01

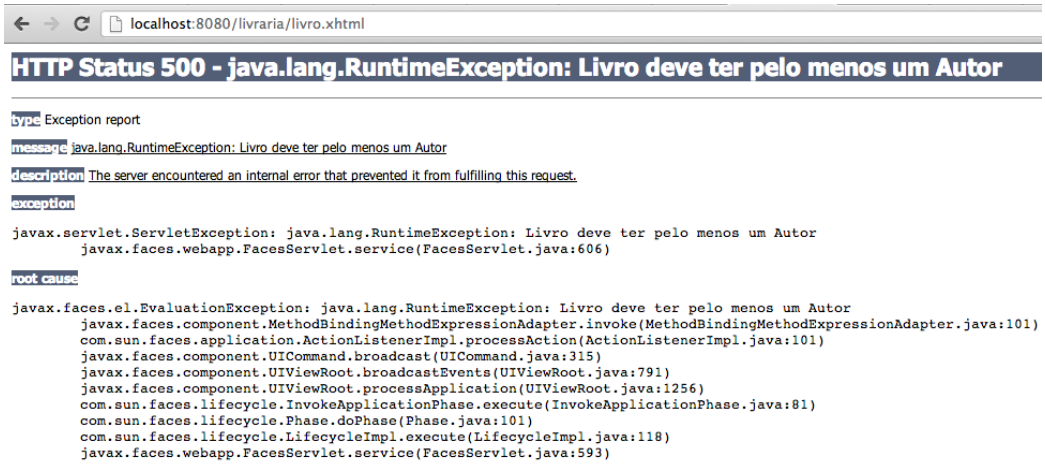
Completando o sistema e lidando com escopos do JSF 2

Transcrição

##Downloads Caso queira começar o treinamento a partir desse vídeo, pode baixar o projeto aqui (http://s3.amazonaws.com/caelum-online-public/JSF/livraria-capitulo4.zip). Só baixe este arquivo se **não tiver feito os exercícios dos capítulos anteriores.**

##Relacionamento entre Livro e Autor

Nesta aula completaremos o formulário do Livro e criaremos o relacionamento entre Livro e Autor. A nossa aplicação já está rodando e vamos testar uma vez a inserção de um livro. Ao preencher o formulário e apertar o botão para gravar, recebemos uma exceção.



A mensagem da exceção indica que **não podemos gravar um livro sem autor**. Isso faz sentido, pois não deve existir um livro sem autor.

O teste se o livro possui um Autor ou não já foi implementado nesse projeto. Ao abrir o `LivroBean`, notamos que dentro do método `gravar()` há um `if` que verifica se existe pelo menos um Autor, caso contrário, lança a exceção já vista.

```
@ManagedBean
public class LivroBean {

    //outros trechos omitidos

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

        if(livro.getAutores().isEmpty()) {
            throw new RuntimeException("Livro deve ter pelo menos um Autor");
        }

        new DAO<Livro>(Livro.class).adiciona(this.livro);
    }
}
```

##Criação e definição do combobox

Vamos implementar o relacionamento no formulário livro.xhtml. Para selecionar o Autor do Livro, adicionaremos um *combobox*.

Para isso criaremos um novo fieldset e, dentro dele, um legend indicando a seção do autor. Teremos um outputLabel com o valor *Selecione Autor*. O componente que representa um combobox se chama `h:selectOneMenu`. Vamos adicioná-lo depois do `h:outputLabel`.

Além do combobox, também criamos um botão para confirmar a seleção do autor. O `h:commandButton` com o valor *Gravar Autor*.

```
<fieldset>
    <legend>Dados do Autor</legend>
    <h:outputLabel value="Selecione Autor:" for="autor"/>
    <h:selectOneMenu  id="autor"/>
    <h:commandButton value="Gravar Autor" />
</fieldset>
```

A estrutura inicial já está pronta para ser visualizada no navegador. Ao atualizar a página, aparece a seção do autor com o combobox vazio.

##Preenchendo o h:selectOneMenu

O próximo passo é o preenchimento do `selectOneMenu` com os autores e o usuário fará isso, selecionando o nome do autor. Para tal, precisamos preencher o `selectOneMenu` com "*items*". Isto é feito por um outro componente que não faz parte do namespace "`h`", sendo necessário declarar uma nova biblioteca no início da página. Ela também já vem com a especificação e tem a URI *http://java.sun.com/jsf/core (http://java.sun.com/jsf/core)*. Utilizaremos o apelido padrão "`f`" para referenciá-la.

```
<html xmlns="http://www.w3.org/1999/xhtml"
        xmlns:h="http://java.sun.com/jsf/html"
        xmlns:f="http://java.sun.com/jsf/core">
```

