

JavaServer Faces

JavaServer Faces (JSF) é uma especificação padronizada para a construção de Interfaces de Usuários para aplicações baseadas em Servidor. Antes do JSF, os desenvolvedores de aplicações web necessitavam criar páginas HTML através dos componentes Servlets ou Páginas JSP. Isto ocorria principalmente porque o HTML é o componente mais comum suportado por navegadores web. A principal implicação é que tal aplicação web tem uma interface de usuário simples, sem muitos recursos avançados se comparado com aplicações desktop.

Adicionalmente, se você trabalha no desenvolvimento de uma aplicação web, é possível que você se depare com muitos desafios técnicos, tais como a implementações de componentes para a construção de "queries" ou a construção de uma tabela para visualização de dados resultantes de uma consulta em um DataBase. A construção destes componentes necessita experiência e uma significativa quantidade de tempo para a construção bem como para o teste. Num ambiente ideal, os desenvolvedores poderiam utilizar rotinas prontas, testadas e facilmente configuráveis que se integrariam em suas aplicações.

JSF é uma tecnologia baseada em servidor que possibilita o desenvolvimento de aplicações web com interfaces de usuário rico em recursos, assemelhando a aplicações desktop. Com JSF, é possível a resolução de muitos desafios técnicos tais como a criação de componentes customizados para interface de usuário, de forma que o código da interface execute no servidor respondendo por eventos gerados no lado cliente (Navegador web).

A tecnologia JSF é baseada na arquitetura MVC (Model View Controller) para a separação entre lógica e a apresentação.

Este Documento oferece uma visão rápida para que você inicie a construção de aplicações Web com JSF.

1 Visão Geral de JavaServer Faces

JavaServer Faces (JSF) é a tecnologia que iniciou com a liderança da Sun Microsystems com o código JSR 127 sobre o Java Community Process (JCP). O objetivo é criar um framework padronizado para componentes de interface de usuário para aplicações web. Como mencionado anteriormente, JSF permite que seja construída uma aplicação web que execute num Servidor de aplicações Java e constrói a interface do usuário para o cliente. Esta tecnologia oferece um gerenciamento do ciclo de vida através de um servlet controlados. e um modelo de componentes com manipulação de eventos e componentes de renderização (construção de interface).

Para iniciar o uso de JSF, necessitamos efetuar o download da biblioteca que implementa a especificação ou nos certificarmos que o Servidor de Aplicações Java já dá o suporte a JSF.

1.1 A Arquitetura do JSF

A figura abaixo ilustra a arquitetura JSF, seus componentes e componentes auxiliares. É necessária uma implementação mínima de Servlet 2.3 e JSP 1.2 para suportar JSF 1.1.

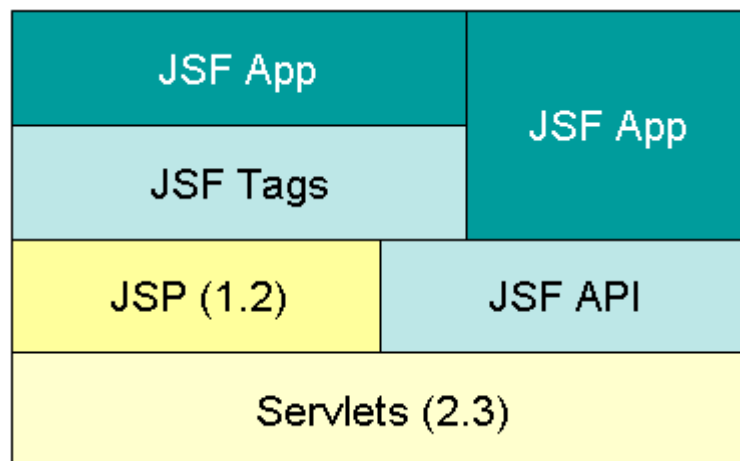


figura 1

A tecnologia JSF consiste de dois principais componentes:

1. Java APIs para representar componentes de Interface de Usuário, gerenciamento de estado, manipulação de eventos e validação de entrada de dados. A API também suporta internacionalização e acessibilidade.
2. Duas bibliotecas de Tags para representar a interface de usuário em páginas JSP, e para ligar estes aos componentes no lado servidor.

A figura abaixo demonstra o relacionamento entre o cliente, servidor, e JSF.

