01

Melhorando a página dos autores

Transcrição

Começando daqui? Você pode fazer o DOWNLOAD (https://s3.amazonaws.com/caelum-online-public/jsf-primefaces/stages/capitulo-3.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo.

Melhorando a página dos autores

Com a página de login terminada, podemos focar agora na página `autor.xhtml`. A primeira coisa que temos que fazer, assim como em `login.xhtml`, e declarar o *namespace* do Primefaces dentro da tag `<html>`, juntamente com os outros *namespaces*:

```
xmlns:p="http://primefaces.org/ui"
```

Com o *namespace* declarado, vamos começar pelo **formulário de cadastro de autores**. Alguns componentes nós já vimos, como o `<p:outputPanel>`; o `<p:fieldset>`, que já tem um atributo `legend`; `<p:outputLabel>`; `<p:inputText>` e o `<p:message>`. Então vamos começar fazendo as alterações que envolvem esses componentes:

```
<ui:define name="titulo">
    <p:outputPanel>Novo Autor</p:outputPanel>
</ui:define>

<ui:define name="conteudo">
    <h:form id="autor">
        <p:fieldset legend="Dados do Autor">
            <h:panelGrid columns="3">

                <p:outputLabel value="Nome:" for="nome" />
                <p:inputText id="nome" value="#{autorBean.autor.nome}" required="true">
                    <f:validateLength minimum="5" />
                    <f:ajax event="blur" render="messageNome" />
                </p:inputText>
                <p:message for="nome" id="messageNome" />

                <p:outputLabel value="Email:" for="email" />
                <p:inputText id="email" value="#{autorBean.autor.email}" required="true">
                    <f:passThroughAttribute name="type" value="email" />
                </p:inputText>

                <p:message for="email" id="messageEmail" />

                <p:commandButton value="Gravar" action="#{autorBean.gravar}" />
            </h:panelGrid>
        </p:fieldset>
    </h:form>
    <!-- formulário de exibição dos autores -->
</ui:define>
```

Como o `commandButton` do Primefaces já usa ajax, precisamos definir o que queremos enviar, que no nosso caso é o formulário, e o que queremos atualizar. Além de atualizar o formulário, também precisamos atualizar o formulário que exibe os autores, vamos referenciá-lo através do seu id `formTabelaAutors` :

```
<p:commandButton value="Gravar" action="#{autorBean.gravar}" process="@form" update="@form :forr
```

Pronto, mas o que fizemos até agora foi basicamente uma revisão do capítulo anterior, ainda não há nenhuma novidade.

O campo de e-mail

Quem se lembra do treinamento anterior, sabe que utilizamos um *passThroughAttribute* no campo de e-mail, porque o JSF ainda não dá suporte para todas as tags do HTML5, então podemos definir um atributo do HTML5 que gostaríamos de renderizar.

Mas se olharmos o código fonte da nossa página, vemos um problema. Dizemos que o `input` deveria ser do tipo `email` , mas o componente do Primefaces também disse que o `input` é do tipo `text` , ou seja, nosso `input` tem dois tipos, está duplicado. O problema é que o *passThroughAttribute* é relativamente recente, e o Primefaces já dava suporte a uma forma de passar atributos para o `inputText` , utilizando o `f:attribute` :

```
<p:outputLabel value="Email:" for="email" />
<p:inputText id="email" value="#{autorBean.autor.email}" required="true">
    <f:attribute name="type" value="email" />
</p:inputText>
```

Agora o `type="text"` desapareceu, então tudo deveria funcionar corretamente, assim como a validação do e-mail, mas não é isso que acontece. O que está acontecendo é que o `p:commandButton` utiliza ajax, como já sabemos, então nós não passamos pela validação do navegador.

Para resolver esse problema, nós vamos utilizar um componente que já existe na especificação JSF, o *validator*. Iremos utilizar um `validatorRegex` , que aplicará um padrão em cima dos dados que usuário digitar, ou seja, esses dados terão que coincidir com o padrão que nós definirmos. E o padrão que iremos definir é que o nosso e-mail terá qualquer caractere (.) uma ou mais vezes (+), depois terá que vir um arroba (@) juntamente com qualquer caractere (.), novamente uma ou mais vezes (+). Logo, nossa *regex* será `.+@.+` :

```
<p:outputLabel value="Email:" for="email" />
<p:inputText id="email" value="#{autorBean.autor.email}" required="true">
    <f:attribute name="type" value="email" />
    <f:validateRegex pattern=".+@.+" />
</p:inputText>
```

Além disso, queremos que isso funcione com ajax (assim como o nome) para renderizar uma mensagem e vamos adicionar mais um atributo do HTML5, o `placeholder` , com o texto "Email do autor":

```
<p:inputText id="email" value="#{autorBean.autor.email}" required="true">
    <f:attribute name="type" value="email" />
    <f:passThroughAttribute name="placeholder" value="Email do autor" />
    <f:validateRegex pattern=".+@.+" />
```

```
    <f:ajax event="blur" render="messageEmail" />
</p:inputText>
```

Agora, se colocarmos um e-mail errado, ou seja, sem arroba (@) recebemos uma mensagem de erro de validação, mas é uma mensagem bem complicada do usuário entender. Para melhorar isso, podemos definir uma mensagem de validação no `inputText`, através do atributo `validatorMessage`:

```
<p:inputText id="email" value="#{autorBean.autor.email}" required="true" validatorMessage="Emai
    <f:attribute name="type" value="email" />
    <f:passThroughAttribute name="placeholder" value="Email do autor" />
    <f:validateRegex pattern=".+@.+" />
    <f:ajax event="blur" render="messageEmail" />
</p:inputText>
```