01

Continuando mudanças

Transcrição

Começando daqui? Você pode fazer o DOWNLOAD (https://s3.amazonaws.com/caelum-online-public/jsf-primefaces/stages/capitulo-5.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo.

Como foi falado no capítulo anterior, neste capítulo iremos finalizar a página `livro.xhtml`, alterando a tabela que exibe os livros cadastrados. Nesse momento, o `dataTable` está assim:

```
<h:dataTable value="#{livroBean.livros}" var="livro" id="tabelaLivros">
    <h:column>
        <f:facet name="header">Título</f:facet>
        <h:outputText value="#{livro.titulo}" />
    </h:column>
    <h:column>
        <f:facet name="header">ISBN</f:facet>
        <h:outputText value="#{livro.isbn}" />
    </h:column>
    <h:column>
        <f:facet name="header">Preço</f:facet>
        <h:outputText value="#{livro.preco}">
            <f:convertNumber type="currency" pattern="R$ #0.00"
                currencySymbol="R$" locale="pt_BR" />
        </h:outputText>
    </h:column>
    <h:column>
        <f:facet name="header">Data</f:facet>
        <h:outputText value="#{livro.dataLancamento.time}">
            <f:convertDateTime pattern="dd/MM/yyyy"
                timeZone="America/Sao_Paulo" />
        </h:outputText>
    </h:column>

    <h:column>
        <f:facet name="header">Alterar</f:facet>
        <h:commandLink value="altera" action="#{livroBean.carregar(livro)}"/>
    </h:column>

    <h:column>
        <f:facet name="header">Remover</f:facet>
        <h:commandLink value="remove" action="#{livroBean.remover(livro)}"/>
    </h:column>
</h:dataTable>
```

Podemos começar com o que já vimos no capítulo anterior, trocando o prefixo dos componentes `dataTable` e `column`, para utilizar o Primefaces. Podemos também adicionar um `f:facet` para definir o cabeçalho da tabela, e remover os `f:facet` s das colunas, já que o `p:column` já tem o atributo `headerText` (você pode fazer de qualquer um dos modos, ou utiliza o `headerText`, ou `f:facet`, ambos funcionam):

```
<p:dataTable value="#{livroBean.livros}" var="livro" id="tabelaLivros">
    <f:facet name="header">Livros</f:facet>
    <p:column headerText="Título">
        <h:outputText value="#{livro.titulo}" />
    </p:column>
    <p:column headerText="ISBN">
        <h:outputText value="#{livro.isbn}" />
    </p:column>
    <p:column headerText="Preço">
        <h:outputText value="#{livro.preco}">
            <f:convertNumber type="currency" pattern="R$ #0.00"
                currencySymbol="R$" locale="pt_BR" />
        </h:outputText>
    </p:column>
    <p:column headerText="Data">
        <h:outputText value="#{livro.dataLancamento.time}">
            <f:convertDateTime pattern="dd/MM/yyyy"
                timeZone="America/Sao_Paulo" />
        </h:outputText>
    </p:column>

    <p:column headerText="Alterar">
        <h:commandLink value="altera" action="#{livroBean.carregar(livro)}"/>
    </p:column>

    <p:column headerText="Remover">
        <h:commandLink value="remove" action="#{livroBean.remover(livro)}"/>
    </p:column>
</p:dataTable>
```

Ordenando os livros

Com isso, o visual da tabela já está praticamente pronto, o que podemos fazer agora é adicionar novos recursos. Na [página][1] do `dataTable` há os recursos que ele oferece. O primeiro recurso que utilizaremos é o [**Sort**][2], que serve para ordenar as coluna baseado nos valores que quisermos. Vamos fazer isso para todas as tabelas com os valores do livro, basta adicionar o atributo `sortBy` na coluna, com o valor que queremos ordenar:

```
<p:dataTable value="#{livroBean.livros}" var="livro" id="tabelaLivros">
    <f:facet name="header">Livros</f:facet>
    <p:column headerText="Título" sortBy="#{livro.titulo}">
        <h:outputText value="#{livro.titulo}" />
    </p:column>
    <p:column headerText="ISBN" sortBy="#{livro.isbn}">
        <h:outputText value="#{livro.isbn}" />
    </p:column>
    <p:column headerText="Preço" sortBy="#{livro.preco}">
        <h:outputText value="#{livro.preco}">
            <f:convertNumber type="currency" pattern="R$ #0.00"
                currencySymbol="R$" locale="pt_BR" />
        </h:outputText>
    </p:column>
    <p:column headerText="Data" sortBy="#{livro.dataLancamento.time}">
        <h:outputText value="#{livro.dataLancamento.time}">
            <f:convertDateTime pattern="dd/MM/yyyy"
```

```
                timeZone="America/Sao_Paulo" />
        </h:outputText>
    </p:column>

    <p:column headerText="Alterar">
        <h:commandLink value="altera" action="#{livroBean.carregar(livro)}"/>
    </p:column>

    <p:column headerText="Remover">
        <h:commandLink value="remove" action="#{livroBean.remover(livro)}"/>
    </p:column>
</p:dataTable>
```

Ajustando a camada de persistência

Agora podemos testar... Mas não funciona! O problema está na nossa aplicação, quando clicamos na coluna que queremos ordenar, o Hibernate seleciona os dados no banco de dados, é assim toda vez que clicamos em alguma coluna, por isso o Primefaces não consegue ordená-los. Precisamos fazer um *cache* dos dados da nossa aplicação, para aí o Primefaces conseguir ordená-los.

Precisamos alterar o método `getLivros`, da classe `LivroBean`. Esse método que é chamado cada vez que clicamos em alguma coluna, e ele lista todos os livros chamando o método `listaTodos` do DAO. Isso não faz sentido, não precisamos carregar os livros toda hora, só precisamos listar os livros quando a sessão for iniciada ou quando adicionamos um livro, de resto podemos trabalhar com eles em memória.

Para resolver isso, vamos alterar a classe `LivroBean`, primeiro criando na classe um atributo `livros`. Dentro do método `getLivros`, se esse atributo for nulo, ou seja, se ainda não tivermos carregado os livros, nós chamamos o método `listaTodos` do DAO para carregá-los; se o atributo não for nulo, significa que já carregamos os livros, então não precisamos fazer nada:

```
private List<Livro> livros;

public List<Livro> getLivros() {

    DAO<Livro> dao = new DAO<Livro>(Livro.class);

    if(this.livros == null) {
        this.livros = dao.listaTodos();
    }

    return livros;
}
```

Mas também precisamos atualizar a lista de livros se adicionarmos algum novo, caso contrário ele será listado somente se recarregarmos a página. Então, no método `gravar`, vamos listar os livros sempre que um for adicionado:

```
public void gravar() {
    System.out.println("Gravando livro " + this.livro.getTitulo());

    if (livro.getAutores().isEmpty()) {
        FacesContext.getCurrentInstance().addMessage("autor",
                new FacesMessage("Livro deve ter pelo menos um Autor."));
```

```
            return;
        }

        DAO<Livro> dao = new DAO<Livro>(Livro.class);

        if(this.livro.getId() == null) {
            dao.adiciona(this.livro);

            // Novo livro adicionado, listamos todos os livros novamente
            this.livros = dao.listaTodos();
        } else {
            dao.atualiza(this.livro);
        }

        this.livro = new Livro();
    }
```

Reiniciando o Tomcat e testando a ordenação, vemos que agora tudo funciona corretamente! Podemos voltar a testar novos recursos do `dataTable` do Primefaces.