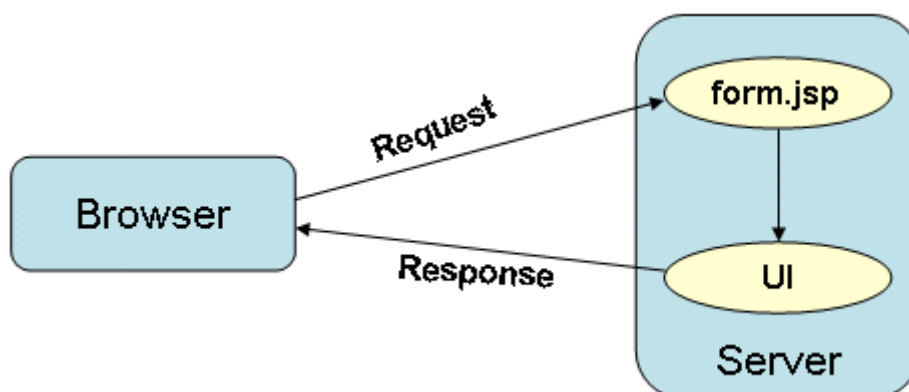


figura 2

Aqui, uma página JSP representa os componentes de interface de usuário através de Tags JSF, ao invés de utilizar marcadores HTML. O componente UI gerencia os objetos gerados para a página JSP.

Esta tecnologia abre o mercado para a construção de componentes de interface reutilizáveis. Desenvolvedores e Fabricantes de Software podem utilizar JSF como blocos de construção para a elaboração de "Custom Faces".

Uma das vantagens do JSF é que está baseada na arquitetura MCV, para oferecer uma clara separação entre a apresentação e a lógica. Isto parece familiar para aqueles que utilizam frameworks tais com o Struts. No entanto, note que JSF e Struts não são tecnologias que competem entre si, e sim, elas são interoperáveis. JSF, no entanto, tem algumas vantagens sobre Struts. Por exemplo, no Struts temos somente uma forma de criar um elemento (Renderizar), enquanto JSF oferece vários mecanismos. É opção do desenvolvedor a seleção do mecanismo, e este não necessita conhecer qual utilizar.

Este é o link para mais informações sobre a integração de JSF e Struts:

<http://jakarta.apache.org/struts/proposals/struts-faces.html>

1.2 Criando uma aplicação JSF

Uma aplicação JSF é como qualquer outra aplicação baseada em servidores de aplicação Java; ele é executado em um contêiner servlet, e contém:

1. Componentes JavaBeans (ou Objetos "camada Model") contendo dados e a funcionalidade específica da aplicação.
2. "Listeners" de eventos.
3. Páginas JSP.
4. Classes auxiliares baseadas no servidor.
5. Uma biblioteca de tags para construir (renderizar) os componentes de interface.
6. Uma biblioteca de tags representando os manipuladores de eventos e validadores.
7. Componentes de interface representados como objetos com estado ("Statefull") no servidor
8. Validadores, manipuladores de evento, e manipuladores de navegação.
9. Arquivo de configuração da aplicação.

Abaixo segue um exemplo utilizando as bibliotecas de tags. Para uma lista das tags oferecidas e outras informações sobre JSF utilize:

- JavaServer Faces specifications: <http://www.jcp.org/en/jsr/detail?id=127>
- e Capítulos 17–21 do Java EE tutorial:
<http://java.sun.com/javaee/5/docs/tutorial/doc/>

```

<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<body bgcolor="white">
<f:view>
<h:form id="helloform">
<h2>Qual é o seu nome?</h2>
    <h:inputText id="username" value="#{UserNameBean.userName}"
        validator="#{UserNameBean.validate}"/>
    <h:commandButton id="submit" action="success" value="Submit"/>
</h:form>
</f:view>

```

Os componentes da arquitetura JSF foram construídos de forma que estes mantêm seu estado. Um “renderkit” define como os componentes são mapeados para as tags apropriadas para um particular cliente. A implementação de referência JSF, inclui um “RenderKit” padrão para a construção (renderização) para um cliente HTML. Cada componente JSP no “RenderKit” html é composto pela funcionalidade definida pela classe “UIComponent”, e os atributos definidos pela classe “Renderer”. Por exemplo, as tags “commandButton” e “commandHyperLink” ambas representam um “UIComponent”, mas elas são construídas (renderizadas) de formas diferentes. O “commandButton” aparece como um botão e o “commandHyperLink” como um link.

Quando uma pagina JSP é criada utilizando componentes JSF, uma árvore de componentes ou uma “view” é construída na memória do servidor com cada tag correspondendo a um “UIComponent” na árvore. A árvore de componentes é utilizada pelo framework JSF para manipular as requisições da aplicação e criar as respostas. Quando um evento é gerado (por exemplo, clicks do usuário em um botão), o ciclo de vida JSF manipula o evento e gera a resposta apropriada. Note que o ponto de entrada no framework JSF é o “FacesServlet”. Ele atua como o controlador central e manipula as requisições processando o ciclo de vida JSF.

