ao Proxy, para a nova classe criada:

```
class ProxyFactory {

    static create(objeto, props, acao) {

        return new Proxy(objeto, {

            get(target, prop, receiver) {

                if(props.includes(prop) && typeof(target[prop]) == typeof(Function)) {
                    return function() {

                        console.log(`a propriedade "${prop}" foi interceptada`);
                        Reflect.apply(target[prop], target, arguments);
                        return acao(target);
                    }
                }
                return Reflect.get(target, prop, receiver);
            }
        })
    }
}
```

Dentro de `props` , temos um array no qual está a propriedade que queremos verificar. A `acao()` nos devolverá um valor. O próximo passo será importar o `ProxyFactory` no `index.html` .

```
    <script src="js/app/models/Negociacao.js"></script>
    <script src="js/app/models/ListaNegociacoes.js"></script>
    <script src="js/app/models/Mensagem.js"></script>
    <script src="js/app/controllers/NegociacaoController.js"></script>
    <script src="js/app/helpers/DateHelper.js"></script>
    <script src="js/app/views/View.js"></script>
    <script src="js/app/views/NegociacoesView.js"></script>
    <script src="js/app/views/MensagemView.js"></script>
    <script src="js/app/services/ProxyFactory.js"></script>
    <script>
        let negociacaoController = new NegociacaoController();
    </script>
```

Agora, não precisaremos mais do código do Proxy no `NegociacaoController.js`. No lugar, vamos adicionar uma `_listaNegociacoes` com o `ProxyFactory`:

```
this._listaNegociacoes = ProxyFactory.create (
    new ListaNegociacoes(),
    ['adiciona', 'esvazia'], model =>
        this._negociacoesView.update(model));

this._negociacoesView = new NegociacoesView($('#negociacoesView'));
this._negociacoesView.update(this._listaNegociacoes);
```

Dentro do `ProxyFactory.create()`, incluímos um array com as propriedades com `adiciona` e `esvazia`. Já não será necessário usar a variável `self` e, como a *arrow function* tem escopo léxico, ela entenderá que o `this` é referente a `controller`. Se testarmos no navegador, veremos que conseguimos preencher o formulário corretamente.

Nós já melhoramos a manutenção e a legibilidade do código. Aproveitaremos para fazer o mesmo com o `Mensagem`.

```
this._mensagem = ProxyFactory.create(
    new Mensagem(), ['texto'], model =>
        this._mensagemView.update(model));
this._mensagemView = new MensagemView($('#mensagemView'));
```

No entanto, ainda teremos que chamar manualmente a View, usando o `update()` e chamando `this._mensagem`. Sempre que criamos o modelo autoatualizável, a View só será recarregada, quando o modelo for modificado. Mas vamos resolver isso.

Iremos para o método `adiciona()`, removeremos o `update()`.

```
adiciona(event) {

    event.preventDefault();
    this._listaNegociacoes.adiciona(this._criaNegociacao());
    this._mensagem.texto = 'Negociação adicionada com sucesso';
    this._limpaFormulario();
}
```

Faremos ajustes em `apaga()`:

```
apaga(){

    this._listaNegociacoes.esvazia();
    this._mensagem.texto = 'Negociações apagadas com sucesso';

}
```

Se recarregarmos a página no navegador, veremos que o formulário funciona corretamente, mas não atualizou a mensagem.