10

# Listando os Livros e Autores

### Transcrição

Nosso próximo passo é conseguir ver os livros e autores cadastrados, vamos criar um novo arquivo dentro de `src/main/webapp/livros`. Basta copiar o `form.xhtml` e colar, modificando o nome do arquivo para `lista.xhtml`. Dentro do arquivo vamos remover todo o form dele e aproveitar apenas a estrutura, por isso que copiamos e colamos ele, pois o Eclipse não possui um template default com essa estrutura base. Assim, temos o arquivo `lista.xhtml` apenas com as tags abaixo:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:f="http://xmlns.jcp.org/jsf/core">

    <!-- Código vai aqui -->

</html>
```

Dentro do nosso novo arquivo, vamos adicionar um `h:datatable` que é a tabela de dados onde iremos exibir nossos livros. Nossa tabela de livros tem várias colunas, vamos adicionar a princípio somente a coluna título com a tag h:column. Perceba no código abaixo, que o resultado do título precisa de um `livro`, esse livro é o `var` que colocamos no `h:dataTable` e o livro, vem da lista de livros do atributo `value`. Assim, temos o seguinte código:

```
<h:dataTable var="livro" value="#{adminListaLivrosBean.livros}">
        <h:column>
         #{livro.titulo}
        </h:column>
</h:dataTable>
```

Observe que assim como fizemos no `h:selectManyListbox`, onde tivemos que informar de onde vem os dados, temos que fazer o mesmo para o nosso `h:datatable`. Vamos precisar criar um novo manage bean e dentro dele criar um método que nos devolva os livros do banco de dados. Essa prática de usar um manage bean por tela é bem comum no desenvolvimento de projetos JSF, cada *xhtml* tem um *ManageBean* por trás. Perceba que no atributo `value`, já usamos o nome `adminListaLivrosBean`, vamos então criar a classe `AdminListaLivrosBean` no pacote `br.com.casadocodigo.loja.beans` que será o nosso Bean.

Como ele é um bean que pertence ao JSF, deveríamos anotá-lo com o `@Named` e para que os dados permaneçam na tela durante o *request*, deveríamos anota-lo com `@RequestScoped`, mas o pessoal do CDI decidiu unificar estas duas anotações em uma só, criando assim a annotation `@Model`. Desta forma nossa bean ficará assim:

```
@Model
public class AdminListaLivrosBean {

    private List<Livro> livros = new ArrayList<>();
```

```
    public List<Livro> getLivros() {
        return livros;
    }

}
```

Se você pressionar o **F3** pelo Eclipse, em cima da annotation `@Model`, você perceberá que ela é um *Estereótipo* (ou metadado) para `@RequestScoped` e `@Named`.

Precisamos preencher o nosso atributo livros. Que classe consegue manipular os dados do livro no banco? A classe `LivroDao`. Então vamos injeta-la no nosso bean e chamar o método listar dentro do método `getLivros()`.

```
@Model
public class AdminListaLivrosBean {

        @Inject
        private LivroDao dao;

        private List<Livro> livros = new ArrayList<>();

        public List<Livro> getLivros() {
         this.livros = dao.listar();

         return livros;
        }
}
```

Opa, o metodo `listar` ainda não existe na nossa classe `LivroDao`, precisamos cria-lo. Abra seu arquivo `LivroDao` e adicione o método a seguir:

```
    public List<Livro> listar() {
        String jpql = "select l from Livro l";

        return manager.createQuery(jpql, Livro.class).getResultList();
    }
```

Certo, temos agora o nosso método e o *JPQL* informado nos informa que serão retornados todos os livros, mas falta algo, precisamos também dos autores do livro, temos que modificar o nosso JPQL.

Sempre que for necessário relacionar uma tabela com a outra temos que usar o Join. E vamos precisar que este *Join*, faça a junção entre Livro e Autores, ou seja, para cada livro ele deve carregar os autores, vamos usar então o `join fetch`. Lembrando que estamos tratando de **Objetos**, assim usamos `Livro` referenciando a classe e `autores` como sendo o atributo `List<Autor>` dentro da classe `Livro` que está sendo referenciada pelo JPA.

Existe outro problema que precisamos resolver. Como podemos ter mais de um autor para cada livro temos que fazer um `distinct` para distinguir o livro independente da quantidade de autores, ou seja, só trará um único resultado de livro independente da quantidade de autores. Desta forma nosso código ficará assim:

```
    public List<Livro> listar() {
        String jpql = "select distinct(l) from Livro l"
            + " join fetch l.autores";
```