1.3 A implementação de referência JSF 1.2

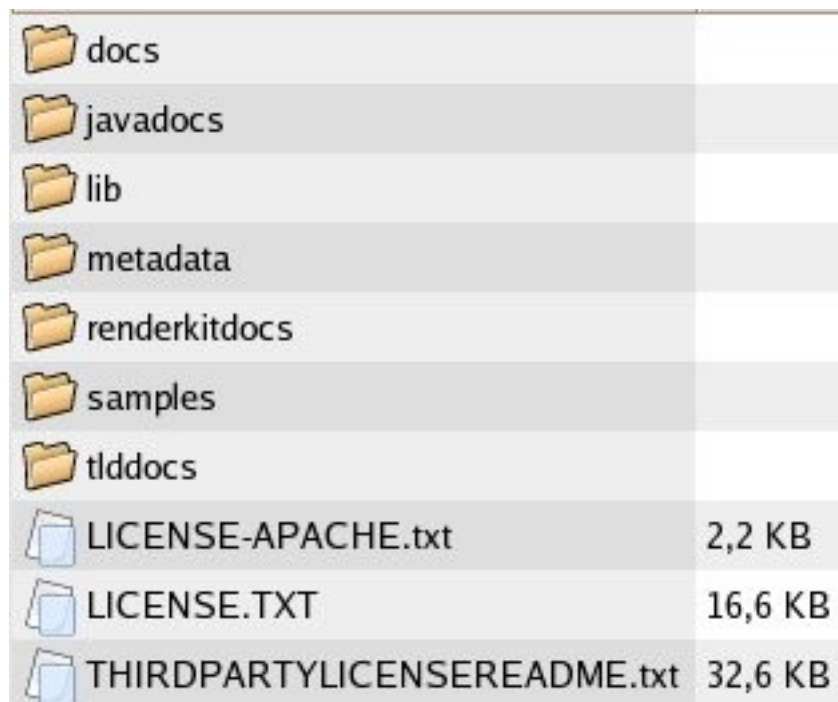
A implementação JSF vem com o seguinte:

- html_basic.tld: Uma biblioteca de tags para a construção de aplicações JSF que gera saída para clientes HTML.
- jsf_core.tld: uma biblioteca de tags para representar o núcleo das ações independente do “RenderKit”.
- APIs que oferecem componentes de interface de usuário, objetos gerenciadores, mecanismos de renderização, validação no lado server, conversão de dados, processamento de eventos, gerenciamento de fluxo de páginas, e extensibilidade para todos os componentes.
- Um conjunto de demonstrações localizadas no diretório “samples” da instalação.

Pode ser efetuado o download através do link:

<http://java.sun.com/javaee/javaserverfaces/download.html>

A implementação de referência é distribuída em um arquivo compactado com a seguinte estrutura:



docs	
javadocs	
lib	
metadata	
renderkitdocs	
samples	
tlddocs	
LICENSE-APACHE.txt	2,2 KB
LICENSE.TXT	16,6 KB
THIRDPARTYLICENSEREADME.txt	32,6 KB

figura 3

O diretório “samples” contém vários arquivos “war” (Web Archives) representando os exemplos. O diretório “lib” contém todas as bibliotecas necessárias para as aplicações:

- jsf-api.jar: Contém as classes da API “javax.faces.*”.
- jsf-impl.jar: Contém as classes que implementam a especificação JSF.

2 Entendendo uma aplicação JavaServer Faces

Uma aplicação JSF consiste dos seguintes arquivos:

- Arquivos JSP com componentes JSF representando a interface do usuário.
- JavaBeans que definem o modelo de dados.
- Arquivos de configuração da aplicação e especificação do controlador servlet, beans gerenciáveis e manipuladores de navegação.

