

Anotações e Navegação no JSF 2.0

Simplifique a navegação e a configuração XML com o JSF 2.0

Aprenda a utilizar anotações e navegações implícita e condicional em suas aplicações JSF 2.0

MARCEL TOKARSKI

A Internet vem evoluindo a cada ano. E com o advento da Web 2.0, o dinamismo e a interatividade dos sites com os usuários ganharam uma importância fundamental. Hoje, cada vez mais pessoas têm acesso à rede mundial de computadores, seja de suas residências ou de seus escritórios, através de um computador ou de dispositivos móveis.

Por conta disso, há uma demanda crescente por desenvolvimento para web, como a criação ou remodelagem de sites, redes sociais, sistemas para Intranet de uma empresa, etc. Neste contexto, o primeiro contato que o usuário tem com a aplicação é através de sua interface, uma de suas partes mais importantes.

A tecnologia JavaServer Faces (JSF), em sua essência, é um framework Java destinado à criação de componentes de interface gráfica para aplicações web. Ele simplifica o desenvolvimento da Interface com o Usuário, que geralmente é uma das partes mais trabalhosas no desenvolvimento web.

Neste artigo veremos, com exemplos práticos, algumas dessas novidades: o uso de anotações nos *beans* gerenciáveis (*managed beans*) e a navegação implícita e condicional. Porém, antes de entrarmos nos detalhes do JSF 2.0, vamos configurar um ambiente de desenvolvimento e criar um projeto web que será utilizado em todo o artigo.

Configurando o ambiente de desenvolvimento

Para a montagem do ambiente, utilizaremos o JDK 6 *Update* 21, o NetBeans IDE 6.9.1 para Java e o servidor de aplicações GlassFish versão 3.0.1.

A melhor maneira de apresentar as novidades do JSF 2.0, destacadas nesse artigo, é através de uma aplicação exemplo. Para isso, crie um novo projeto web no NetBeans com o nome de “ExemploJSF2”. Mantenha as configurações padrões durante a criação, e não se esqueça de selecionar o framework “JavaServer Faces” antes de finalizá-la.

Pronto! Agora temos um projeto web utilizando JSF 2.0. Notem que o NetBeans criou automaticamente quatro arquivos, como pode ser observado na **Figura 1**:

- *WEB-INF/web.xml*: arquivo descritor para aplicações web Java EE. Entre outras coisas, nesse arquivo é definida a página inicial da aplicação (neste caso, *index.xhtml*);
- *WEB-INF/sun-web.xml*: arquivo descritor de *deployment* para o servidor de aplicações GlassFish. Nesse arquivo é definido o contexto da aplicação (no caso */ExemploJSF2*);
- *index.xhtml*: página inicial JSF (padrão Facelets);
- *MANIFEST.MF*: arquivo de manifesto padrão.

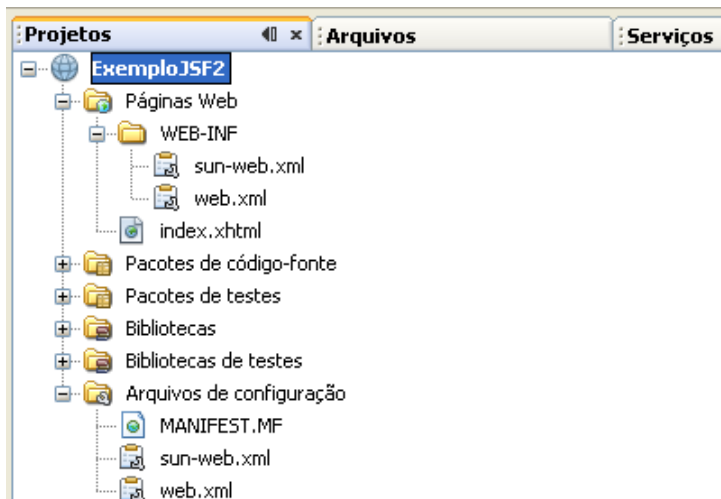


Figura 1. Projeto ExemploJSF2 criado no NetBeans.

Não iremos alterar esses arquivos, exceto o *index.xhtml*, no qual criaremos um formulário de cadastro de clientes. A Figura 2 mostra as páginas e os fluxos de navegação que construiremos no nosso projeto.

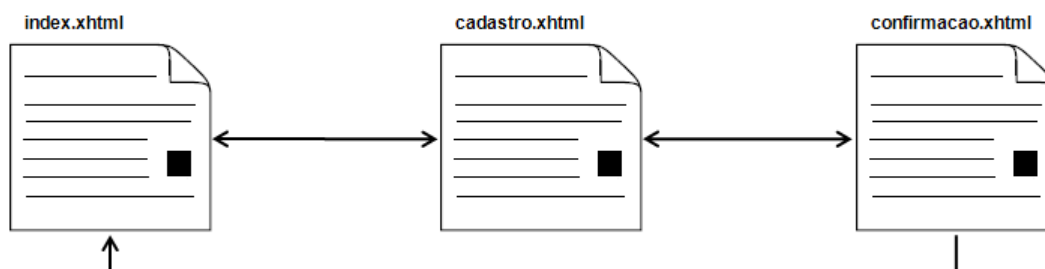


Figura 2. Páginas e fluxo de navegação do projeto ExemploJSF2.

Como podemos observar, trata-se de um projeto simples. Na página *index.xhtml*, existe uma lista de clientes cadastrados e uma opção para cadastro de novo cliente. A página *cadastro.xhtml* possui um formulário contendo as informações necessárias para a criação do cliente. E a página *confirmacao.xhtml* mostra as informações incluídas no formulário, para que o usuário verifique se os dados estão corretos e confirme a inclusão do novo cliente. Após a confirmação, a página *index.html* é mostrada novamente, com a lista de clientes atualizada. Além disso, o usuário também tem a opção de cadastrar um novo cliente sem a necessidade de confirmar os dados dele.

