04

Salvando os Autores

Transcrição

Já conseguimos enviar do bean para a tela os autores, mas ainda não conseguimos fazer o inverso. Precisamos agora ligar os autores da tela e relaciona-los com o nosso bean. Para fazer isso precisamos ter uma forma de pegar os autores. Lembra do nosso formulário? Temos o atributo `itemValue` da `h:selectManyListbox` que nos devolve os *id's* dos Autores, por isso criaremos então uma lista para armazenar esses *id's* que serão enviados do formulário.

```
public class AdminLivrosBean{
    private List<Integer> autoresId = new ArrayList<>(); // fazemos new para evitar NullPointerI

    // demais atributos e métodos abaixo
}
```

Vamos alterar o nosso formulário para vincular ao nosso atributo `autoresId` .

```
<h:selectManyListbox value="#{adminLivrosBean.autoresId}">
    <!-- selectItens aqui no meio, não altere -->
</h:selectManyListbox>
```

E uma vez que temos essa ligação, podemos alterar o método salvar do `AdminLivrosBean` para *setar* os autores no livro.

```
public void salvar(){
        for(Integer autorId : autoresId){
         livro.getAutores().add(new Autor(autorId));
        }

        dao.salvar();
        System.out.println("Livro cadastrado com sucesso " + livro);
}
```

Perceba que nesse momento, o código não está compilando, pois não temos os autores dentro da classe `Livro` . Vamos resolver isso agora, criando um relacionamento entre as entidades `Livro` e `Autor` , assim, vamos abrir a classe `Livro` e adicionar o atributo `Autor` .

```
public class Livro{

    // demais atributos acima

    private List<Autor> autores;

}
```

Para o JPA somente este atributo não é suficiente para relacionar as entidades Livro e Autor, precisamos anotar o atributo, mas qual anotação usar? Vamos pensar, quantos autores podemos ter em um livro? Quantos livros um autor

pode escrever? A resposta é, um autor pode escrever vários livros e um livro pode ter vários autores, vamos usar a anotação @ManyToMany.

```
public class Livro{

    // demais atributos acima

    @ManyToMany
    private List<Autor> autores = new ArrayList<>();

}
```

O `@ManyToMany` está reclamando que o Autor não é uma entidade, precisamos anotar a classe `Autor` como `Entity`. Um outro ponto importante aqui é que o JPA precisa de um construtor vazio para instanciar a classe, vamos cria-lo também e já remover o atributo `nome` do segundo construtor, pois no `AdminLivrosBean` estamos passando apenas o `ID`. Não esqueça de sobrescrever o toString().

```
@Entity
public class Autor{

    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;

    private String nome;

    public Autor() {}

    public Autor(Integer id) {
        this.id = id;
    }

    // getters e setters abaixo e toString
}
```

Recapitulando o que vai acontecer aqui com toda essa configuração *JPA* realizada. O ManyToMany, cria uma tabela auxiliar para armazenar os id's de cada tabela, assim podemos ter vários autores para um livro, e um autor pode escrever vários livros.

Mas até o momento, nossa classe `AdminLivrosBean` só estava criando 2 autores na mão. Vamos alterar isso, para que possamos pegar os autores direto do banco. Para isso vamos criar primeiro a classe `AutorDao` no pacote `br.com.casadocodigo.loja.daos` com o método que nos retorna a lista de autores.

```
public class AutorDAO {

    @PersistenceContext
    private EntityManager manager;

    public List<Autor> listar(){
        manager.createQuery("select a from Autor a", Autor.class)
            .getResultList();
    }
```

