06

Usando DTO

Transcrição

[00:00] Continuando nosso treinamento de API REST com Spring Boot. Na última aula montamos o endpoint para carregar todos os tópicos. Testamos, vimos que funcionou certinho. Só que aí, eu tinha comentado que poderíamos deixar esse código mais simples, dar uma simplificada em algumas coisas. Nesta aula vamos ver exatamente esses detalhes.

[00:22] A primeira coisa que vamos fazer é: em todos os métodos que tivermos, em todos os "controllers", precisamos colocar o `@ResponseBody`. Se não colocarmos, o Spring, por padrão, considera que vamos fazer uma navegação para uma página (JSP, Thymeleaf ou qualquer tecnologia de view). Mas, nesse caso, como é uma API REST, não temos navegação para a página. Queremos devolver de fato o que estivermos retornando no método (por isso, temos a anotação `@ResponseBody`).

[00:50] Mas fica chato repetir essa anotação toda vez. Corremos o risco de esquecer. Para evitar isso, existe outra anotação que colocamos em cima da classe, em vez de usar o `@Controller`, que é o `@RestController`. Essa anotação é justamente para dizer que o `@controller` é um `@Rest controller`. Por padrão, ele já assume que todo método já vai ter o `@ResponseBody`. Se usarmos o `@RestController`, não precisamos mais ficar colocando o `@ResponseBody` em cima dos métodos. Ele já assume que esse é o comportamento padrão de todos os métodos.

```
package br.com.alura.forum.controller;

import java.util.Arrays;

@RestController
public class TopicosController {

    @RequestMapping("/topicos")
    public List<TopicoDto> lista() {
        Topico topico = new Topico("Duvida", "Duvida com
Spring", new Curso("Spring", "Programção"));

        return Arrays.asList(topico, topico, topico);
    }

}
```

COPIAR CÓDIGO

[01:31] Essa foi a primeira melhoria. A segunda coisa que podemos melhorar é que, conforme vimos, toda vez que temos nosso projeto rodando, se mexermos em qualquer coisa no código, essa mudança não é feita automaticamente. O Spring não detecta essa mudança. Se mexermos em qualquer coisa e atualizarmos o navegador, não vai acontecer nada.

[01:57] Temos que lembrar de ficar reiniciando o servidor. Alterei o código, - criando um novo `Controller` , um novo método, mapeando uma nova URL - tenho que parar o servidor e iniciar novamente. Isso é até improdutivo, ter que ficar parando e subindo o servidor o tempo inteiro, é meio chato.

[02:12] Para evitar essa necessidade, o pessoal do Spring Boot criou um módulo chamado "DevTools". Se utilizarmos esse módulo, não vamos mais precisar ficar reiniciando o servidor. Alteramos o código, salvamos, e na hora que salvarmos o Spring já reinicia sozinho.

[02:32] Vamos utilizar esse módulo do Devtools para ter essa comodidade. Como é um módulo do Spring Boot, precisamos adicioná-lo no projeto. Fazemos isso via dependências do Maven. Precisamos abrir o `pom.xml-forum`, selecionar o comando "Ctrl + shift + R". Depois, no código, ir até a parte das dependências, `<dependencies>`. Vou copiar a dependência que diz respeito ao `starter-web`, colar embaixo e vamos renomear: ao invés de ser `starter-web`, vai ser `spring-boot-devtools`.

```
<dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
```

COPIAR CÓDIGO

[03:05] Outro detalhe. Precisamos passar qual vai ser o escopo dessa dependência, que vai ser `runtime`. Ele só vai rodar essa dependência durante a execução do projeto. Vou salvar, o Maven vai baixar, e só isso. Vou rodar o projeto. Abrindo o "Console" (localizado na barra vertical do lado direito da tela) se você olhar no log, ele até detecta "Devtools property defaults active!", isto é, ele habilitou o DevTools no projeto.

```
<dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
        </dependency>
```

