05

HttpClient e injeção de dependência

Transcrição

Com o Angular CLI rodando de um lado, e do outro nossa Web API, poderemos consumir os dados retornados por ela e exibir, no lugar de três imagens fixas, aquelas baseadas nos dados trazidos. Como faremos esta integração?

Sabemos que o primeiro componente a ser carregado pela aplicação é `AppComponent` (em `app.component.ts`), em que temos os dados fixos das imagens, os quais substituiremos por um *array* vazio:

```
export class AppComponent {

    photos = [];

}
```

COPIAR CÓDIGO

Ao salvarmos o arquivo e voltarmos ao navegador, nada aparecerá na página, já que se formos a `app.component.html`, o `ngFor` pegará um *array* vazio e não iterará em dado algum. Para termos acesso à comunicação com a Web API, podemos optar pela requisição Ajax, jQuery, entre outros. No entanto, no Angular existe um serviço específico para este tipo de tarefa, integrando-se ao *framework*, e através do qual toda comunicação com o back end será feita: o `HttpClient`.

Criaremos uma propriedade denominada `http` no `AppComponent`, e ela guardará uma instância. Ao digitarmos `new HttpClient`, o Visual Studio nos

dará a opção de realizarmos *auto import*. Essa é uma vantagem que o programa nos fornece, pois sem ela teríamos que fazer o *import* primeiro e, se não temos o endereço decorado mentalmente, não teríamos como fazê-lo.

Entretanto, ao pressionarmos "Enter", o programa inclui um `HttpClient` que não é aquele que queremos utilizar, e sim um do pacote `selenium-webdriver/http`, que serve para testes. Então, deletaremos a linha gerada automaticamente, clicaremos em `HttpClient` e no ícone de lâmpada para solicitarmos outro tipo de *import*. A única opção que o Visual Studio oferece é justamente "Import 'HttpClient' from module "selenium-webdriver/http"".

Então, incluiremos manualmente a seguinte linha no começo do arquivo `app.component.ts`, e o salvaremos:

```
import { HttpClient } from '@angular/common/http';
```

COPIAR CÓDIGO

Isso faz com que o programa nos informe que, para criarmos uma instância de `HttpClient` precisamos passar um `HttpHandler` no construtor. Vamos tentar:

```
export class AppComponent {

    photos = [];
    http = new HttpClient(new HttpHandler());

}
```

COPIAR CÓDIGO

Porém o programa continua acusando um erro e, ao passarmos o mouse sobre `HttpHandler`, teremos a mensagem de que ela é uma classe abstrata — por lidarmos com TypeScript, neste caso não poderemos usar o operador `new` —. e

isso parece ficar cada vez mais complicado. Queremos poder trabalhar com o `HttpClient` e só temos dificuldades.

Não nos interessam os detalhes de criação de um `HttpClient`. É como se quiséssemos desparafusar uma placa da parede, e quiséssemos uma chave de fenda, sendo irrelevante sabermos como ela foi criada, quem foi o inventor, e demais detalhes.

Neste sentido, queremos que nossa aplicação faça isso por nós. No caso, queremos uma **injeção de dependências**. No Angular, precisamos incluir o `constructor`, o qual toda classe em ECMAScript e TypeScript possui, e em que usaremos o parâmetro de quem dependemos, neste caso, `http`.

O Angular não sabe identificá-lo, podendo este ser uma *string* ou um número, por exemplo, então é necessário tiparmos, **explicitar seu tipo**. Em linguagens como Java, temos algo como `HttpClient http = new HttpClient()`, ou seja, o tipo vem antes da variável `http`. No TypeScript acontece o contrário, isto é, primeiro vem a variável, dois pontos, e o tipo. E então usamos `console.log()` para saber se a injeção realmente está acontecendo.

```
photos = [];

constructor(http: HttpClient) {
    console.log(http);
}
```

COPIAR CÓDIGO

Deste modo, o `AppComponent` será criado pelo Angular, e então passará pelo `constructor()`, e solicitará o `HttpClient`. E o Angular, de alguma forma, pegará o `HttpClient` e o injetará no construtor para que possamos utilizá-lo.

Ao abrirmos o navegador, nada será exibido, pois nenhum dado foi coletado. Porém, teremos muitos erros no console. É indicado "No provider for

HttpClient!", ou seja, que não há provedor para ele. O que acontece é que o `AppComponent` não obtém resposta em relação à solicitação do `HttpClient`, por sua complexidade. Ou seja, caímos no mesmo problema: não conseguimos criar um `HttpClient`, e o Angular também não.

Para resolvermos isto, precisamos de um *provider* para que o Angular o injete. Não o temos ainda, mas a boa notícia é que existe um módulo do próprio Angular que, ao ser importado em nossa aplicação, automaticamente terá um *provider* configurado para uso.

O Angular estava querendo fazer o *import* de `selenium-webdriver`, e desconhece o `HttpClient` que queremos, porque não importamos o módulo `HttpClientModule` no nosso projeto. Sendo assim, o Visual Studio com TypeScript é esperto o suficiente para entender que os artefatos e classes deste módulo estão indisponíveis. Lidaremos, então, com dois problemas:

- O Visual Studio precisa saber fazer o *import* do `HttpClient` correto;
- a ausência do *provider*.

Iremos ao módulo principal da aplicação, `app.module.ts`, e incluiremos `HttpClientModule` em `imports`:

```
imports: [
    BrowserModule,
    PhotosModule,
    HttpClientModule
],
```

COPIAR CÓDIGO

Teremos que importá-lo manualmente, digitando `import { HttpClientModule } from '@angular/common/http';` no começo do código. Salvaremos e, desta vez, ao retornarmos para `app.component.ts`, clicarmos em `HttpClient` e no ícone com lâmpada, serão exibidas várias opções. Selecionaremos "Import

'HttpClient' from module "@angular/common/http"", incluiremos `console.log(http)` novamente, voltaremos ao navegador, e deixamos de ter quaisquer problemas.

Com isso, o `HttpClient` foi injetado, e poderá ser trabalhado para fazer as requisições Ajax para o back end.