Navegação implícita e condicional

O modelo de navegação do JSF contempla as regras que definem como e quando a navegação entre páginas deve ocorrer em uma aplicação. Uma novidade apresentada no JSF 2.0 é a chamada **navegação implícita**, na qual não há a necessidade de configuração XML. O JSF 2.0 também apresenta uma novidade para a navegação baseada em regras (aquela definida através de configuração XML): a **navegação condicional**. Nela, é possível definir restrições no arquivo de configuração do JSF, o *faces-config.xml*. Veremos exemplos desses dois tipos.

Para começar, vamos criar a primeira página. No projeto ExemploJSF2, abra o arquivo *index.xhtml* e altere seu código padrão, conforme a **Listagem 1**.

Listagem 1. Arquivo *index.xhtml* contendo a lista de clientes.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<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>

```

```

<title>Exemplo simples de aplicação JSF 2.0</title>
</h:head>
<h:body>
  <h:form>
    <h3>Lista de Clientes</h3>
    <h:dataTable border="1" var="clienteLista" value="#{clienteBean.listaCliente}">
      <h:column>
        <f:facet name="header">Nome</f:facet>
        #{clienteLista.nome}
      </h:column>
      <h:column>
        <f:facet name="header">Data de Nascimento</f:facet>
        <h:outputText value="#{clienteLista.dataNascimento}">
          <f:convertDateTime pattern="dd/MM/yyyy" />
        </h:outputText>
      </h:column>
      <h:column>
        <f:facet name="header">Sexo</f:facet>
        #{clienteLista.sexo}
      </h:column>
      <h:column>
        <f:facet name="header">Cidade</f:facet>
        #{clienteLista.localidadeCliente.cidade}
      </h:column>
      <h:column>
        <f:facet name="header">Estado</f:facet>
        #{clienteLista.localidadeCliente.estado}
      </h:column>
    </h:dataTable>
    <br/>
    <h:commandButton value="Cadastrar" action="cadastro" />
  </h:form>
</h:body>
</html>

```

Na **Listagem 1** você pode notar a presença de algumas *tags* novas, introduzidas no JSF 2.0: **<h:head>** e **<h:body>**. Elas são componentes JSF que renderizam as *tags* HTML **<head>** e **<body>**, respectivamente.

Crie também as páginas *cadastro.xhtml* e *confirmacao.xhtml*, conforme as **Listagens 2 e 3**.

Listagem 2. Arquivo *cadastro.xhtml* contendo o formulário de criação de cliente.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<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>
    <title>Exemplo simples de aplicação JSF 2.0</title>
  </h:head>
  <h:body>
    <h:form>
      <h3>Cadastro de cliente</h3>
      <table>
        <tr>
          <td>Nome:</td>
          <td>
            <h:inputText id="nome"
              value="#{clienteBean.cliente.nome}" required="true" />
            <h:message for="nome" />
          </td>
        </tr>
        <tr>
          <td>Data de Nascimento:</td>
          <td>
            <h:inputText id="dataNasc"
              value="#{clienteBean.cliente.dataNascimento}" required="true">
              <f:convertDateTime pattern="dd/MM/yyyy" />
            </h:inputText> (dd/mm/yyyy)
            <h:message for="dataNasc" />
          </td>
        </tr>
        <tr>
          <td>Sexo:</td>
          <td>
            <h:selectOneMenu value="#{clienteBean.cliente.sexo}">

```

```

        <f:selectItems value="#{clienteBean.listaSexo}" var="sexo"
            itemLabel="#{sexo}" itemValue="#{sexo}"/>
    </h:selectOneMenu>
</td>
</tr>
<tr>
    <td>Cidade:</td>
    <td>
        <h:inputText id="cidade"
            value="#{clienteBean.cliente.localidadeCliente.cidade}"
            required="true" />
        <h:message for="cidade" />
    </td>
</tr>
<tr>
    <td>Estado:</td>
    <td>
        <h:inputText id="estado"
            value="#{clienteBean.cliente.localidadeCliente.estado}"
            required="true" />
        <h:message for="estado" />
    </td>
</tr>
</table>
<h:messages />
<br/>
<h:commandButton value="Criar" action="confirmacao" />
<h:commandButton value="Cancelar" immediate="true"
    action="#{clienteBean.cancelar}" />
</h:form>
</h:body>
</html>

```

Listagem 3. Arquivo *confirmacao.xhtml* com pedido de confirmação de criação de cliente.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<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>
        <title>Exemplo simples de aplicação JSF 2.0</title>
    </h:head>
    <h:body>
        <h:form>
            <h3>Cadastro de cliente - Confirmação</h3>
            <table>
                <tr>
                    <td>Nome:</td>
                    <td>
                        <h:outputText value="#{clienteBean.cliente.nome}"/>
                    </td>
                </tr>
                <tr>
                    <td>Data de Nascimento:</td>
                    <td>
                        <h:outputText value="#{clienteBean.cliente.dataNascimento}">
                            <f:convertDateTime pattern="dd/MM/yyyy" />
                        </h:outputText>
                    </td>
                </tr>
                <tr>
                    <td>Sexo:</td>
                    <td>
                        <h:outputText value="#{clienteBean.cliente.sexo}"/>
                    </td>
                </tr>
                <tr>
                    <td>Cidade:</td>
                    <td>
                        <h:outputText value="#{clienteBean.cliente.localidadeCliente.cidade}"/>
                    </td>
                </tr>
                <tr>
                    <td>Estado:</td>
                    <td>
                        <h:outputText value="#{clienteBean.cliente.localidadeCliente.estado}"/>
                    </td>
                </tr>
            </table>

```