2.1 As páginas JSP

Tomando como exemplo a aplicação “guessNumber” que vem junto com a implementação de referência temos:

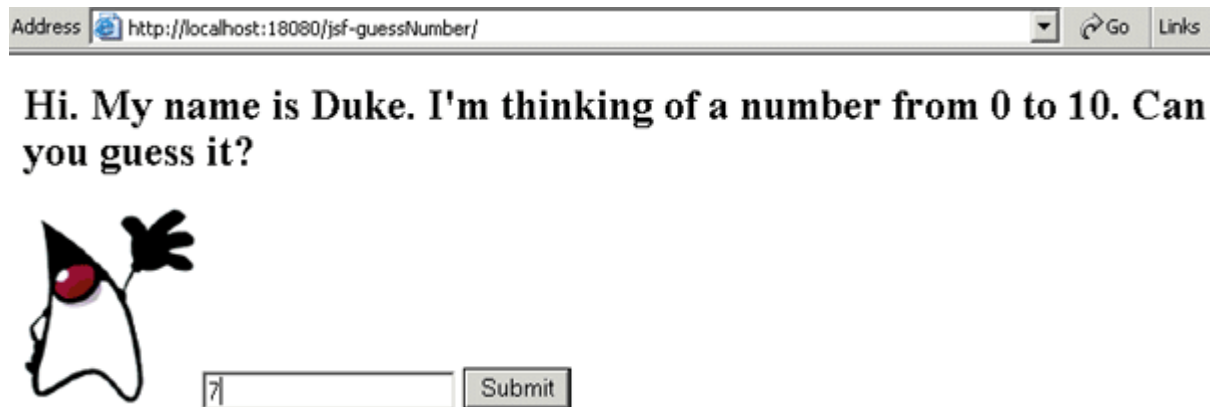


figura 4 – página inicial da aplicação guessNumber

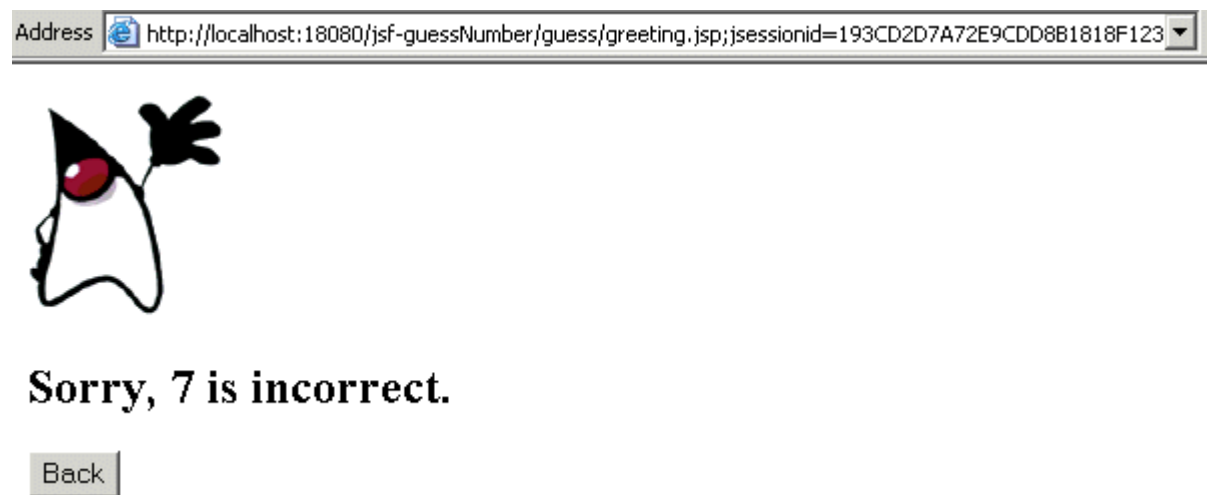


figura 5 – Aplicação guessNumber: Tela de erro



figura 6 – Aplicação guessNumber: Tela sucesso

Esta aplicação usa as seguintes páginas JSP: index.jsp, greeting.jsp, e response.jsp.

Código exemplo: index.jsp

```
<html>
<head>
</head>
<body>
  <jsp:forward page="guess/greeting.jsp" />
</body>
</html>
```

Como você pode ver, esta página simplesmente redireciona o usuário para a página principal “greeting.jsp”, esta é a forma para entrar numa aplicação JSF.

A página principal é apresentada na figura 4.

Código exemplo: greeting.jsp

```
<html>
<head><title>Hello</title></head>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<body bgcolor="white">
<f:view>
  <h:form id="helloForm" >
    <h2>Hi. My name is Duke. I'm thinking of a number from
    <h:outputText value="#{UserNumberBean.minimum}"/> to
    <h:outputText value="#{UserNumberBean.maximum}"/>.
      Can you guess it?</h2>
    <h:graphicImage id="waveImg" url="/wave.med.gif" />
    <h:inputText id="userNo" value="#{UserNumberBean.userNumber}"
      validator="#{UserNumberBean.validate}"/>
    <h:commandButton id="submit" action="success"
      value="Submit" />
    <p>
    <h:message style="color: red;
      font-family: 'New Century Schoolbook',
      serif; font-style: oblique; text-decoration: overline"
      id="errors1" for="userNo"/>
    </h:form>
  </f:view>
</body>
</html>
```

