06

Medindo o tempo de execução de métodos

Transcrição

Agora, só precisamos guardar o tempo antes da chamada do método original e logo depois da sua chamada, para no fim, realizarmos o cálculo do tempo gasto. Inclusive, vamos exibir no console os parâmetros recebidos pelo método, inclusive seu retorno:

```
export function logarTempoDeExecucao() {

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

            const metodoOriginal = descriptor.value;

            descriptor.value = function(...args: any[]) {
                console.log('-----------------------')
                console.log(`Parâmetros do método ${propertyKey}: ${JSON.stringify(args)}`);
                const t1 = performance.now();
                const resultado = metodoOriginal.apply(this, args);
                console.log(`Resultado do método: ${JSON.stringify(resultado)}` )
                const t2 = performance.now();
                console.log(`${propertyKey} demorou ${t2 - t1} ms`);
                console.log('-----------------------')
                return resultado;
            }
            return descriptor;
    }

}
```

Vamos analisar também o tempo de execução dos métodos `adiciona` e `paraArray`, porque os métodos recebem e retornam valores, respectivamente, e isso nos permitirá escrutinar seus valores:

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

export class Negociacoes {

    private _negociacoes: Negociacao[] = [];


    @logarTempoDeExecucao()
    adiciona(negociacao: Negociacao): void {

        this._negociacoes.push(negociacao);
    }


    @logarTempoDeExecucao()
    paraArray(): Negociacao[] {
```

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

Excelente! Tudo funciona, mas que tal se pudermos dar para o usuário a possibilidade de mostrar o tempo gasto com a chamada do método em segundos, em vez de milissegundos? Podemos passar um parâmetro para nosso decorator e a partir desse parâmetro ajustar como ele deve se comportar:

```
export function logarTempoDeExecucao(emSegundos: boolean = false) {

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

            const metodoOriginal = descriptor.value;

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

                let divisor = 1;
                let unidade = 'milisegundos'
                if(emSegundos) {
                    divisor = 1000;
                    unidade = 'segundos';
                }

                console.log('------------------------')
                console.log(`Parâmetros do método ${propertyKey}: ${JSON.stringify(args)}`);
                const t1 = performance.now();
                const resultado = metodoOriginal.apply(this, args);
                console.log(`Resultado do método: ${JSON.stringify(resultado)}` )
                const t2 = performance.now();
                console.log(`${propertyKey} demorou ${(t2 - t1)/divisor} ${unidade}`);
                console.log('------------------------')
                return resultado;
            }
            return descriptor;
    }
}
```

Se o parâmetro não for passado ou for passado `false`, será considerado milissegundos. Se o parâmetro for `true`, usaremos a segundos como unidade:

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

export class Negociacoes {

    private _negociacoes: Negociacao[] = [];


    @logarTempoDeExecucao(true)
    adiciona(negociacao: Negociacao): void {

        this._negociacoes.push(negociacao);
    }
```

```
    @logarTempoDeExecucao(true)
    paraArray(): Negociacao[] {

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

```
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(true)
    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;

}
```

Para não ficarmos poluindo nosso console, vamos deixar apenas o decorator no método `update` de `View`.