10

Ciclo de vida de um componente

Transcrição

Antes de terminarmos a aula, há mais uma melhoria que podemos aplicar. Atualmente, buscamos as imagens e acessamos a API no `constructor` da classe `AppComponent`. O que acontece quando o componente é construído? Inicialmente o Angular cria uma instância de `AppComponent`, e depois aplica o *decorator* `@Component` para tornar esta instância em um componente efetivamente.

A maneira como organizamos nosso código funciona, mas podemos padronizá-lo mantendo o `constructor` apenas para injeção de dependência, e qualquer lógica que queiramos executar será colocada em uma fase do ciclo de vida que todo componente Angular possui.

Todo componente Angular possui um ciclo de vida, e focaremos agora em ***ngOnInit*** ou, abreviando, ***OnInit***. Se passarmos o mouse sobre este método da maneira em que está em `app.component.ts`, nada será retornado, e por isso ele é `void`. Moveremos o código de `photoService` para `ngOnInit()`, mas sabemos que este código não funcionará, pois o `photoService` é acessível somente no `constructor`, sendo preciso acessá-lo como propriedade de classe. Para tal, usaremos `private`:

```
  constructor(private photoService: PhotoService) { }

  ngOnInit(): void {

      this.photoService
```

```
            .listFromUser('flavio')
            .subscribe(photos => this.photos = photos);
    }
```

COPIAR CÓDIGO

No Java, não é necessário incluir `this`, mas em JavaScript e TypeScript, usamos ele para acessar a propriedade de uma classe. Salvaremos, voltaremos ao navegador, e tudo continua funcionando bem. A fase *OnInit* ocorre depois da instanciação de `AppComponent`, e depois do componente receber as *inbound properties*.

O mais importante, agora, é que este método nos salvará em algumas situações, mas por enquanto queremos convencionar e usar o construtor apenas para injeção de dependência, e qualquer código de inicialização de configuração será feito no `ngOnInit`. Porém, se escrevermos o método errado, não temos erro de compilação. No entanto, quando retornarmos ao navegador, nada é exibido.

Isso ocorre porque ao tratá-lo como um componente, o Angular espera encontrar o método `ngOnInit()` exatamente desta forma, ignorando e deixando de chamá-lo caso ele não seja encontrado. Seria melhor se o TypeScript pudesse nos avisar caso tenhamos algum erro deste tipo.

Podemos voltar o código para a maneira como estava antes de colocarmos o método e, logo após `AppComponent`, incluir `implements OnInit` e pressionar "Enter". O programa então fará a importação da interface `OnInit` de `angular/core`, que permitirá a definição da forma de um objeto para que possamos tratá-la de maneira tipada, além do uso do *autocomplete*, entre outros.

No entanto, uma interface também nos obriga a usar determinados métodos, então, quando implementamos a interface `OnInit` e passamos a mouse por cima, o programa diz que está faltando `ngOnInit()` na nossa classe. Ao clicarmos em `AppComponent`, no ícone de lâmpada, e então em "Implement

interface 'OnInit'", o Angular inclui automaticamente o método `ngOnInit()`. O nosso código continua compilando, porém ao rodarmos a aplicação teremos um problema, já que o método lança uma exceção.

Assim, moveremos o código de `photoService` usando a tecla "Alt" junto com as setas para cima ou para baixo, para dentro do método. Em seguida, moveremos todo o método para após o `constructor`, por convenção, e tornaremos `photoService` privada. Incluiremos `this` antes de `photoService`, e então poderemos salvar o arquivo.

```
import { Component, OnInit } from '@angular/core';

import { PhotoService } from './photos/photo/photo.service';

@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit{


    photos: any[] = [];

    constructor(private photoService: PhotoService) { }

    ngOnInit(): void {

        this.photoService
            .listFromUser('flavio')
            .subscribe(photos => this.photos = photos);
    }
}
```

COPIAR CÓDIGO

Desta vez, se digitarmos algo errado, a classe não irá compilar, e nenhuma alteração do projeto será vista enquanto o `ngOnInit()` não for implementado. Salvaremos, voltaremos ao navegador, e tudo continuará funcionando conforme esperado. O Angular possui outros ciclos de vida, mas por enquanto vimos este para convencionar que o `constructor` será destinado à injeção de dependência, e qualquer inicialização que queiramos fazer posteriormente será no `ngOnInit()`.

Por fim, acessando a definição da classe `photoService` (clique com "Ctrl"), sabemos que ela depende de um `HttpClient`, e para que ele esteja disponível, o `app.module.ts` faz a importação de `HttpClientModule`. Mas se pararmos para pensar, quem depende disso é `app.module.ts` ou `photos.module.ts`? Porque se `photo.service.ts` pertence a `photos.module.ts`, a ideia é que o segundo forneça o `HttpClient` de que necessita.

Caso `HttpClient` não esteja em `app.module.ts`, a aplicação não irá funcionar. Então, garantimos que a dependência necessária para este módulo funcionar esteja em `photos.module.ts`, e por isso incluiremos a linha `imports: [ HttpClientModule ]` em `@NgModule`, e removeremos `HttpClientModule` e a linha referente ao seu *import* em `app.module.ts`.

Salvaremos, voltaremos à aplicação, e tudo continua funcionando como antes. A grande questão é que se pegarmos a pasta `photos.module.ts`, ela importará tudo que for necessário para que seus componentes funcionem e, agora, sim, ela depende de `HttpClientModule`.