01

Chega de múltiplos console.log

Transcrição

Começando deste ponto? Você pode fazer o download (https://s3.amazonaws.com/caelum-online-public/typescript/10-alurabank.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo.

Durante a construção da nossa aplicação utilizamos de maneira corriqueira a função `console.log` para escrutinarmos o valor das propriedade de nossos objetos. Por exemplo:

```
console.log(negociacao);
```

Mas dessa forma, estamos pegando carona no padrão de impressão do console.log. Se quisermos algo mais detalhado teremos que explicar isso em nosso código. Vejamos:

```
console.log(
    `Data: ${this.data}
    Quantidade: ${this.quantidade},
    Valor: ${this.valor},
    Volume: ${this.volume}`
)
```

Muito mais explicativo, mas imagina termos que repetir esse código em todo lugar que quisermos imprimir nosso objeto `Negociacao` ? Podemos isolar essa lógica na própria classe através do método `paraTexto()` .

```
export class Negociacao {

    constructor(readonly data: Date, readonly quantidade: number, readonly valor: number) {}

    get volume() {

        return this.quantidade * this.valor;
    }

    paraTexto(): void {
        console.log('-- paraTexto --');
        console.log(
            `Data: ${this.data}
            Quantidade: ${this.quantidade},
            Valor: ${this.valor},
            Volume: ${this.volume}`
        );
    }
}
```

Agora, toda vez que quisermos escrutinas os valores das propriedades do instância de `Negociacao` para fazermos:

```
negociacao.paraTexto();
```

Esse comportamento não é exclusivo de `Negociacao`, queremos a mesma coisa para `Negociacoes`. Contudo, não podemos simplesmente reaproveitar o código do método `paraTexto()` em `Negociacoes`, porque, apesar do método ter o mesmo nome, quantidade de parâmetros (nenhum) e retorno `void`, sua implementação é diferente.

Vamos adicionar o método `paraTexto()` em `Negociacoes`:

```
import { Negociacao } from './Negociacao';

export class Negociacoes {

    private _negociacoes: Negociacao[] = [];

    adiciona(negociacao: Negociacao): void {

        this._negociacoes.push(negociacao);
    }

    paraArray(): Negociacao[] {

        return ([] as Negociacao[]).concat(this._negociacoes);
    }

    paraTexto(): void {

        console.log('-- paraTexto --');
        console.log(JSON.stringify(this._negociacoes));
    }
}
```

Perfeito, agora podemos fazer:

```
negociacao.paraTexto();
this._negociacoes.paraTexto();
```

Contudo, muitas vezes queremos imprimir no console uma sequencia de objetos. Que tal criarmos uma função chamada `imprime` que receba como parâmetro um ou mais elementos que desejamos invocar o método `paraTexto`. Sabemos que através do REST Operator podemos passar uma quantidade indeterminada de parâmetros para a função.

Vamos criar nossa função em `app/ts/helpers/Utils.ts`. O nome `Utils` não é por acaso, todas nossas funções utilitárias ficarão nesse arquivo.

Agora, em `Utils.ts`, vamos criar e exportar a função `imprime`:

```
import { Negociacao } from '../models/index';

export function imprime(...negociacoes: Negociacao[]) {
    negociacoes.forEach(negociacao => negociacao.paraTexto());
}
```

E claro, nosso barrel:

```
// app/ts/helpers/index.ts

export * from './Utils';
```

Perfeito, agora vamos importar nossa função utilitária em `NegociacaoController` e utilizá-la para chamar, numa tacada só, o método `paraTexto()` de cada objeto que passarmos para ele:

```
import { NegociacoesView, MensagemView } from '../views/index';
import { Negociacao, Negociacoes } from '../models/index';
import { domInject, throttle } from '../helpers/decorators/index';
import { NegociacaoParcial, Imprimivel } from '../models/index';
import { NegociacaoService } from '../services/index';

// importou a função utilitária

import { imprime } from '../helpers/index';

// código anterior omitido

    adiciona() {

        // código anterior omitido

        this._negociacoes.adiciona(negociacao);
        this._negociacoesView.update(this._negociacoes);
        this._mensagemView.update('Negociação adicionada com sucesso!');

        // imprime no console a negociacao
        imprime(negociacao);
    }
```

Excelente! Agora, vamos passar um segundo parâmetro, desta vez, `this._negociacoes` :

```
// código anterior omitido

    adiciona() {

        // código anterior omitido

        this._negociacoes.adiciona(negociacao);
        this._negociacoesView.update(this._negociacoes);
        this._mensagemView.update('Negociação adicionada com sucesso!');

        // OPS! Erro de compilação!
        imprime(negociacao, this._negociacoes);
    }
```

Temos um problema. A função imprime aceita receber apenas objetos do tipo `Negociacao` , por isso estamos tendo o erro

```
Argument of type 'Negociacoes' is not assignable to parameter of type 'Negociacao'.
```

E agora? Precisamos fazer com que ele aceite qualquer objeto. Que tal usarmos o tipo `any[]` em vez de `Negociacao[]`? Isso significa que podemos passar qualquer objeto.

```
export function imprime(...objetos: any[]) {
    objetos.forEach(objeto => objeto.paraTexto());
}
```

Excelente, essa alteração silencia o compilador e nosso código funciona como esperado. No entanto, o que acontecerá se fizermos isso:

```
imprime(negociacao, this._negociacoes, data);
```

Estamos passando um objeto do tipo `Date`. Como usarmos o tipo `any[]`, ele será aceito. Mas será que nossa função `imprime` funcionará em tempo de execução? Um teste exibe o seguinte erro no console:

```
Uncaught TypeError: objeto.paraTexto is not a function
```

É, não vai funcionar, porque `Date` não possui o método `paraTexto`. Do jeito que esta, nada impede do programador passar qualquer objeto que não tenha o método `paraTexto`. O ideal seria que o compilador nos avisa-se da ausência desse método, para que possamos descobrir este erro em tempo de compilação e não runtime. A boa notícia é que há um recurso que podemos utilizar, assunto do próximo vídeo.