```

        <tr>
            <td>Novo cadastro?</td>
            <td>
                <h:selectBooleanCheckbox value="#{clienteBean.novoCadastro}" />
            </td>
        </tr>
    </table>
    <br/>
    <h:commandButton value="Confirmar Criação" action="#{clienteBean.criarCliente}" />
    <h:commandButton value="Editar" action="cadastro"/>
</h:form>
</h:body>
</html>

```

Outra novidade do JSF 2 é que agora podemos utilizar o componente **f:selectItems**, como visto na **Listagem 2**, sem a necessidade de criar uma lista ou um *array* de **SelectItem** no *bean* gerenciável, implementação obrigatória nas versões anteriores.

Finalmente, crie as classes **Cliente**, **LocalidadeCliente** e **ClienteBean**, utilizadas pelas páginas JSF. A classe **Cliente** será responsável por conter as informações pessoais relacionadas ao cliente, e **LocalidadeCliente** pelas informações de localidade dele. Já a classe **ClienteBean** será nosso *bean* gerenciável. As **Listagens 4, 5 e 6** mostram os códigos destas classes.

Listagem 4. Código da classe Cliente.

```

package br.com.javamagazine;

import java.io.Serializable;
import java.util.Date;

public class Cliente implements Serializable {
    private String nome;
    private Date dataNascimento;
    private String sexo;
    private LocalidadeCliente localidadeCliente;

    // métodos getters e setters das propriedades
}

```

Listagem 5. Código da classe LocalidadeCliente.

```

package br.com.javamagazine;

import java.io.Serializable;

public class LocalidadeCliente implements Serializable {
    private String cidade;
    private String estado;

    // métodos getters e setters das propriedades
}

```

Listagem 6. Código da classe ClienteBean.

```

package br.com.javamagazine;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class ClienteBean implements Serializable {
    private List<Cliente> listaCliente;
    private List<String> listaSexo;
    private Cliente cliente;
    private boolean novoCadastro;

    public ClienteBean() {
        listaCliente = new ArrayList<Cliente>();
        listaSexo = new ArrayList<String>();
        listaSexo.add("Masculino");
        listaSexo.add("Feminino");
        cliente = new Cliente();
        cliente.setLocalidadeCliente(new LocalidadeCliente());
    }
}

```

```

}

public String criarCliente() {
    listaCliente.add(cliente);
    cliente = new Cliente();
    cliente.setLocalidadeCliente(new LocalidadeCliente());
    return "index";
}

public String cancelar() {
    cliente = new Cliente();
    cliente.setLocalidadeCliente(new LocalidadeCliente());
    return "index";
}

// métodos getters e setters das propriedades
}

```

Precisamos agora registrar a classe **ClienteBean** como um *bean* gerenciável pelo JSF. Para isso, vamos criar o arquivo de configuração *faces-config.xml* dentro da pasta *WEB-INF*. O conteúdo desse arquivo está descrito na **Listagem 7**.

Listagem 7. O arquivo de configuração do JSF *faces-config.xml*.