Após a declaração, podemos utilizar os novos componentes, no nosso caso, `f:selectItems`. Para especificar os itens do combobox, utilizaremos o atributo `value` com a expression language `#{livroBean.autores}`. Nessa expressão, chamaremos um método da classe `LivroBean`, que devolverá uma lista de autores.

```
<fieldset>
    <legend>Dados do Autor</legend>
    <h:outputLabel value="Selecione Autor:" for="autor"/>
    <h:selectOneMenu id="autor">
        <f:selectItems value="#{livroBean.autores}" />
    </h:selectOneMenu>
    <h:commandButton value="Gravar Autor" action="#{livroBean.gravarAutor}"/>
</fieldset>
```

Ao abrir a classe, adicionaremos o método `getAutores()`. Dentro do mesmo, aproveitaremos o DAO, que possui o método `listaTodos()`.

```
@ManagedBean
public class LivroBean {

    //outros trechos omitidos

    public List<Autor> getAutores() {
        return new DAO<Autor>(Autor.class).listaTodos();
    }
}
```

Voltando ao formulário, percebemos que falta definir o que queremos apresentar de cada autor. Vamos mostrar o nome do autor indicado pelo atributo `itemLabel`. Para isso é preciso ter uma variável autor. Ou seja, o `f:selectItems` vai disponibilizar para cada item um autor. Baseado nesse autor, vamos usar a expression language `#{autor.nome}` para acessar o atributo nome. Igualmente será associado um valor ao item. Usaremos a ID do autor, ou seja, `#{autor.id}`.

```
<!-- outros codigos omitidos -->
<h:selectOneMenu id="autor">
    <f:selectItems value="#{livroBean.autores}" var="autor"
        itemLabel="#{autor.nome}" itemValue="#{autor.id}"/>
</h:selectOneMenu>
```

Já podemos visualizar o resultado no navegador. Ao atualizar a página, aparece o combobox populado.

##Recuperando dos valores dentro do ManagedBean

Agora vamos implementar o botão *Gravar Autor* e chamar um método ao clicar no botão. Mas antes disso é preciso saber qual autor o usuário selecionou no combobox. O componente `h:selectOneMenu`, como outros inputs, também possui um atributo `value` que usaremos para capturar a ID do autor selecionado.

```
<h:selectOneMenu value="#{livroBean.autorId}" id="autor">
    <f:selectItems value="#{livroBean.autores}" var="autor"
        itemLabel="#{autor.nome}" itemValue="#{autor.id}"/>
</h:selectOneMenu>
```

No `f:selectItems` definimos como `itemValue` a ID do autor, e justamente essa ID que recebemos na classe `LivroBean`. Vamos então criar no `LivroBean` um atributo `autorId` com um *getter* e *setter* para capturar a ID do autor.

```
@ManagedBean
public class LivroBean {

    //outros trechos omitidos

    private Integer autorId;

    public void setAutorId(Integer autorId) {
        this.autorId = autorId;
    }

    public Integer getAutorId() {
        return autorId;
    }
```

```
    //outros trechos omitidos
}
```

Para finalizar devemos criar o método para gravar o autor. Aqui não há novidade. Usaremos a expression language `#{livroBean.gravarAutor}` para chamar o método do `LivroBean`.

Vamos então voltar à classe para implementar o método. O método na verdade não grava no banco, e sim associa apenas o autor com o livro usando o relacionamento já definido, mas para isso é preciso carregar o autor, pois temos apenas a ID dele. Usaremos o DAO que possui um método `buscaPorId(id)` passando a ID do autor (autorId) como parâmetro. O retorno do método é o autor selecionado.

Depois da busca do autor, vamos relacionar o livro com o autor pelo método auxiliar `adicionaAutor()`. Pronto! E quando gravarmos o livro, o JPA também criará o relacionamento no banco de dados.

```
@ManagedBean
public class LivroBean {

    //outros trechos omitidos

    public void gravarAutor() {
        Autor autor = new DAO<Autor>(Autor.class).buscaPorId(this.autorId);
        this.livro.adicionaAutor(autor);
    }

    //outros trechos omitidos
}
```

##Resumo e teste dos componentes

Resumindo, o `f:selectItems` recebe uma lista de autores. No mesmo componente definimos o que queremos mostrar do autor e qual é o valor associado (e enviado no request). Para receber o autor selecionado, definimos o atributo `autorId`. Para associar o autor com o livro, criamos o método `gravarAutor()`, que é chamado pelo commandButton.

Chegou a hora de testar a tela no navegador, mas antes vamos reiniciar o servidor. Ao atualizar a página aparece no navegador e o combobox preenchido com os autores. Ótimo!

