04

Atualizando para JSF 2.2

Começando daqui? Você pode fazer o DOWNLOAD (https://s3.amazonaws.com/caelum-online-public/JSF/livraria-capitulo10.zip) do projeto completo do capítulo anterior e continuar seus estudos a partir deste capítulo.

Atualização do JSF 2.2

Quando começamos o nosso projeto de livraria, usamos a versão 2.1 do JSF. Nesse meio tempo já saiu uma versão mais recente, a versão 2.2, razão suficiente para uma atualização, não?

O primeiro passo é baixar o novo JAR. Pegaremos o JAR do repositório do Maven através deste link (http://mvnrepository.com/artifact/org.glassfish/javax.faces/2.2.13).

O link está na explicação e no exercício também. O JAR novo ficará na pasta `WebContent/WEB-INF/lib`, mas não podemos esquecer de apagar a versão antiga do JAR.

Uma vez copiado o JAR, devemos deixar claro no arquivo de configuração `faces-config.xml` que estamos usando a versão 2.2. Faremos isso pelo novo cabeçalho que indica a versão, **substituindo o antigo**:

```
<faces-config version="2.2"
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/w

  <!-- resto omitido -->
```

Novo namespace do JSF 2.2

O JSF 2.2 trouxe vários bug-fixes, comparado com a versão 2.1, mas também algumas novidades. Veremos duas aqui nesse curso, as *viewActions* e *pass through atributes*.

No entanto, antes de realmente usar uma nova feature do JSF 2.2, devemos atualizar o *namespace* do mesmo! O que? Se lembra que todos os componentes JSF usam um prefixo, como `h:` ou `f:`? Esse prefixo na verdade é um atalho para **não escrever** uma URL (ou *namespace*) completa.

Até a versão JSF 2.2, todas as URLs começavam com `http://java.sun.com`, já que foi a antiga empresa **Sun** que começou com o desenvolvimento JSF. O tempo passou e a **Sun** nem existe mais, por isso os membros da especificação mudaram a URL (o *namespace*) para `http://xmlns.jcp.org`.

Ou seja, devemos substituir o *namespace* `http://java.sun.com/jsf/html` por `http://xmlns.jcp.org/jsf/html`, ok?

Então a declaração no início da página ficará:

```
<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">
```

Faremos a atualização em cada página `xhtml`.

Novidades do JSF 2.2

Entre as novidades do JSF 2.2, escolhemos duas que são bem úteis no dia a dia de um desenvolvedor JSF!

Já falamos em uma aula anterior que no mundo JSF toda comunicação entre navegador e servidor é feita através de requisições HTTP POST. Apenas a primeira requisição, aquela que constrói a árvore de componentes, é um HTTP GET, a partir daí tudo é um POST.

No entanto, imagine que o cliente está trabalhando na aplicação de livraria e gostaria de passar um link para outro funcionário que mostra o formulário do autor. Sabemos que o link do formulário é http://localhost:8080/livraria/autor.xhtml (http://localhost:8080/livraria/autor.xhtml), só que esse formulário sempre fica vazio, pois ao chamar enviamos um GET que cria uma tela nova.

ViewAction

Queremos passar no formulário o autor a ser editado, e o formulário deverá vir preenchido automaticamente. Isso é algo muito comum na web e normalmente é feito através de um parâmetro na requisição GET, por exemplo:

http://localhost:8080/livraria/autor.xhtml?autorId=1 (http://localhost:8080/livraria/autor.xhtml?autorId=1)

Ou seja, queremos enviar um GET mas já populando o formulário com o autor que possui a id 1! Como fazer isso? Bom, JSF 2.2 trouxe um recurso para tal, que foi batizado de *viewAction*. Podemos associar, ao construir a view, uma ação que vai carregar o autor automaticamente!

Perfeito, mas como funciona? Nada complicado, devemos adicionar no início da página um *metadata*, que define qual parâmetro queremos ler, e qual é a *action*/método a chamar:

```
<f:metadata>
    <f:viewParam name="autorId" value="#{autorBean.autorId}"/>
    <f:viewAction action="#{autorBean.carregarAutorPelaId}"/>
</f:metadata>
```

Vamos adicionar essas linhas no início da página `autor.xhtml`, logo abaixo de `<ui:composition>`.

