02

Começando com Spring Data JPA

Transcrição

[00:00] Na última aula, demos uma melhorada no código e seguimos algumas práticas, como o uso do DTO, para evitar ficar levando informações desnecessárias para o cliente. Na aula de hoje vamos trocar a seguinte parte:

```
Topico topico = new Topico("Dúvida", "Dúvida com Spring",
new Curso("Spring", "Programação"));
```

COPIAR CÓDIGO

Vamos fazer a lógica para acessar os dados de um banco de dados, ao invés de criar a lista em memória.

[00:25] Para fazer isso, no nosso projeto vamos utilizar JPA, especificação do Java para acesso a banco de dados. O Spring Boot tem um módulo para trabalhar com JPA, deixando as coisas mais fáceis.

[00:39] O primeiro passo é adicionar o Spring Boot JPA no nosso projeto. Como de costume, vamos abrir nosso arquivo `pom.xml-forum` . Vou pegar alguma das dependências, por exemplo, a do `starter-web` . Depois, vou duplicar o código. E aí só preciso trocar o final, de `starter-web` para `starter-data-jpa` .

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

COPIAR CÓDIGO

Depois de salvo, o Maven vai baixar as dependências do Spring Boot Data JPA no projeto.

[01:12] Por padrão, o Spring Boot usa o "Hibernate" como implementação da JPA, mas você pode configurar se quiser utilizar outra implementação, como "EclipseLink". No curso, vamos utilizar "Hibernate" que é o padrão de mercado, o mais comum.

[01:28] Além disso, outra coisa que precisamos adicionar no projeto é o banco de dados. Precisamos colocar as dependências do driver do banco de dados. Poderia ser uma SQL, um Postgre, ou qualquer outro banco da sua preferência. Mas no curso, para simplificar um pouco a infraestrutura - para você não ter que instalar nada no seu computador - vamos utilizar o banco de dados H2, que é um banco de dados em memória, mais rápido e mais fácil de trabalhar.

[01:58] Precisamos, então, colocar mais uma dependência, `<dependency>`. Vou duplicar a linha anterior `starter-data-jpa` e adicionar o H2. No caso, não vai ser do Spring Boot. No `<groupId>` vai ser `com.h2database`. E no `<artifactId>` vamos apagar `spring-boot-starter-data-jpa` e colocar apenas `h2`. Essa é a dependência do banco de dados H2. Vou salvar e o Maven vai baixar também as dependências do H2.

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
</dependency>
```

COPIAR CÓDIGO

[02:25] São só essas duas dependências (O Spring Boot Data JPA e o H2, que é o nosso banco de dados). Agora, o próximo passo é configurar as informações do banco de dados - qual o endereço do banco de dados, usuário e a senha.

Em um projeto com Spring Boot, quase todas as configurações ficam em um arquivo chamado `application.properties`. Se você olhar no diretório "src/main/resources", você vai ver que, quando criamos o projeto, já tem esse arquivo `aplication.properties`, que no caso está vazio porque não precisamos configurar nada até então.

[03:00] Nesse arquivo, temos que colocar algumas linhas para configurar o H2 e o JPA. Vou colar algumas propriedades. Você não precisa decorar nenhuma delas. No exercício vocês vão puxar todas essas configurações prontas.

```
# datasource
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.url=jdbc:h2:mem:alura-forum
spring.datasource.username=sa
spring.datasource.password=

# jpa
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update

# Nova propriedade a partir da versao 2.5 do Spring Boot:
spring.jpa.defer-datasource-initialization=true

# h2
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
```

COPIAR CÓDIGO

[03:23] No primeiro bloco, onde consta o comentário `datasource`, temos as configurações do `datasource`, do banco de dados para o Spring. A primeira coisa que precisamos dizer é qual o driver de acesso ao banco de dados (o driver do JDBC). No caso, passamos a classe do H2. Você colocaria a classe do seu driver - do seu banco de dados.

```
#datasource
spring.datasource.driverClassName=org.h2.Driver
```

COPIAR CÓDIGO

[03:43] A segunda propriedade é a "URL" de acesso ao banco de dados. No nosso caso vai ser `jdbc:h2:mem` . O database vai se chamar `alura-forum` . Isto é, ele vai criar um banco de dados em memória chamado `alura-forum` .

```
#datasource
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.url=jdbc:h2:mem:alura-forum
```

COPIAR CÓDIGO

[04:02] As duas próximas propriedades são do usuário e da senha do banco de dados. Geralmente com o H2, o pessoal coloca o usuário "sa" e a senha fica em branco.

```
#datasource
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.url=jdbc:h2:mem:alura-forum
spring.datasource.username=sa
spring.datasource.password=
```

COPIAR CÓDIGO

No próximo bloco estão configurações específicas da JPA. Estamos usando o `hibernate` e ele precisa saber qual é o dialeto do banco de dados. Por isso, precisamos passar uma classe do `hibernate` que representa o dialeto do banco de dados. No caso do H2, a classe é `org.hibernate.dialect.HSDialect`

```
# jpa
sprinf.jpa.database-
plataform=org.hibernate.dialect.H2Dialect
```

COPIAR CÓDIGO

[04:30] Continuando, eu quero que o hibernate crie automaticamente o banco de dados, e, sempre que tiver uma mudança, uma nova tabela, uma nova coluna, puxe das propriedades das nossas entidades e atualize o banco de dados automaticamente. Então, colocamos a propriedade `jpa.hibernate.ddl-auto=update` .