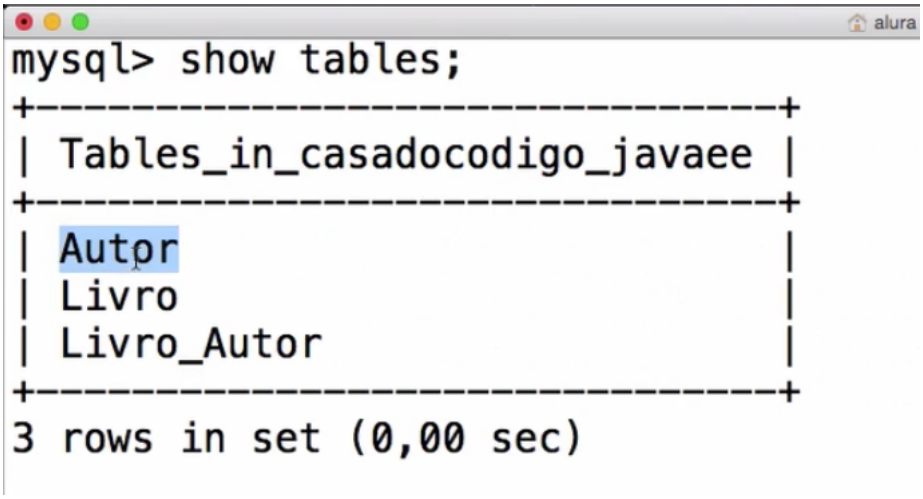
```
}
```

Temos nossa lista de autores, vamos voltar para o bean `AdminLivrosBean` e modificar o método `getAutores()`. Para isso vamos precisar injetar o `AutorDao`, usando mais uma vez o **CDI** conforme abaixo:

```
public class AdminLivrosBean {

    // demais atributos acima
    @Inject
    private AutorDao autorDao;

    public List<Autor> getAutores() {
        return autorDao.listar();
    }

    // demais métodos abaixo
}
```

Agora vamos testar. Não esqueça que não temos nenhum autor cadastrado, desta forma não aparecerá nenhum autor no form. Mas precisamos subir a aplicação para que o *Hibernate* crie a nossa tabela de Autor.

Depois de subir o servidor vamos verificar o nosso banco, veja que aparece não só a tabela de `Autor`, mas também a `Livro_Autor` que foi gerado devido a nossa anotação `@ManyToMany`, onde serão armazenado os id's que vinculam a tabela `Livro` e `Autor`. Ao realizar um `show tables;` pelo MySQL temos o resultado:



Vamos inserir alguns autores para continuar com nosso teste. Conecte-se ao seu MySQL com sua ferramenta preferida, e execute o seguinte comando:

```
insert into Autor (nome) values ('Paulo Silveira'), ('Sérgio Lopes'), ('Guilherme Silveira'), (
```

Com a tabela autor preenchida, ao selecionarmos um Autor e tentar cadastrar um Livro recebemos um erro: `ClassCastException`. Isso aconteceu por que o JSF está passando os `id's` de autores como `String` e no nosso bean `AdminLivrosBean` esperamos uma lista de `Integer`. Para resolver esse problema, usaremos um `Converter` do JSF que já está pronto para nos ajudar a resolver esse problema. Um converter, serve para informar ao JSF que o valor passado para o bean deve ser convertido para `Integer`, o nome do converter que usaremos é `javax.faces.Integer`.

```
<!-- Adicionamos o converter aqui - CÓDIGO NOVO -->
<h:selectManyListbox value="#{adminLivrosBean.autoresId}" converter="javax.faces.Integer">
    <f:selectItems value="#{adminLivrosBean.autores}"
         var="autor"
         itemValue="#{autor.id}"
         itemLabel="#{autor.nome}" />
</h:selectManyListbox>
```

Vamos subir nossa aplicação após essas alterações e tentar cadastrar um novo livro. Após clicar em salvar verifique pelo console do Eclipse, o log do Hibernate para conferir se aparecem os *inserts* de Livro e Livro_Autor.

Vá até seu cliente de MySQL e faça um select na tabela Livro_Autor para ver o relacionamento entre as duas tabelas: `select * from Autor_Livro; .`

Uma boa prática, é já realizar uma pesquisa no google sobre os converters existentes para o JSF. Temos diversos conversores e vamos falar mais sobre eles logo logo.

Você deve ter observado que após salvar, os dados preenchidos continuam aparecendo no formulário. Vamos limpa-los, e para isso no método salvar vamos limpar os atributos de Livro e de Autor.

```
@Transactional
public void salvar(){
    // código já existente aqui, continue ao final do método

    this.livro = new Livro();
    this.autoresId = new ArrayList<>();
}
```

Realize o *Full Publish* novamente, e tente salvar um novo Livro. Perceba que agora sim, após clicar em Cadastrar, nosso formulário volta a ficar vazio.