Vamos preencher os dados do livro. São eles: **titulo**, **isbn**, **preço** e **data**, e depois selecionaremos o autor. Não podemos esquecer de apertar o botão *Gravar Autor* para criar o relacionamento. No final gravamos o livro.

##Entendendo os escopos no desenvolvimento web

Para nossa surpresa recebemos novamente uma exceção, e pior, recebemos a mesma exceção acusando que o livro não possui um autor. Vamos tentar de novo, mas agora, para deixar mais claro, imprimimos o nome do autor no método `gravarAutor()` para verificar a existência do autor.

Ao voltar no formulário, vamos gravar o autor e verificar o console. Aqui está tudo certo, o autor foi relacionado com livro. A implementação está certa. Vamos testar de novo e gravar o livro, mas infelizmente o exceção continua.

Para entender o que aconteceu, vamos visualizar o fluxo da aplicação. Após ter preenchido o formulário, selecionamos o autor e apertamos o botão *Gravar Autor*. Isso disparou um request que o controlador passou para a árvore de

componentes, a nossa tela em memória. Como associamos os componentes com o `LivroBean`, o controlador cria um objeto dessa classe e junto com o `LivroBean`, também é criado o Livro.

No método `gravarAutor()`, carregamos o Autor associando-o com o Livro. Até aqui é tudo como o esperado. Após isso, apertamos no formulário o botão *Gravar* para realmente gravar o Livro. Isso causou um novo request que novamente cai na nossa tela, porém agora é criado um **NOVO** `livroBean` e consequentemente um novo livro. Perdemos o relacionamento do livro com o autor pois o `LivroBean` antigo não existe mais.

Em geral, cada request causa a criação de um novo `LivroBean`, pois a vida do **Managed Bean** dura apenas um request. Isso também se chama "*escopo de requisição*".

Porém, na nossa aplicação, queremos uma vida mais longa pra ele. Queremos que o `LivroBean` exista enquanto a tela existir. Esse novo escopo se chama **`ViewScoped`** e sobrevive a vários requests.

##Usando ViewScoped

A configuração do escopo no `ManagedBean` também é feita através de anotações. O padrão é o escopo de requisição que podemos deixar explícito com a anotação `@RequestScoped`. Para configurar o `ViewScoped` usaremos a anotação `@ViewScoped`.

```
@ManagedBean
@ViewScoped
public class LivroBean {
```

Após reiniciar o servidor e atualizar a página, testamos novamente no navegador. Vamos inserir os dados e selecionar o Autor. Não podemos esquecer de apertar o botão *Gravar Autor*. Agora podemos gravar o livro.

Dessa vez não recebemos nenhuma exceção, e ao verificar o console no Eclipse, podemos ver o SQL gerado pelo JPA. Foi inserido o livro e o relacionamento no banco de dados. Perfeito!

##Exibindo autores do livro com o h:dataTable

Agora vamos melhorar mais um pouco a usabilidade da aplicação. Um livro pode ser escrito por vários autores e faz sentido mostrar todos os autores do livro antes de gravar o mesmo.

Para que isso seja feito, vamos renderizar uma tabela abaixo do combobox. Mãos a obra, então. O componente `h:dataTable` recebe uma lista pelo atributo `value`, novamente pela expression language `#{livroBean.autoresDoLivro}`.

```
<fieldset>
    <legend>Dados do Autor</legend>

    <!-- outros trechos omitidos -->

    <h:dataTable value="#{livroBean.autoresDoLivro}" >

    </h:dataTable>
</fieldset>
```

Já no `LivroBean`, criaremos um método que devolve uma lista de autores. Ao abrir a classe, vamos criar o método getAutoresDoLivro(), que devolve os autores do livro atual.

```
@ManagedBean
@ViewScoped
public class LivroBean {

    //outros trechos omitidos

    public List<Autor> getAutoresDoLivro() {
        return this.livro.getAutores();
    }
```

Agora só falta declarar a variável que representa um autor. Depois disso vamos definir o que queremos mostrar em cada coluna. Para isso usaremos o componente `h:column`, que conterá o componente `h:outputText`. O outputText usa a variável para declarar o nome do autor pela expression language. No nosso caso vamos usar apenas uma coluna.

```
<fieldset>
    <legend>Dados do Autor</legend>

    <!-- outros trechos omitidos -->

    <h:dataTable value="#{livroBean.autoresDoLivro}" var="autor">
        <h:column>
            <h:outputText value="#{autor.nome}"/>
        </h:column>
    </h:dataTable>
</fieldset>
```

Após reiniciar o servidor e atualizar a página, testamos novamente o formulário. Vamos inserir os dados e selecionar o primeiro Autor. Ao apertar o botão, aparece o nome do autor abaixo do combobox. A mesma coisa acontece para o segundo e terceiro autor.