```

<?xml version="1.0" encoding="UTF-8"?>
<faces-config version="2.0" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-facesconfig_1_2.xsd">
    <managed-bean>
        <managed-bean-name>clienteBean</managed-bean-name>
        <managed-bean-class>br.com.javamagazine.ClienteBean</managed-bean-class>
        <managed-bean-scope>session</managed-bean-scope>
    </managed-bean>
</faces-config>

```

Como podemos notar, não há nenhuma novidade no conteúdo do *faces-config.xml*, exceto pela ausência da navegação entre as páginas, já que utilizamos a navegação implícita. Antes de falar sobre ela, vamos rodar a aplicação.

Ao executar o projeto que estamos desenvolvendo, a primeira página apresentada no navegador é a *index.xhtml*. Ela mostra a relação de clientes cadastrados no sistema. Como no início não há nenhum dado cadastrado, a lista de clientes estará vazia. Para criar um novo cliente, clique no botão **Cadastrar**. A **Figura 3** exibe a tela inicial.

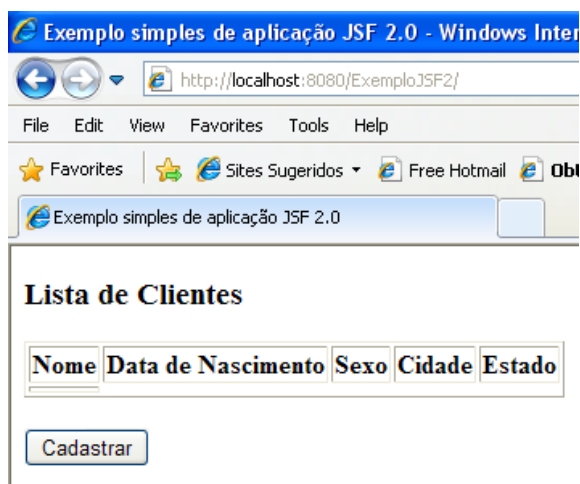


Figura 3. Página inicial da aplicação.

Na página de cadastro, preencha o formulário, como mostrado na **Figura 4**. Se clicarmos no botão *Cancelar*, a operação é cancelada e a primeira página é mostrada novamente. Clique no botão *Criar* para prosseguir.

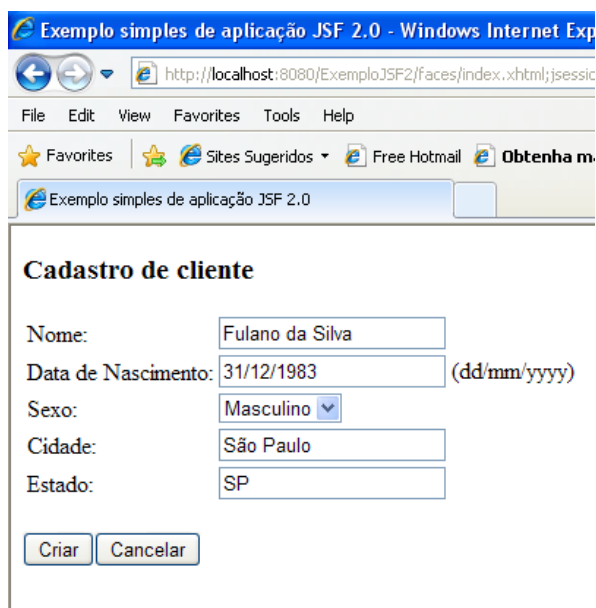


Figura 4. Página de cadastro de cliente.

A página mostrada a seguir é uma página de confirmação dos dados preenchidos, conforme a **Figura 5**. O *checkbox* *Novo cadastro?* não tem nenhuma função nesse momento. Ele será utilizado quando falarmos sobre navegação condicional. Clicando no botão *Confirmar Criação*, somos direcionados à página inicial, com o novo cliente cadastrado (**Figura 6**). Caso você tenha digitado alguma informação errada, na página de confirmação basta clicar no botão *Editar*. Assim a página de cadastro é exibida novamente, e você pode alterar as informações necessárias.

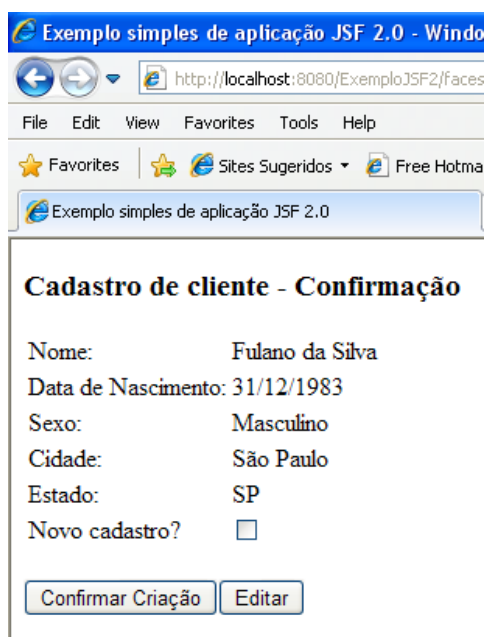


Figura 5. Página de confirmação de cadastro.

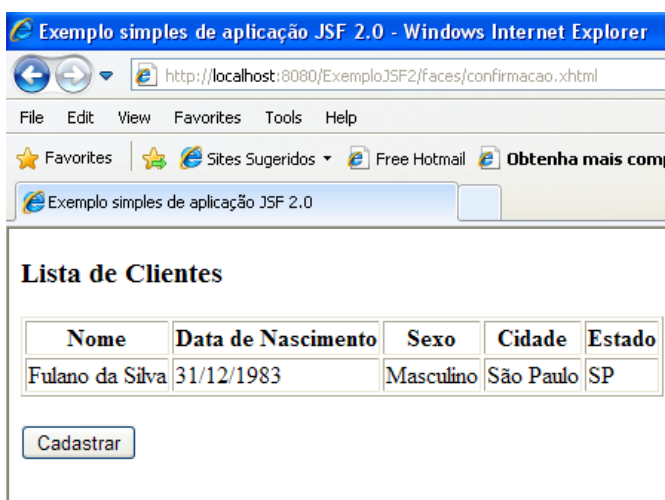


Figura 6. Página inicial com o novo cliente cadastrado.

A **Listagem 7** não mostra as regras de navegação entre as páginas porque a navegação foi definida nas páginas e no *bean* gerenciável. Na **Listagem 1**, podemos notar que a tag `<h:commandButton>` utiliza a **action cadastro** para navegar para a página *cadastro.xhtml*. Essa é a chamada **navegação implícita**. Caso a aplicação não possua o arquivo *faces-config.xml* (arquivo não necessário quando utiliza-se anotações para o registro dos *beans* gerenciáveis), ou nenhuma regra de navegação descrita nele, o valor do atributo **action** é inspecionado. Caso exista uma página que possua como nome base (nome da página sem a extensão) o valor desse atributo, a página em questão é carregada.

Na página *cadastro.xhtml*, a navegação para a página de confirmação (*confirmacao.xhtml*) é definida de maneira similar. Porém, para a navegação para a página inicial, a **action** da tag `<h:commandButton>` é uma chamada ao método **cancelar()** da classe **ClienteBean**. Esse método define de forma programática a navegação para a próxima página, que no caso é a página *index.xhtml* (valor de retorno “index”), como pode ser visto na **Listagem 6**.

Adicionar navegação em uma aplicação JSF 2.0 ficou bastante simples com o uso da navegação implícita. Como vimos, basta adicionar o nome da página de destino (sem a necessidade de colocar a extensão .xhtml) diretamente no código na página de origem, ou como resultado de um método do *bean* gerenciável. E o mais interessante: você não precisa colocar nenhuma regra de navegação no *faces-config.xml*.

Navegação condicional

Diferentemente da implícita, a navegação condicional é definida juntamente com as regras de navegação da aplicação, no arquivo *faces-config.xml*.

Conforme as versões anteriores do JSF, as regras de navegação do JSF 2.0 no *faces-config.xml* são definidas através das tags `<navigation-rule>` e `<navigation-case>`, sendo que dentro de um elemento `<navigation-rule>`, os itens `<navigation-case>` são processados na ordem em que aparecem. A primeira condição satisfeita fará com que a próxima página a ser mostrada no fluxo de navegação seja aquela definida na tag `<to-view-id>`. Para que um `<navigation-case>` seja satisfeito, basta que o resultado de um método do *bean* gerenciável, ou da página de origem, seja igual ao valor da tag `<from-outcome>`. A **navegação condicional** possibilita definir uma verificação a mais dentro de um `<navigation-case>`, com o uso da nova tag `<if>`.

No nosso projeto de exemplo, a partir da página *confirmacao.xhtml* pode-se chegar tanto na página *cadastro.xhtml* (quando você clica no botão *Editar*), como em *index.xhtml* (ao confirmar um novo cadastro de cliente). A fim de melhorar a usabilidade da aplicação, vamos deixar o usuário decidir se, após a confirmação de um novo cadastro, ele deseja voltar à tela inicial ou cadastrar um novo cliente, utilizando a navegação condicional. Para isso, no arquivo *faces-config.xml*, defina as regras de navegação a partir da página *confirmacao.xhtml* conforme a **Listagem 8**.

Listagem 8. O arquivo de configuração *faces-config.xml* com regras de navegação a partir de *confirmacao.xhtml*.