COPIAR CÓDIGO

[03:58] No navegador, acessando a URL http://localhost:8080/topicos (http://localhost:8080/topicos), pressionaremos "F5" e constataremos que carregou nossa lista com os tópicos. Só para testar se o DevTools está

funcionando, de volta ao `TopicosController.java`, vou abrir o controller e mudar algumas coisas pequenas (Acentuar a palavra "Dúvida" e incluir a vogal "a" na palavra "Programação"), depois que alterei, vou salvar com o comando "Ctrl + S". Perceba que, no console, ele já reiniciou sozinho o servidor. Se eu atualizar a página do navegador, a mudança aparece.

```
package br.com.alura.forum.controller;

import java.util.Arrays;

@RestController
public class TopicosController {

    @RequestMapping("/topicos")
    public List<TopicoDto> lista() {
        Topico topico = new Topico("Dúvida", "Dúvida com
Spring", new Curso("Spring", "Programação"));

        return Arrays.asList(topico, topico, topico));
    }

}
```

COPIAR CÓDIGO

A partir de agora, toda vez que você mexer no código e salvar, o Spring já vai reiniciar sozinho o servidor para você. Você não precisa parar e subir o servidor novamente, é muito mais produtivo.

[04:47] A terceira coisa que vamos melhorar é o retorno do controller. Nós devolvemos uma lista de `<Topico>`. Tópico é uma classe do nosso domínio, da nossa aplicação. Na próxima aula vamos utilizar JPA, então essa classe vai acabá sendo uma entidade do JPA, porque vai ter uma tabela no banco de dados.

[05:05] Não é uma boa prática devolver entidades da JPA no seu controller, porque na entidade da JPA - na classe de domínio - você tem um monte de atributos (título, mensagem, data de criação e outros). E você tem atributos que são outras entidades, outras classes que têm outros atributos, sendo que um deles pode ser outra classe. O Jackson, por padrão, "serializa" todos os atributos que estiverem dentro da classe. E nem sempre, eu vou querer devolver no meu JSON todos os atributos da minha classe. Posso querer devolver só dois, três atributos ou ter uma flexibilidade. Por exemplo, em um endpoint devolver x atributos, e, em outro, y.

[05:46] Se você sempre devolve a classe de domínio - a entidade da JPA - ele sempre vai devolver todos os atributos, você não tem flexibilidade. Por isso não é uma boa prática.

[05:57] Ao invés de devolvermos uma lista de tópicos, `List<Topico>`, vamos criar outra classe que representa só os dados que quero devolver nesse endpoint. E essa classe é só uma classe de valor, que só tem aqueles atributos que quero devolver. Geralmente o pessoal utiliza o padrão "*DTO*", Data Transfer Object, que muita gente também chama de "*VO*", Value Object, para esse tipo de classe.

[06:22] Nesse treinamento, vamos utilizar o padrão DTO. Então, ao invés de devolver uma "lista de tópicos", `List<Topico>`, vou devolver uma "lista de tópico DTO", `List<TopicoDto>`. Ele vai alertar que não existe essa classe no nosso projeto, então, vamos ter que criar.

[06:38] Vou usar o atalho "Ctrl + 1", selecionar a opção "Create Class 'TopicoDto'". E aí, dentro do pacote controller, vou criar um pacote chamado `.dto`, só para guardar as classes (Package: br.com.alura.forum.controller.dto). No código, vou criar a classe `topicoDto`. Perceba que, no console, o Spring já detectou a mudança, já reiniciou sozinho.

[06:56] Dentro do código, vou declarar somente os atributos, usando as classes primitivas do Java. Só vai ter "int", "long", ou "string" e "data". Não vai ter nenhuma classe de domínio aqui. Vai ser uma classe só com atributos primitivos.

[07:17] Neste DTO, quais são os campos do tópico que quero devolver? Para esse exemplo, vou devolver apenas o "id" (`private Long id;`), o "título" (`private String titulo;`), a "mensagem" (`private String mensagem;`) e a "data de criação" (`private LocalDateTime dataCriacao;` e importar o `localDateTime`).

```
package br.com.alura.forum.controller.dto;

import java.time.LocalDateTime;

public class TopicoDto {

    private Long id;
        private String titulo;
        private String mensagem;
        private LocalDateTime dataCriacao;

        }
```

COPIAR CÓDIGO

[08:09] Além dos atributos, vou gerar os getters, então, aperto o botão direito, seleciono "source", e, em seguida, "Generate Getters and Setters". Nesse tipo de classe, na verdade, só vou gerar os getters, não preciso dos setters. Então, na próxima tela, vou marcar que eu só quero gerar os Getters em "Select Getters". Mas, se eu só tenho os getters, vou precisar saber de onde ele vai puxar os dados. Então, aqui, vamos utilizar aquele esquema do construtor.