```
# jpa
sprinf.jpa.database-
plataform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update
```

COPIAR CÓDIGO

[04:52] Embaixo, no bloco seguinte, vem algo interessante. Como vou fazer para acessar o H2 se não tenho nada instalado na máquina? O Spring Boot tem uma comodidade - se você estiver utilizando o H2 - que é a ideia de ter um console do H2. Nada mais é do que uma interface gráfica para o H2 que acessamos direto pelo navegador. Precisamos habilitar essa interface gráfica, colocando a propriedade `spring.h2.console.enable=true` .

```
# h2
spring.h2.console.enable=true
```

COPIAR CÓDIGO

[05:24] A próxima linha é para dizer qual é o `path` , o endereço para acessar a interface de gerenciamento do H2. Coloquei no exemplo `h2-console` . Conseguiremos acessar o banco de dados H2 direto no navegador, acessando a URL http://localhost:8080/h2-console (http://localhost:8080/h2-console). Depois vamos ver como vai funcionar isso.

```
# h2
spring.h2.console.enable=true
spring.h2.console.path=/h2-console
```

COPIAR CÓDIGO

[05:47] Essas são as propriedades para configurar o Datasource, a JPA, e, no nosso caso, o console do H2. Feito isso, já tenho a JPA baixada no projeto e configurada. O próximo passo é: como estamos utilizando a JPA, precisamos transformar as classes de domínio em entidades da JPA.

[06:12] Eu vou pegar, por exemplo, a classe `Usuario`. Em cima da classe usuário, adicionaremos uma entidade, considerando que toda entidade tem que ter a anotação `@Entity` que vem do pacote `javax.persistence.Entity`. Além disso, em cima do atributo que representa a chave primária, tem que ter duas anotações, o `@Id` e o `@GeneratedValue` (no caso, a chave primária vai ser gerada automaticamente pelo banco de dados).

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

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;


@Entity
public class Usuario {

    @Id @GeneratedValue
    private Long id;
    private String nome;
    private String email;
    private String senha;
//...
}
```

COPIAR CÓDIGO

[06:39] No `@GeneratedValue`, tenho que colocar qual é a estratégia, `strategy`. Geralmente colocamos a `IDENTITY`, se for "auto-increment", ou `SEQUENCE`, se seu banco de dados utilizar.

```
@Entity
public class Usuario {

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private String email;
    private String senha;
//...
}
```

COPIAR CÓDIGO

Vou importar o `@Id` que vem do `javax.persistence.Id` e, em seguida, copiar a linha `@Id @GeneratedValue(strategy = GenerationType.IDENTITY)`, porque precisaremos dela nas outras entidades. Agora, vou abrir a entidade `Curso`, `Curso.java`, que precisa ter o `@Entity` em cima da classe, vou importar do `javax.persistence.Entity` e colar a linha que copiei.

```
@Entity
public class Curso {

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private String categoria;

    public Curso(String nome, String categoria) {
        this.nome = nome;
        this.categoria = categoria;

    }
```

```
}

//...
```

COPIAR CÓDIGO

A próxima classe vai ser `Topico`. Então, vamos repetir `@Entity` em cima da classe e que vem lá do pacote `javax.persistence.Entity`. Depois, `@Id @GeneratedValue(strategy = GenerationType.IDENTITY)`. Só que, na classe `Topico`, por ser a principal, tem algumas coisas a mais que precisamos adicionar.

Por exemplo, o atributo `status` é um enum, e quero que o hibernate grave no banco de dados o nome da constante do enum, ao invés da ordem de declaração. Então, preciso colocar a anotação `@Enumerated()`, que, por padrão, grava a ordem, mas, como eu não quero guardar a ordem, vou passar o valor `STRING`.

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

import java.time.LocalDateTime;

@Entity
public class Topico {

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String título;
    private String mensagem;
    private LocalDateTime dataCriacao = localDateTime.now();

    @Enumerated(EnumType.STRING)
    private StatusTopico status =
StatusTopico.NAO_RESPONDIDO;
    private Usuario autor;
    private Cruso curso;
```