```
        return manager.createQuery(jpql, Livro.class).getResultList();
    }
```

Voltando para a classe `AdminListaLivrosBean`, agora não temos mais erros. Vamos rodar a nossa aplicação e testar a lista.

Tudo agora deve estar funcionando, mas estamos exibindo apenas a coluna título, vamos colocar os demais valores. Nossa `dataTable` completa ficará assim:

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:f="http://xmlns.jcp.org/jsf/core"
    xmlns:ui="http://xmlns.jcp.org/jsf/facelets"> <!-- Novo namespace -->

    <h:dataTable var="livro" value="#{adminListaLivrosBean.livros}">
        <h:column>
            <f:facet name="header">Título</f:facet>
            #{livro.titulo}
        </h:column>
        <h:column>
            <f:facet name="header">Descrição</f:facet>
            #{livro.descricao}
        </h:column>
        <h:column>
            <f:facet name="header">Páginas</f:facet>
            #{livro.numeroPaginas}
        </h:column>
        <h:column>
            <f:facet name="header">Preço</f:facet>
            #{livro.preco}
        </h:column>
        <h:column>
            <f:facet name="header">Autores</f:facet>
            <ui:repeat value="#{livro.autores}" var="autor">
                #{autor.nome},
            </ui:repeat>
        </h:column>
    </h:dataTable>

</html>
```

Alguns detalhes importantes de notar que fizemos na `h:dataTable` acima:

- Aproveitamos e adicionamos a tag `f:facet` com o atributo `name="header"`, que cria um cabeçalho na nossa coluna.
- Observe a coluna de autores, usamos o componente `ui:repeat` que é do *facelets*. Por isso adicionamos um novo *namespace* na tag `html` com o prefixo `ui`.
- Feito isso podemos usar o componente `ui:repeat`, que irá iterar e imprimir todos os autores vinculados a cada livro. O `ui:repeat` é basicamente um `for`, onde o `value` é a lista e o `var` o variável que ficará disponível com o autor dentro do `for`.

Realize mais um *Full Publish* para verificar sua lista de livros com os autores relacionados a ele.

Agora que já temos a listagem de livros, seria bem interessante que após o cadastro de nosso formulário, o usário fosse direcionado para a listagem, onde ele pode ver o livro que ele acabou de cadastrar.

Vamos voltar para a classe `AdminLivrosBean`, precisaremos modificar o método salvar que atualmente salva o nosso livro no banco e em seguida limpa o formulário, para que redirecione o usuário para nossa lista de livros.

```
@Transactional
public String salvar() { // Mudamos o tipo de retorno
    for (Integer autorId : autoresId) {
        livro.getAutores().add(new Autor(autorId));
    }

    dao.salvar(livro);
    System.out.println("Livro Cadastrado: " + livro);

    return "/livros/lista"; // E retornamos a página que o usuário irá sem o .xhtml
}
```

Observe que mudamos o retorno que antes era `void` para `String`, assim conseguimos dizer ao JSF que envie o usuário de volta para a lista de livros. E no retorno, dizemos o caminho a partir da raiz `/src/main/webapp/`, onde ele irá procurar o arquivo, assim `return "/livros/lista";` está procurando por `/src/main/webapp/livros/lista.xhtml`. Também não precisamos adicionar a extensão do arquivo.

Vamos testar nossa aplicação e percebemos que conseguimos salvar e após salvar somos direcionados para a tela de listar livros. Agora experimente apertar F5 após o cadastro de algum produto.

Veja que o navegador nos questiona se queremos resubmeter o formulário. Isso quer dizer que se confirmarmos, ele vai enviar os dados do produto novamente e teremos o produto recadastrado. Duplicando os produtos, o que não é bom!

O que aconteceu foi que o navegador ainda está guardando os dados do post do formulário. Apesar de ser um problema real, não podemos culpar o navegador, pois este é o funcionamento normal no caso de *methods posts* de formulário. Modificaremos então este comportamento em nossa aplicação.

Resolveremos isto através de recursos do protocolo HTTP chamado de *redirect*. O *redirect* passa um *http status* para o navegador carregar uma outra página e esquecer dos dados da requisição anterior. O *http status* que o navegador recebe é um 302.

Para isso devemos mudar o retorno do método salvar, que além do caminho que já retornava, retornará um parametro informando ao JSF para enviar um *redirect* ao navegado.

```
        return "/livros/lista?faces-redirect=true";
```

Vamos testar a aplicação e cadastrar um novo livro. Observe que a URL mudou de `form.xhtml` para `lista.xhtml`, confirmando que houve o redirect.