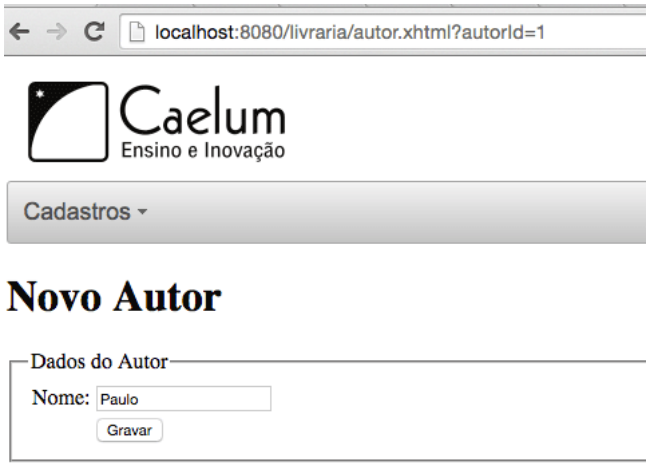
Agora falta preparar o `AutorBean`, para realmente carregar o autor pela id. Vamos adicionar o atributo `autorId`, seu *getter* e *setter*, e um novo método com o nome definido no atributo `action`:

```
private Integer autorId;

public Integer getAutorId() {
    return autorId;
}

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

```
public void carregarAutorPelaId() {
    this.autor = new DAO<Autor>(Autor.class).buscaPorId(autorId);
}
```

Isso já é todo o necessário para funcionar. Vamos testar e enviar uma requisição GET:

http://localhost:8080/livraria/autor.xhtml?autorId=1 (http://localhost:8080/livraria/autor.xhtml?autorId=1)



O formulário já vem preenchido, ou seja, a nossa *viewAction* foi executada.

Atributos passThrough

Tudo mundo já ouviu falar de HTML5, certo? HTML5 é a nova versão da linguagem HTML, que traz importantes mudanças, entre eles várias novas tags para criar páginas mais semânticas.

Podemos ver as novas funcionalidades e a documentação do HTML 5 diretamente no site do W3Schools:
http://www.w3schools.com/html/html_form_input_types.asp
(http://www.w3schools.com/html/html_form_input_types.asp)

Agora temos o problema que toda nossa página HTML é renderizada através dos componentes JSF. Perdemos o controle do HTML que é um *tradeoff* do mundo JSF. Mas felizmente o JSF 2.2 trouxe uma solução para aproveitar algumas novas tags do HTML5. Essa nova feature se chama **passThroughAttribute**.

A ideia é que podemos definir um atributo no componente, que é simplesmente passado para o HTML, sem alteração. Assim, podemos incluir atributos que são ignorados no mundo JSF, mas importam dentro do HTML.

Veja o exemplo abaixo, onde definimos um `inputText` que recebe um atributo `type="email"` usando o componente aninhado `f:passThroughAttribute` . Baseado nesse atributo o navegador executa uma validação do e-mail digitado, mas no mundo JSF não há nenhuma importância:

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

Assim podemos usufruir dos tipos de `input` s do HTML5, como `type="number"` ou `type="date"` , que não existem no JSF.

Vamos colocar o código no formulário dos dados do autor, da página `autor.xhtml`, logo após `h:message`:

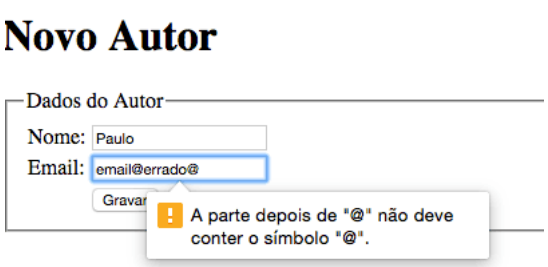
```
<h:form id="autor">
    <fieldset>
        <legend>Dados do Autor</legend>
        <h:panelGrid columns="3">

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

            <h:message for="nome" id="messageNome" />

            <!-- NOVO CÓDIGO AQUI -->

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

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

            <h:commandButton value="Gravar" action="#{autorBean.gravar}" />
        </h:panelGrid>
    </fieldset>
</h:form>
```

Agora só falta criar o e-mail dentro do nosso modelo, na classe `Autor`:

```
@Entity
public class Autor implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue
    private Integer id;
    private String nome;
    private String email; // Não esqueça de criar o gettter e setter
```

Vamos chamar novamente a página `autor.xhtml` (aproveitando a `viewAction`):

http://localhost:8080/livraria/autor.xhtml?autorId=1 (http://localhost:8080/livraria/autor.xhtml?autorId=1)



A validação do campo e-mail é executada pelo navegador e não pelo JSF. É tudo client-side!

O que aprendemos

- Usamos a versão 2.2 do JSF
- Usamos o novo *namespace* para declarar os componentes
- Testamos a `viewAction` para carregar um formulário pelo HTTP GET
- Testamos `passThroughAttribute` para definir um novo atributo no HTML