01

Páginas mais dinâmicas com AJAX

Transcrição

Caso queira começar o treinamento a partir desse vídeo, pode baixar o projeto aqui (http://s3.amazonaws.com/caelum-online-public/JSF/livraria-capitulo6.zip). Só baixe este arquivo se **não tiver feito os exercícios dos capítulos anteriores**.

Melhorando a usabilidade da aplicação

No último vídeo vimos como o JSF trata a conversão e validação de dados. Alteramos o formulário na página **livro.xhtml** para converter a data e apresentar no formato brasileiro. Nos outros *inputs* aplicamos regras de validação como no exemplo do titulo que verifica a existência do valor. Personalizamos a validação com nossa regra dentro do **LivroBean**. Para tal, criamos um método que executa uma validação e gera uma mensagem JSF caso haja algum problema. Para enfileirar uma mensagem `FacesMessage`, vimos como usar o `FacesContext`.

Voltando ao *xhtml*, aprendemos que os *converters* também se aplicam para `h:outputText` para formatar, por exemplo, preço ou data do livro. No **autor.xhtml** aplicamos a validação e aprendemos que `h:message` pode ser usado para mostrar a mensagem de um campo específico.

Vamos inicializar a aplicação uma vez e testar a interface. Ao chamar o **livro.xhtml** e gravar um livro inválido, o JSF mostra as mensagens de validação. Vamos testar também o formulário do autor. Um autor sem nome ou com menos de 5 caracteres é considerado inválido, e é sinalizado pelas mensagens ao lado do campo.

Vamos voltar ao formulário do livro e atualizar a tela para começar do zero. Agora podemos perceber um comportamento estranho. Repare que o formulário não aceita a inserção de um autor se o livro a cadastrar possuir campos com falhas de validação. No ponto de vista do formulário há de fato um problema com a validação, mas no ponto de vista do usuário não tentamos gravar o livro ainda, queremos apenas selecionar um autor.

Vamos analisar esse caso para entender melhor o que está acontecendo. Ao apertar o botão **Gravar Autor** foi enviada uma requisição para chamar o método `gravarAutor()` dentro do **LivroBean**. Mas antes disso o controlador passa todos os parâmetros da requisição para a árvore de componentes. O problema é que, ao apertarmos o botão, não só enviamos o `ID` do autor como também todos os outros dados do formulário. Acontece que os componentes receberam os valores para `titulo`, `isbn` etc, mas tudo em branco. Como associamos validações aos *inputs*, os componentes recusam-se a executar o método `gravarAutor()` do **LivroBean**, pois de fato há um problema de validação.

Entendendo AJAX em nossa aplicação

Para resolver isso vamos tentar enviar apenas uma parte do formulário. Ou seja, vamos enviar apenas os dados do autor. Não enviar todas as informações também pode ser vantajoso porque o processamento dos dados no lado do servidor pode ser demorado. Nesse caso o navegador espera sincronamente até a resposta chegar. Ele fica aguardando a resposta para atualizar a tela inteira e causa a famosa tela branca. Isso cria uma experiência ruim para o usuário final.

Então vamos mandar uma requisição por trás da interface, que não necessita de uma atualização completa da tela, e envia apenas uma parte do formulário. O usuário nem deve sentir que houve essa requisição. Assim que a resposta chegar, atualizamos apenas a parte da tela em questão. Essa forma de enviar as requisições se chama **AJAX**.

Aliando JSF 2 e AJAX

AJAX melhora a usabilidade da interface e, graças aos componentes JSF, pode ser feito de uma forma muito simples. Não é preciso saber detalhes sobre JavaScript e a atualização programática da tela. Tudo isso é feito de forma

transparente para o desenvolvedor, bastando apenas ativar o AJAX no lugar desejado. Para tal existe um componente: `f:ajax`

Vamos associar `f:ajax` com o `h:commandButton`. O botão sabe então que vai enviar uma requisição AJAX que **NÃO** submete o formulário inteiro. Falta então definir o que queremos enviar nessa requisição. Claro, apenas o valor do `h:combobox`. Para isso existe o atributo `execute`, nele definimos os `IDs` dos componentes que queremos enviar na requisição.

```
<h:commandButton value="Gravar Autor" action="#{livroBean.gravarAutor}" >
    <f:ajax execute="autor" />
</h:commandButton>
```

Vamos adicionar no componente `h:selectOneMenu` o `ID`. O mesmo `ID` é utilizado no atributo `execute` do `f:ajax`. Assim o comando sabe que queremos enviar o valor do componentes na requisição.

```
<h:selectOneMenu value="#{livroBean.autorId}" id="autor">
    <!-- codigo omitido -->
</h:selectOneMenu>
```

Por último falta adicionar no início da página o componente `h:head`. Como **AJAX** é baseado no *JavaScript*, que roda no navegador, o JSF precisa associar no cabeçalho uma biblioteca JavaScript. Para tal precisa ter o `h:head`.