[08:39] Nessa classe, vou criar um construtor, `TopicoDto` . Toda vez que eu der `new` no `topicoDto` , passo como parâmetro um objeto do tipo `topico` . Isto é,

`TopicoDto(Topico topico)` Dentro do `Topico`, já tenho as informações: o "id", o "título" e a "mensagem". Recebo o tópico e, a partir desse tópico, preencho os atributos.

[09:00] Eu vou preencher de onde o "id" vem, por exemplo e vai ficar `topico`, que foi passado como parâmetro construtor, mais `getId()`, ou seja, `this.id = topico.getId();`. A mesma coisa com os outros atributos.

```
public TopicoDto(Topico topico) {
    this.id = topico.getId();
        this.titulo = topico.getTitulo();
        this.mensagem = topico.getMensagem();
        this.dataCriacao = topico.getDataCriacao();

        }
```

COPIAR CÓDIGO

[09:34] É isso. Quando eu der `new` nessa classe, passo o tópico e do tópico acesso as informações. Não precisa de setters, apenas de getters.

[09:42] Agora, no meu Controller, ele está reclamando porque esse método tem que devolver uma lista de `<TopicoDto>`, e não mais de `Topico`. Um lado negativo, uma parte chata é que vou ter a classe `<Topico>`, mas tenho que devolver o "Dto". Então, vou ter que fazer essa conversão de `<Topico>` para `<TopicoDto>`.

[10:02] Para não deixar essa lógica solta, vou criar um método que a encapsula dentro da própria classe DTO. Vou pegar o `TopicoDto` e, dentro dele, vou criar um método chamado ´`converter`. Passo para ele a lista de `Topico` e ele devolve a lista de `TopicoDto`.

```
return TopicoDto.converter(Arrays.asList(topico, topico,
topico));
```

COPIAR CÓDIGO

[10:27] Ele está reclamando porque esse método não existe, então, vou selecionar "Ctrl + 1" e escolher "Create method `convert(Lis<Topico>)` in type `TopicoDto` " para ele criar o método. Agora, basta renomear a variável `asList` para `topicos` . Continuando, esse método recebe a lista de `topicos` e preciso devolver uma lista de `topicoDto` , isto é, preciso fazer a conversão.

```
}

public static List<TopicoDto> converter(List<Topico> topicos)
{
    return null;
        }

}
```

COPIAR CÓDIGO

[10:42] Aqui, posso usar o Java 8 (A API de Stream do Java 8) para não ter que fazer manualmente. O código ficará:

```
return topicos.stream().map(TopicoDto::new);
```

COPIAR CÓDIGO

[11:13] A função do mapeamento será `TopicoDto::new` , porque ele vai chamar o construtor que recebe o próprio tópico como parâmetro. No final, tenho que transformar isso em uma lista, então vou encadear a chamada para o método `collect()` , passando `collectors.toList()` para transformar numa lista.

```
    }

    public static List<TopicoDto> converter(List<Topico> topicos)
    {
        return
    topicos.stream().map(TopicoDto::new).collect(collectors.toList());

            }

    }
```

COPIAR CÓDIGO

[11:32] Essa é a sintaxe do Java 8. Sem ele, teríamos que pegar essa lista de tópicos, fazer um `for` para cada tópico, dar `new` no `topicoDto` , guardar em uma lista de `topicoDto` e devolver essa lista de `topicoDto` no final. Desse jeito, ele faz tudo isso em uma linha só usando API de strings do Java 8.

[11:54] Voltando para o tópico controller, salvei, está tudo compilando, já reiniciou sozinho por causa do Spring Boot DevTools. Vou no navegador, acesso a URL http://localhost:8080/topicos (http://localhost:8080/topicos), recarrego, e tudo certo. Agora ele devolve somente o "id", o "título", a "mensagem" e a "data de criação".

[12:19] Desse jeito, tenho uma flexibilidade, consigo controlar quais campos quero devolver, porque nem sempre eu quero devolver tudo que tem na minha classe de domínio. Um exemplo é nossa classe "usuário", que tem o campo "senha". Se eu devolvesse uma lista de usuário, eu iria devolver a senha do usuário para o browser. Isso seria uma falha de segurança.

[12:49] Com isso, demos uma simplificada. A partir de agora, nos próximos endpoint que formos construir, vamos deixar o código mais simples e seguir esse

padrão.