04

Lidando com tipos genéricos

Transcrição

Se olharmos atentamente nossas classes de `view`, vemos que o método `update` e `template` são bem parecidos. No entanto, o método `template()` é bem específico da classe filha, pois retorna strings especializadas. Que tal movermos o método `update()` para dentro de `View`?

```
class View {

    protected _elemento: Element;

    constructor(seletor: string) {

        this._elemento = document.querySelector(seletor);
    }

    update(model: string) {
        // erro de compilação
        this._elemento.innerHTML = this.template(model);
    }

}
```

Ops! Temos um erro de compilação, pois o método `update` depende do método `template(). Temos que mover o método` template` também.

```
class View {

    protected _elemento: Element;

    constructor(seletor: string) {

        this._elemento = document.querySelector(seletor);
    }

    update(model: string) {

        this._elemento.innerHTML = this.template(model);
    }

    template(model: string): string {

        return `<p class="alert alert-info">${model}</p>`;
    }

}
```

No caso do método template, vamos deixá-lo vazio, lançando uma exceção forçando as classes filhas a implementá-lo:

```
class View {

    protected _elemento: Element;

    constructor(seletor: string) {

        this._elemento = document.querySelector(seletor);
    }

    update(model: string) {

        this._elemento.innerHTML = this.template(model);
    }

    template(model: string): string {

        throw new Error('Você deve implementar o método template');
    }

}
```

Agora, nas classes `NegociacoesView` e `MensagemView` vamos remover o método `update()`. A classe `MensagemView` compila, mas `NegociacoesView` não.

```
Class 'NegociacoesView' incorrectly extends base class 'View'.
  Types of property 'template' are incompatible.
    Type '(model: Negociacoes) => string' is not assignable to type '(model: string) => string'
      Types of parameters 'model' and 'model' are incompatible.
        Type 'string' is not assignable to type 'Negociacoes'.
```

O problema é que em `View` temos fixos como parâmetro de `update` e `template` o tipo string, mas no caso de `NegociacoesView` o tipo precisa ser `Negociacoes`. E agora?

Podemos lançar mão de um recurso chamado **generics**. Vejamos a declaração da classe `View` com este recurso:

```
class View<T> {

    protected _elemento: Element;

    constructor(seletor: string) {

        this._elemento = document.querySelector(seletor);
    }

    update(model: T) {

        this._elemento.innerHTML = this.template(model);
    }

    template(model: T): string {

        throw new Error('Você deve implementar o método template');
    }
```

```
}
```

Vejam a sintaxe `class View<T>`. Nela, estamos indicando que a classe ao ser usada pode receber um tipo como indicação e esse tipo será usado em todos os lugares que `T` estiver presente.

Agora, quem for herdar de `View`, precisará indicar o tipo que desejamos utilizar na classe:

```
class NegociacoesView extends View<Negociacoes>{

    template(model: Negociacoes): string {

        return `
        <table class="table table-hover table-bordered">
            <thead>
                <tr>
                    <th>DATA</th>
                    <th>QUANTIDADE</th>
                    <th>VALOR</th>
                    <th>VOLUME</th>
                </tr>
            </thead>

            <tbody>

            ${model.paraArray().map(negociacao =>
                `
                    <tr>
                        <td>${negociacao.data.getDate()}/${negociacao.data.getMonth()+1}/${nego
                        <td>${negociacao.quantidade}</td>
                        <td>${negociacao.valor}</td>
                        <td>${negociacao.volume}</td>
                    </tr>
                `).join('')}
            </tbody>

            <tfoot>
            </tfoot>
        </table>
        `
    }
}
```

```
class MensagemView extends View<string> {

    template(model: string): string {

        return `<p class="alert alert-info">${model}</p>`;
    }
}
```

Nosso código continua compilando! Agora, não precisamos mais do modificador `protected`, pois agora acessarmos a propriedade através dos métodos da classe:

```
class View<T> {

    private _elemento: Element;

    constructor(seletor: string) {

        this._elemento = document.querySelector(seletor);
    }

    update(model: T) {

        this._elemento.innerHTML = this.template(model);
    }

    template(model: T): string {

        throw new Error('Você deve implementar o método template');
    }

}
```

Mas será que podemos melhorar nosso código?