```
    private List<Resposta> respostas = new ArrayList<>();
    //...
}
```

COPIAR CÓDIGO

[07:52] Além disso, nossa classe `Topico` tem alguns relacionamentos com outras entidades. Preciso dizer a cardinalidade, de tópico para o usuário, para o autor, que vai ser `@ManyToOne`, muitos para um. Para o `Curso`, a mesma coisa (um curso pode ter vários tópicos, mas o tópico pertence a um único curso). E a lista de resposta, `List<Resposta>`. Como é uma lista, vai ser `@OneToMany`, um tópico pode ter várias respostas.

[08:20] No caso do `@ManyToOne`, tem que passar uma propriedade, que é o `mappedBy`, para ele não achar que é um novo mapeamento, porque na classe, e nela, a resposta, estará mapeado o relacionamento com o tópico. Então, `mappedBy = "topico"` que é o nome do atributo lá na classe resposta.

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

import java.time.LocalDateTime;

@Entity
public class Topico {

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String título;
    private String mensagem;
    private LocalDateTime dataCriacao = localDateTime.now();

    @Enumerated(EnumType.STRING)
    private StatusTopico status =
StatusTopico.NAO_RESPONDIDO;

    @ManyToOne
```

```
    private Usuario autor;

    @ManyToOne
    private Curso curso;

    @OneToMany(mappedBy = "topico")
    private List<Resposta> respostas = new ArrayList<>();
    //...
}
```

COPIAR CÓDIGO

[08:40] Vou só importar essas anotações e, então, a entidade `Resposta`, `Resposta.java`, é a última que falta. Vamos repetir `@Entity` em cima da classe e vem do pacote `javax.persistence.Entity`. Importar da JPA o `@Id` `@GeneratedValue(strategy = GenerationType.IDENTITY)`. Essa classe também tem os relacionamentos, então `@ManyToOne` e tem o `usuario` também que criou a resposta e é uma entidade, então, `@ManyToOne`.

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

import java.time.LocalDateTime;

@Entity
public class Resposta {

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String mensagem;

    @ManyToOne
    private Topico topico;

    private LocalDateTime dataCriacao = localDateTime.now();

    @ManyToOne
    private Usuario autor;
```

```
    private Boolean solucao = false;
    //...
}
```

COPIAR CÓDIGO

Para tudo isso da JPA temos um treinamento. Se você não conhece JPA, recomendo fazer o treinamento pelo menos para ter uma noção básica do que significam essas coisas.

[09:16] Já coloquei todas as anotações de mapeamento. O Spring Devtools ficou reiniciando sozinho, só para garantir eu vou parar. Vou rodar de novo nosso projeto e conferir se não tem nenhum erro.Às vezes tem um erro no mapeamento, alguma entidade, algum atributo ficou com algum problema. Na hora de rodar o projeto com Spring Boot ele já joga vários problemas com "exception" se tiver algo errado.

[09:45] Como é um banco de dados em memória, toda vez que eu rodar o projeto ele vai criar o banco de dados de novo e vou perder os dados. Eu não tenho nenhum dado cadastrado no banco de dados.

Só para termos algumas informações já populadas no nosso banco de dados, uma coisa que podemos fazer no projeto é: lá no diretório "src/main/resources", se você tiver um arquivo chamado `data.sql` (no meu projeto eu até já copiei esse arquivo), o Spring automaticamente pega os comandos SQL que estiverem aqui dentro e executa no banco de dados.

[10:21] No `data.sql` do nosso projeto, vamos ter alguns "inserts", só para popular nosso banco de dados. Então, estou inserindo no banco um usuário, dois cursos e três tópicos. Toda vez que eu rodar o projeto, ele vai ler o `data.sql` e rodar no nosso banco de dados H2 para popular o banco e termos sempre registros preenchidos.

[10:46] Vou abrir o navegador. O nosso controller ainda não está acessando o banco de dados, ele está devolvendo a lista estática em memória. Lembre-se que no `application.properties` nós colocamos o console do H2?

```
# h2
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
```

COPIAR CÓDIGO

Eu consigo acessar o H2 pelo navegador entrando no endereço "/h2-console". Então, acessarei no navegador a URL http://localhost:8080/h2-console (http://localhost:8080/h2-console), pressionar "Enter" e verei a interface do H2. Se precisarmos gerenciar, acessar o banco de dados, podemos fazer por essa interface do console do H2.

[11:22] Ele já vem com alguns valores preenchidos, só precisamos preencher a "JDBC URL" com o endereço "jdbc:h2:mem:alura-forum", que é o mesmo configurado no `application.properties` . Seguindo, vou selecionar "Connect" e ele vai acessar o H2.

No H2, o `hibernate` gerou as tabelas. Vamos ver se ele populou o banco de dados com o arquivo `data.sql` . Para isso, vou selecionar a tabela `TOPICO` e ele já joga o comando "SELECT FROM TOPICO", agora vou pressionar "Run" e ele trará certinho os três registros.

Vou limpar pressionando "Clear" e depois selecionar a tabela "CURSO", depois selecionar "Run" e estarão lá os dois registros de cursos. Então, funcionou. Ele configurou o banco de dados H2, o `hibernate` gerou as tabelas e leu aquele arquivo `data.sql` populando o nosso banco de dados.

[12:16] No vídeo de hoje, essa era a ideia: baixar a JPA, configurar, mapear as entidades, configurar o H2, o console do H2 para acessar no navegador, e está tudo funcionando certinho. No próximo vídeo vamos começar a acessar o

banco de dados. Ao invés de ter um tópico criado em memória, vamos acessar o banco de dados de verdade.