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

    <h:head />

    <h:body>
            <!-- codigo omitido -->
    </h:body>
</html>
```

Ao recarregar a página podemos visualizar o código fonte dela (no navegador Chrome e Firefox, pressione ctrl + u). O código gerado está um pouco mal formatado, mas podemos procurar o elemento `head` e lá dentro podemos ver a arquivo *JavaScript* adicionado automaticamente pelo JSF. É este arquivo que possui as funções **AJAX**, mas ainda bem que não precisamos nos preocupar com esses detalhes.

Vamos fechar essa janela e testar o formulário. Ao apertar o botão **Gravar Autor** parece que nada acontece na aplicação. Ao verificar o console do Eclipse podemos ver que o método `gravarAutor()` foi chamado, ou seja, o JSF fez a requisição **AJAX** e enviou o *ID* do Autor selecionado para o servidor.

Renderizando componentes do JSF com AJAX

Faltou ainda definir qual parte da página queremos atualizar. Quando selecionarmos o Autor seu nome deve aparecer na tabela abaixo do `h:combobox`. Aqui também é preciso configurar o componente `f:ajax`. O atributo **render** recebe os *ID*s dos componentes que queremos atualizar, no nosso caso o *ID* da tabela de autores. Como a tabela não possui ainda um *ID*, colocaremos um agora repetindo-o no atributo `render`. Pronto!

```
<h:commandButton value="Gravar Autor" action="#{livroBean.gravarAutor}" >
    <f:ajax execute="autor" render="tabelaAutores"/>
</h:commandButton>
```

Ao voltar no navegador e testar novamente o botão gravar, o Autor é enviado e a requisição **AJAX** atualiza a tabela. Repare que os dados do livro não foram enviados, não houve validação do titulo. Houve uma atualização parcial da página.

AJAX nos componentes de entrada

Ótimo, usamos **AJAX** com um `h:commandButton`, mas `f:ajax` não se limita aos comandos e também pode ser aplicado nos componentes de *input*. Vamos mostrar isso no `h:inputText` do título. O `f:ajax` dentro de um *input* significa que os dados serão enviados via **AJAX**. Aqui há algumas opções para definir o momento exato da requisição. Podemos declarar que cada vez que soltamos uma tecla deve ser enviado uma requisição, ou quando o *input* recebe (ou perde) o `focus`. No exemplo submetemos os dados quando o campo perde o `focus`, indicado pelo atributo `event` com o valor `blur`.

Igual a parte anterior, também podemos especificar o que queremos atualizar após a requisição. A vantagem de usar **AJAX** no *input* é que a validação é executada na hora, quando o usuário digita no campo. Novamente irá melhorar a usabilidade, mas isso só funciona se atualizarmos as mensagens para apresentar possíveis problemas de validação. Aqui também é preciso fazer a associação com o *ID* do componente.

```
<h:inputText id="titulo" value="#{livroBean.livro.titulo}" required="true"
    requiredMessage="Título obrigatório" validatorMessage="Título não pode ser superior a 40">
            <f:validateLength maximum="40" />
            <f:ajax event="blur" render="messages"/>
</h:inputText>
```

No navegador vamos atualizar página e testar o titular. Vamos deixar o campo em branco. Repare que, ao perder o `focus`, aparece na mesma hora o erro de validação.

Enviando o formulário inteiro

Por último vamos usar **AJAX** também no botão que grava o livro. Enviaremos todas as informações do livro com **AJAX**. Aqui não há novidade, temos a tag `f:ajax` dentro do botão. Também precisamos declarar os *IDs* dos componentes que queremos incluir na requisição. No atributo `render` podemos usar uma lista de *ID*s, porém muitos *ID*s ficam trabalhosos, facilitando erros. Por este motivo existe um atalho, o `@form`, para indicar o envio do formulário inteiro. Além do `@form` há outras possibilidades, como por exemplo `@all`. Ele indica que queremos enviar a página inteira.

Além disso, é preciso definir o que queremos atualizar. Nesse exemplo é preciso atualizar o formulário, indicado pelo `@form` e a tabela para o mostrar o novo livro. A tabela será referenciada novamente pelo *ID*, como já fizemos, porém repare que elas não estão no mesmo formulário. Colocando o *ID* apenas, significa que o `f:ajax` procurará o componente no mesmo formulário. Para encontrar o componente fora do form é preciso usar o caminho absoluto indicado pelo **":"** na frente do *ID*.

```
<h:commandButton value="Gravar" action="#{livroBean.gravar}" >
    <f:ajax execute="@form" render="@form :tabelaLivros"/>
</h:commandButton>
```

Revisando as funcionalidades da aplicação

Vamos mostrar uma vez todas as funcionalidades abrindo novamente o navegador. Ao apertar o botão **Gravar** será enviado um *request* **AJAX**. Como há dados inválidos, as mensagens de validação são mostradas. Depois preenchemos o campo **Titulo**, que envia outro *request* **AJAX** após perder o *focus*. Ao preencher os outros dados e gravar o livro será testada a existência do Autor. Na escolha do Autor também usamos **AJAX** para enviá-lo e atualizar a sua tabela. No final podemos cadastrar o Livro, que causa a atualização do formulário e da tabela de livros. Repare que em nenhum momento a página foi atualizada totalmente, pois utilizamos **AJAX**.