02

Decorator de método

Transcrição

Podemos isolar a lógica do nosso teste de performance em um único lugar e aplicá-lo nos métodos que temos interesse sem que tenhamos que modificar sua implementação. Para tal, precisamos ativar em nosso compilador TypeScript a configuração `experimentalDecorators`. Quando `true`, permite utilizar decorators, estrutura que atenderá nossa finalidade.

```
{
    "compilerOptions": {
        "target": "es6",
        "outDir": "app/js",
        "noEmitOnError": true,
        "noImplicitAny": true,
        "removeComments": true,
        "module": "system",
        "strictNullChecks": true,
        "experimentalDecorators": true
    },
    "include": [
        "app/ts/**/*"
    ]
}
```

Como criar um decorator? Vamos criar o arquivo `app/ts/helpers/decorators/logarTempoDeExecucao.ts` e nele exportamos uma função de mesmo nome:

```
export function logarTempoDeExecucao() {

}
```

Essa função indica o nome do nosso decorator, mas sua implementação deve estar na função retornada por `logarTempoDeExecucao`:

```
export function logarTempoDeExecucao() {

    return function(target: any, propertyKey: string, descriptor: PropertyDescriptor) {

    }
}
```

A função retornada não recebe três parâmetros por acaso. O primeiro `target` é aquele que possui uma referência para o elemento cujo método foi decorado por `logarTempoDeExecucao`. O segundo parâmetro é uma string que nos retorna o nome do método decorado. Por fim, o `descriptor` nos dará acesso ao método que desejamos modificar sua execução, através de `descriptor.value`.

A ideia é a seguinte. Vamos guardar uma referência para o método original antes de substituí-lo com nosso código. É importante que no final o `descriptor` seja retornado, com as modificações que faremos:

```
export function logarTempoDeExecucao() {

    return function(target: any, propertyKey: string, descriptor: PropertyDescriptor) {

        const metodoOriginal = descriptor.value;

        // aqui vamos substituir descriptor.value pela lógica do nosso decorator

        return descriptor;
    }
}
```

O valor de `descriptor.value` será `function(...args: any[])`. Isso se dá dessa forma, porque o método que estamos sobrescrevendo pode receber zero, um ou mais parâmetros de tipos que desconhecemos. Usamos `...` para indicar um REST PARAMETER, algo que não é exclusivo do TypeScript, mas do ES2015:

```
export function logarTempoDeExecucao() {

    return function(target: any, propertyKey: string, descriptor: PropertyDescriptor) {

        const metodoOriginal = descriptor.value;

        descriptor.value = function(...args: any[]) {

            const retorno = metodoOriginal.apply(this, args);
            return retorno;
        }

        return descriptor;
    }
}
```

Fazemos `metodoOriginal.apply(this, args)` para invocar o método original, capturar seu resultado, caso exista e retorná-lo. Ainda não há a nossa lógica do teste de performance. Por enquanto vamos deixar assim. Não deve ocorrer nenhum erro e o comportamento da nossa aplicação deve continuar o mesmo. Mas para que possamos utilizá-lo, vamos exportá-lo através de um barril e importá-lo em `View`.

Primeiro, criando o barril

```
// app/ts/helpers/decorators/index.ts
export * from './logarTempoDeExecucao';
```

Agora, importando e utilizando em `View`:

```
import { logarTempoDeExecucao } from '../helpers/decorators/index';

export abstract class View<T> {
```

```
    protected _elemento: JQuery;
    private _escapar: boolean;

    constructor(seletor: string, escapar: boolean = false) {

        this._elemento = $(seletor);
        this._escapar = escapar;
    }

    @logarTempoDeExecucao()
    update(model: T) {

        let template = this.template(model);
        if(this._escapar)
            template = template.replace(/<script>[\s\S]*?<\/script>/g, '');
        this._elemento.html(template);

    }

    abstract template(model: T): string;

}
```

Usamos decorator através de um `@`, seguido do nome do decorator, abrindo e fechando parênteses no final, justo, porque um decorator nada mais é do que uma função.

Recarregando nossa aplicação tudo continua funcionando. Agora precisamos escrever a lógica do teste de performance em nosso decorator.