```

<?xml version="1.0" encoding="UTF-8"?>
<faces-config version="2.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-facesconfig_1_2.xsd">
  <managed-bean>
    <managed-bean-name>clienteBean</managed-bean-name>
    <managed-bean-class>br.com.javamagazine.ClienteBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>
  <navigation-rule>
    <from-view-id>/confirmacao.xhtml</from-view-id>
    <navigation-case>
      <from-outcome>index</from-outcome>
      <if>#{clienteBean.novoCadastro}</if>
      <to-view-id>/cadastro.xhtml</to-view-id>
    </navigation-case>
    <navigation-case>
      <from-outcome>index</from-outcome>
      <to-view-id>/index.xhtml</to-view-id>
    </navigation-case>
  </navigation-rule>
</faces-config>

```

Ao executar a aplicação novamente, podemos notar que para que um novo cadastro seja realizado logo após a confirmação de outro, basta selecionar o *checkbox* *Novo cadastro?* (observe a **Figura 5**). Na **Listagem 8**, no primeiro elemento **<navigation-case>**, a presença da tag **<if>** indica a navegação condicional. Essa tag verifica o valor do atributo **novoCadastro** no **clienteBean**. Se o valor do atributo for **true**, ou seja, o usuário selecionou o *checkbox*, então a próxima página a ser mostrada será *cadastro.xhtml*. Se essa regra falhar, o próximo elemento **<navigation-case>** é avaliado. Como não existe nenhuma verificação adicional no segundo **<navigation-case>**, a próxima página mostrada será *index.xhtml*.

Note que o conteúdo da tag **<if>** é uma Linguagem de Expressão (em inglês, *Expression Language* ou EL). Pode ser estranho o uso de EL no *faces-config.xml*, mas tanto na tag **<if>** quanto na tag **<to-view-id>** é possível utilizar este recurso. Você também pode incluir quantas tags **<if>** achar necessário dentro de um **<navigation-case>**. Nesse caso, a página definida em **<to-view-id>** somente será mostrada se todas as tags **<if>** forem satisfeitas.

Não foram definidas outras tags, além da tag <if>, para a navegação condicional.

Anotações em beans gerenciáveis

Ao registrarmos no JSF uma classe criada em nosso projeto web, ela se torna um *Managed Bean* (ou *bean* gerenciável). Quando isso acontece, esta classe fica gerenciável pelo *framework*, ou seja, a partir de qualquer página JSF é possível acessar seus métodos públicos. Daí o nome *Managed Bean*. No JSF, ele corresponde ao Modelo, da arquitetura *Model-View-Controller* (MVC), para os componentes de UI.

Na versão 1.2 do JSF, para que uma classe se tornasse um *bean* gerenciável, era necessário registrá-la através do *faces-config.xml*. Isso trazia um inconveniente, pois à medida que crescia o número de *beans* gerenciáveis, o arquivo de configuração também crescia na mesma proporção. E com um número grande desses *beans*, se tornava complicado gerenciar todas as alterações em três lugares distintos, porém relacionados: o *faces-config.xml*, a página JSF e o próprio *bean* gerenciável.

O JSF segue o padrão arquitetural Model-View-Controller (MVC), no qual o Modelo (Model) contém a lógica de negócios ou código que não seja de UI, a Visão (View) contém todo o código necessário para apresentar a UI ao usuário, e o Controlador (Controller) é o agente que trata as requisições do usuário e o redireciona para

a Visão apropriada. No JSF, o Modelo é representado pelos beans gerenciáveis, a Visão pelas páginas JSF e o Controlador pelo Faces Servlet.

No JSF 2.0 você não precisa mais registrar um *bean* no arquivo de configuração, como fizemos nas **Listagens 7 e 8**. Você pode agora utilizar anotações para isso. Dessa forma, o *bean* e o seu registro ficam no mesmo lugar, na mesma classe Java. Isso facilita bastante o controle dos *beans* gerenciáveis, como também deixa mais limpo o *faces-config.xml*. Porém, em alguns casos, você será obrigado a registrar os *beans* no arquivo de configuração, por falta de alternativa via anotações. Veremos um exemplo desta situação mais adiante.

Para cada anotação no JSF 2.0 há uma sintaxe XML correspondente a ser utilizada no faces-config.xml. É recomendável utilizar anotações ao invés do XML, pois com o uso de anotações o código torna-se mais coeso e a sua manutenção mais fácil.

Para exemplificar, vamos começar com uma configuração básica de um bean gerenciável. Nas **Listagens 7 e 8** vimos como a classe **ClienteBean** é registrada no JSF 1.2, no arquivo de configuração *faces-config.xml*. A **Listagem 9** mostra o registro da mesma classe no JSF 2.0, mas com o uso de anotações.

Listagem 9. Registro do bean gerenciável clienteBean no JSF 2.0 utilizando anotações.

```
package br.com.javamagazine;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

@ManagedBean
@SessionScoped
public class ClienteBean implements Serializable {
    private List<Cliente> listaCliente;
    private List<String> listaSexo;
    private Cliente cliente;
    private boolean novoCadastro;
    ...
}
```

Neste exemplo, a classe **br.com.javamagazine.ClienteBean** é registrada como um *bean* gerenciável (anotação **@ManagedBean**) de nome **clienteBean** e com escopo **session** (anotação **@SessionScoped**). Nas **Listagens 7 e 8**, o nome é definido pela tag **managed-bean-name**. Na **Listagem 9** o nome do *bean* não é configurado em nenhum lugar. Na verdade, ao utilizar anotações, o nome do *bean* é definido pelo atributo **name** da anotação **@ManagedBean**. Quando esse atributo é omitido, o *bean* terá o mesmo nome da classe (**ClienteBean**), mas com a primeira letra em minúsculo.

O escopo padrão do JSF 2.0 é o **request** (anotação **@RequestScoped**). Os *beans* gerenciáveis do tipo **request** podem ter a anotação de escopo omitida. Porém, para uma melhor legibilidade do código, é recomendável não omitir a anotação de escopo e nem o atributo **name** de **@ManagedBean**.

Caso você queira executar a aplicação novamente para validar as alterações realizadas na **Listagem 9**, edite o arquivo *faces-config.xml* removendo a tag **<managed-bean>** e seu conteúdo.

Escopos

No JSF 2.0 existem as seguintes anotações para definição de escopo de *beans* gerenciáveis:

1. **@NoneScoped**: os *beans* gerenciáveis de escopo **none** não são instanciados e nem salvos em nenhum escopo. Eles são instanciados, sob demanda, por outros *beans* gerenciáveis. Um *bean* gerenciável de escopo **none** somente pode instanciar outros *beans* gerenciáveis de escopo **none**;

2. **@RequestScoped**: os *beans* gerenciáveis de escopo **request** são instanciados e permanecem disponíveis durante uma mesma requisição HTTP. Eles podem instanciar outros *beans* gerenciáveis de escopo: **none**, **request**, **view**, **session** e **application**;
3. **@ViewScoped**: os *beans* gerenciáveis de escopo **view** permanecem disponíveis enquanto o usuário permanecer em uma mesma página de uma aplicação. Eles podem instanciar *beans* de escopo: **none**, **view**, **session** e **application**;
4. **@SessionScoped**: os *beans* gerenciáveis de escopo **session** são salvos na sessão HTTP de um usuário. Podem instanciar *beans* de escopo: **none**, **session** e **application**;
5. **@ApplicationScoped**: os *beans* gerenciáveis de escopo **application** permanecem disponíveis enquanto a aplicação estiver no ar, e podem ser acessados por todos os usuários da aplicação. Podem instanciar outros *beans* de escopo: **none** e **application**.
6. **@CustomScoped**: os *beans* gerenciáveis de escopo **custom** são *beans* que possuem um tempo de vida personalizado. Por exemplo, você pode definir um escopo de conversação, como existe no JBoss Seam, no qual um *bean* permanece disponível para um conjunto de páginas.

Como podemos notar, um *bean* gerenciável somente pode instanciar outro *bean* gerenciável de escopo **none** ou de escopo maior ou igual ao seu.

A anotação **@ManagedBean** possui, além do atributo **name**, o atributo **eager**, de tipo **boolean**, que somente é considerado quando o escopo do *bean* gerenciável for **application**. Caso o valor desse atributo seja **true** e o escopo **application**, o *bean* gerenciável será criado e colocado nesse escopo quando a aplicação iniciar, e não quando ele for referenciado pela primeira vez (quando **eager=false**). Ou seja, o *bean* gerenciável estará disponível antes que o *container* da aplicação comece a responder às requisições dos usuários. Para a fase de testes de uma aplicação, o uso do atributo **eager** pode ser bastante útil, pois o *bean* gerenciável pode realizar algumas tarefas antes de ser referenciado, como popular uma base de dados, por exemplo.

Exemplo de código:

```
@ManagedBean(name="clienteBean", eager=true)
@ApplicationScoped
public class ClienteBean ...
```

Se o atributo **eager** for utilizado e o escopo do *bean* não for **application**, então o *bean* gerenciável será criado quando for referenciado pela primeira vez. Esse é o comportamento padrão. O atributo **eager**, assim como o atributo **name**, é opcional, e seu valor default é **false**.

Inicializando propriedades do bean gerenciável

Podemos ter a necessidade de definir valores iniciais para as propriedades do *bean* quando o mesmo é criado. Deste modo, quando um *bean* for instanciado pelo ciclo de vida do JSF, suas propriedades serão inicializadas com valores pré-definidos, para serem utilizadas em um formulário, ou mesmo para usufruir dos benefícios da injeção de dependência, que torna o código mais desacoplado.

A **Listagem 10** mostra um exemplo de uso da anotação **@ManagedProperty**, que torna tudo isso possível. A **Listagem 11** mostra o mesmo exemplo, porém no arquivo de configuração.

Listagem 10. Inicializando a propriedade novoCadastro através de anotação.

```
package br.com.javamagazine;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import javax.faces.bean.*;

@ManagedBean(name="clienteBean")
@SessionScoped
public class ClienteBean implements Serializable {
    @ManagedProperty(value="true")
```

```
private boolean novoCadastro;
...
}
```

Listagem 11. Inicializando a propriedade novoCadastro através do *faces-config.xml*.

```
...
<managed-bean>
  <managed-bean-name>clienteBean</managed-bean-name>
  <managed-bean-class>br.com.javamagazine.ClienteBean</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
  <managed-property>
    <property-name>novoCadastro</property-name>
    <value>>true</value>
  </managed-property>
</managed-bean>
...
```

Nesse exemplo, quando o **clienteBean** for instanciado, a propriedade **novoCadastro** será inicializada com o valor **true**.

A anotação **@ManagedProperty** também possui um atributo opcional **name**. Esse atributo deve ter como valor o nome da propriedade que se deseja inicializar, por exemplo:

```
@ManagedProperty(name="novoCadastro", value="true")
private boolean novoCadastro;
```

Ao executar a aplicação novamente, considerando as alterações na **Listagem 10**, poderemos notar que o *checkbox Novo cadastro?* aparecerá previamente selecionado na página de confirmação de cadastro.

Utilizando injeção de dependência

O uso de injeção de dependência nos *beans* gerenciáveis simplifica o código e diminui o acoplamento entre os objetos, de modo que a classe se preocupe apenas com as regras de negócio da página em questão, já que as dependências estarão disponíveis quando necessárias.

A injeção de outros objetos no bean é estática, ou seja, somente ocorre durante a sua criação.

Para exemplificar, vamos registrar a classe **LocalidadeCliente** como um *bean* gerenciável, conforme a **Listagem 12**. A **Listagem 13** mostra o **clienteBean** referenciando este novo *bean*.

Listagem 12. Registro do *bean* gerenciável localidadeCliente no JSF 2.0 utilizando anotações.

```
package br.com.javamagazine;

import java.io.Serializable;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.NoneScoped;

@ManagedBean(name="localidadeClienteBean")
@NoneScoped
public class LocalidadeCliente implements Serializable {
  private String cidade;
  private String estado;
  // métodos getters e setters das propriedades
}
```

Listagem 13. Exemplo de uso de injeção de dependência.

```
package br.com.javamagazine;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import javax.faces.bean.*;
import javax.faces.context.FacesContext;

@ManagedBean(name="clienteBean")
```

```

@SessionScoped
public class ClienteBean implements Serializable {
    ...
    @ManagedProperty(value="#{localidadeClienteBean}")
    private LocalidadeCliente localidade;

    public ClienteBean() {
        listaCliente = new ArrayList<Cliente>();
        listaSexo = new ArrayList<String>();
        listaSexo.add("Masculino");
        listaSexo.add("Feminino");
        cliente = new Cliente();
    }

    public String criarCliente() {
        LocalidadeCliente localidadeCliente = new LocalidadeCliente();
        localidadeCliente.setCidade(localidade.getCidade());
        localidadeCliente.setEstado(localidade.getEstado());
        cliente.setLocalidadeCliente(localidadeCliente);
        listaCliente.add(cliente);
        cliente = new Cliente();
        return "index";
    }

    public String cancelar() {
        cliente = new Cliente();
        return "index";
    }
    // métodos getters e setters das propriedades
}

```

A injeção de dependência no *bean* gerenciável é feita através de EL no valor da anotação **@ManagedProperty**. Na **Listagem 13**, note que o valor da EL **#{localidadeClienteBean}** é o nome de registro do *bean* **LocalidadeCliente**, como pode ser visto na **Listagem 12**. Com a injeção de dependência, o **clienteBean** não precisa se preocupar em criar uma nova instância de **LocalidadeCliente** e gerenciá-la; basta utilizá-la quando necessário. Por isso, no construtor e nos métodos **criarCliente()** e **cancelar()** do **ClienteBean**, foi retirada a criação dessa nova instância.

No arquivo de configuração, o código correspondente às **Listagens 12** e **13** está descrito na **Listagem 14**. Na primeira *tag* **<managed-bean>**, a classe **br.com.javamagazine.LocalidadeCliente** é registrada com o nome **localidadeClienteBean** e escopo **none**. Na segunda *tag*, a injeção do *bean* **localidadeClienteBean** é feita na propriedade **localidade**, como mostrado na *tag* **<managed-property>**.

Listagem 14. Exemplo de uso de injeção de dependência através do *faces-config.xml*.

```

...
<managed-bean>
  <managed-bean-name>localidadeClienteBean</managed-bean-name>
  <managed-bean-class>br.com.javamagazine.LocalidadeCliente</managed-bean-class>
  <managed-bean-scope>none</managed-bean-scope>
</managed-bean>
<managed-bean>
  <managed-bean-name>clienteBean</managed-bean-name>
  <managed-bean-class>br.com.javamagazine.ClienteBean</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
  <managed-property>
    <property-name>localidade</property-name>
    <property-class>br.com.javamagazine.LocalidadeCliente</property-class>
    <value>#{localidadeClienteBean}</value>
  </managed-property>
</managed-bean>
...

```

Com as alterações realizadas nas **Listagens 12** e **13**, para acessar as propriedades **cidade** e **estado** de **localidadeClienteBean** nas páginas *cadastro.xhtml* e *confirmacao.xhtml*, basta utilizar as ELs **#{clienteBean.localidade.cidade}** e **#{clienteBean.localidade.estado}**, respectivamente.

Inicializando propriedades do tipo List ou Map do bean gerenciável

Nem sempre a propriedade que desejamos inicializar após a criação de um *bean* gerenciável é um simples objeto. Deste modo, o que fazer quando precisamos inicializar um conjunto de valores para

um *combobox*? Ou quando em um formulário o usuário pode escolher uma ou mais opções disponíveis através de *checkbox*? Também é possível inicializar propriedades do tipo **array**, **java.util.List** ou **java.util.Map** na criação de um *bean* gerenciável, mas não através de anotações. Propriedades desses tipos só podem ser inicializadas através do arquivo de configuração *faces-config.xml*.

Para exemplificar a inicialização de propriedades do tipo **List** e **Map**, vamos alterar o nosso código de modo que o usuário não precise mais digitar o estado ao preencher o formulário, mas sim selecioná-lo através de uma lista. E também, vamos criar uma propriedade nova, que indica em qual região (norte, sul, leste ou oeste) de uma determinada cidade o usuário reside. A **Listagem 15** mostra essas novas propriedades criadas na classe **LocalidadeCliente** e a **Listagem 16** mostra como inicializá-las. No caso da lista, foi utilizada a tag **<list-entries>**, na qual cada elemento é definido pela tag **<value>**. Como a propriedade **listaEstados** é uma lista de **String**, cada valor de **<value>** é do tipo **String**. Para inicializar um map, é utilizada a tag **<map-entries>**, que pode conter um ou mais **<map-entry>**. Por sua vez, cada **<map-entry>** possui uma tag **<key>** (chave do map) e **<value>** (valor correspondente).

Listagem 15. Exemplo de uso das propriedades de tipo *java.util.List* e *java.util.Map*.

```
package br.com.javamagazine;

import java.io.Serializable;
import java.util.List;
import java.util.Map;

public class LocalidadeCliente implements Serializable {
    private String cidade;
    private List<String> listaEstados;
    private Map<String, String> mapRegioes;

    // métodos getters e setters das propriedades
}
```

Listagem 16. Exemplo de uso de inicialização de propriedades do tipo *List* e *Map* no *faces-config.xml*.

```
...
<managed-bean>
  <managed-bean-name>localidadeClienteBean</managed-bean-name>
  <managed-bean-class>br.com.javamagazine.LocalidadeCliente</managed-bean-class>
  <managed-bean-scope>none</managed-bean-scope>
  <managed-property>
    <property-name>listaEstados</property-name>
    <property-class>java.util.List</property-class>
    <list-entries>
      <value>Acre</value>
      <value>Alagoas</value>
      ...
      <value>São Paulo</value>
      <value>Tocantins</value>
    </list-entries>
  </managed-property>
  <managed-property>
    <property-name>mapRegioes</property-name>
    <property-class>java.util.Map</property-class>
    <map-entries>
      <map-entry>
        <key>Norte</key>
        <value>Região Norte</value>
      </map-entry>
      <map-entry>
        <key>Sul</key>
        <value>Região Sul</value>
      </map-entry>
      <map-entry>
        <key>Leste</key>
        <value>Região Leste</value>
      </map-entry>
      <map-entry>
        <key>Oeste</key>
        <value>Região Oeste</value>
      </map-entry>
    </map-entries>
  </managed-property>
</managed-bean>
```

```

        </map-entry>
    </map-entries>
</managed-property>
</managed-bean>
...

```

Como podemos notar, o *bean* não possui mais anotações. O registro dele foi feito totalmente no arquivo de configuração. Quando um *bean* é registrado através do arquivo de configuração, qualquer anotação contida nele será desprezada pelo *framework*.

As anotações descritas nesse artigo são referentes às anotações do JSF 2.0 para beans gerenciáveis. No entanto, eles também podem utilizar anotações do Java EE, como @PostConstruct, @PreDestroy, @Resource, @EJB, entre outras. O uso dessas anotações está fora do escopo deste artigo. Outras anotações do JSF 2.0, como @FacesConverter e @FacesValidator também estão fora do escopo deste artigo, pois não são para utilização em beans gerenciáveis.

Conclusões

O JavaServer Faces é uma das tecnologias mais usadas para o desenvolvimento de aplicações web em Java, e sua versão 2.0 apresentou novidades e melhorias que o tornaram ainda mais robusto e de fácil utilização.

Neste artigo estudamos três destas novidades: as navegações implícita e condicional e as anotações nos *beans* gerenciáveis. Vimos quais são as *tags* correspondentes às anotações no arquivo de configuração *faces-config.xml*, e como ficou mais simples de se desenvolver uma aplicação com JSF em comparação com suas versões anteriores.

De que se trata o artigo:

Neste artigo veremos algumas das novidades do JSF 2.0, como o uso de anotações nos *beans* gerenciáveis (*Managed Beans*) e as novas formas de navegação: implícita e condicional.

Para que serve:

O artigo mostra como ficou mais fácil o desenvolvimento de aplicações JSF na versão 2.0, com a simplificação do modelo de navegação e sem a necessidade de configuração XML para registrar os *beans* gerenciáveis, graças às anotações.

Em que situação o tema útil:

Para quem deseja conhecer um pouco sobre a tecnologia JSF 2.0 e começar a desenvolver aplicações, o artigo mostra exemplos e diferenças da nova versão do JSF em relação às versões anteriores.

Anotações e Navegação no JSF 2.0:

A tecnologia JSF é uma das mais utilizadas para o desenvolvimento de aplicações web em Java. A versão 2.0 trouxe várias novidades que a tornaram ainda melhor. No entanto, como elas são muitas, o artigo se propõe a falar de três muito importantes: o uso de anotações em *beans* gerenciáveis e as navegações implícita e condicional.

Com o uso de anotações, o registro dos *beans* gerenciáveis não precisa mais ser feito através de configuração XML, o que facilita bastante a manutenção e o controle dos mesmos. E com as navegações implícita e condicional, o modelo de navegação do JSF fica mais poderoso e intuitivo, pois agora você tem a possibilidade de definir as regras de navegação nas próprias páginas, assim como estipular condições para se navegar de uma tela para outra.

Links

<http://mkblog.exadel.com/2009/08/learning-jsf2-managed-beans/>

Detalha o uso de anotações nos *managed beans* no JSF 2.0.

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Página para *download* do JDK 6 Update 21.

<http://netbeans.org/downloads/>

Página para *download* do NetBeans IDE 6.9.1 para Java.

Livros

JavaServer Faces 2.0: The Complete Reference, Ed Burns, C. Schalk, N. Griffin, McGraw-Hill, 2009

Livro de referência sobre JSF 2.0

Marcel Tokarski (marcel.tokarski@venturus.org.br) é graduado em Engenharia de Computação (Puccamp) e possui MBA em e-Business (FGV). Atua há mais de 8 anos com desenvolvimento de sistemas web e há 3 anos trabalha com JavaServer Faces. Atualmente está no Venturus – Centro de Inovação Tecnológica como arquiteto de sistemas web, desenvolvendo projetos para a conta Sony Ericsson.