Esta página demonstra vários recursos que podem ser utilizados em muitas aplicações:

- De forma a utilizar as tags JSF, é necessário incluir a declaração “taglib” para as tags “html” e “core” que utilizam o mecanismo de criação de HTML e JSF respectivamente.
- Uma página contendo tags JSF é representada por uma árvore de componentes a qual o “root” é o “UIViewRoot”, que é representada pela tag “view”. Todos os componentes JSF devem ser incluídos nesta tag. Outros conteúdos, tais como HTML e JSP também podem ser incluídos nesta tag.
- Uma página típica JSP inclui um formulário, ao qual é submetido quando um botão é clicado. As tags representando os componentes caixas de texto, botões, etc. Devem ser incluídas dentro da tag “form”.

- A tag “outputText” representa um “label”. Esta página contém duas destas tags que mostram os números 0 e 10. Os atributos “value” das tags obtém os valores das propriedades “minimum” e “maximum” da classe bean, “UserNumberBean”, utilizando expressões de ligação para valores com a sintaxe “#{bean-managed-property}”
- A tag “graphicImage” é utilizada para referenciar uma imagem.
- A tag “inputText” representa um campo de entrada de dados (em HTML <input type=“text”/>). O atributo “id” representa o ID do componente representado por esta tag, e se não for informada, será gerado automaticamente. O atributo “validator” faz referência a uma expressão de ligação que aponta para o método de um bean que efetua a validação da informação do componente.
- A tag “commandButton” representa um botão (em HTML <input type=“submit” />). O atributo “action” auxilia no mecanismo de navegação para decidir qual será a próxima página a abrir.
- A tag “message” mostra mensagens de erro se o valor informado não for válido. O atributo “for” faz referência ao componente que gerou o erro de validação.

A página response.jsp é representada nas figuras 5 e 6, e é apresentada como resposta à requisição do usuário quando clica no botão “Submit”.

Código exemplo: response.jsp

```
<html>
<head> <title>Guess The Number</title></head>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<body bgcolor="white">
<f:view>
  <h:form id="responseForm" >
    <h:graphicImage id="waveImg" url="/wave.med.gif" />
    <h2><h:outputText id="result"
      value="#{UserNumberBean.response}" /></h2>
    <h:commandButton id="back" value="Back" action="success"/><p>
  </h:form>
</f:view>
</body>
</html>
```

2.2 O Modelo

Uma aplicação típica JSF utiliza um bean em cada página da aplicação. O Bean define as propriedades e métodos associados com os componentes da interface de usuário usado na página. Um bean pode também definir um conjunto de métodos que executam funções, tais como validação de dados para os componentes. O Bean tem um conjunto de métodos set’s e get’s. O código a seguir mostra um JavaBean, que é utilizado na página “greeting.jsp” e “response.jsp”.

Código exemplo: UserNumberBean.java

```
package guessNumber;

import javax.faces.component.UIComponent;
```



```

import javax.faces.context.FacesContext;
import javax.faces.validator.LongRangeValidator;
import javax.faces.validator.ValidatorException;
import java.util.Random;

public class UserNumberBean {
    Integer userNumber = null;
    Integer randomInt = null;
    String response = null;
    protected String[] status = null;
    private int maximum = 0;
    private boolean maximumSet = false;
    private int minimum = 0;
    private boolean minimumSet = false;

    public UserNumberBean() {
        Random randomGR = new Random();
        do {
            randomInt = new Integer(randomGR.nextInt(10));
        } while(randomInt.intValue() == 0);
        System.out.println("Duke's number: " + randomInt);
    }

    public void setUserNumber(Integer user_number) {
        userNumber = user_number;
        System.out.println("Set userNumber " + userNumber);
    }

    public Integer getUserNumber() {
        System.out.println("get userNumber " + userNumber);
        return userNumber;
    }

    public String getResponse() {
        if(userNumber != null && userNumber.compareTo(randomInt) == 0) {
            return "Yay! You got it!";
        } else if(userNumber == null) {
            return "Sorry, " + userNumber + " is incorrect. Try a larger number.";
        } else {
            int num = userNumber.intValue();
            if(num > randomInt.intValue()) {
                return "Sorry, " + userNumber + " is incorrect. Try a smaller number.";
            } else {
                return "Sorry, " + userNumber + " is incorrect. Try a larger number.";
            }
        }
    }

    public String[] getStatus() {
        return status;
    }

    public void setStatus(String[] newStatus) {
        status = newStatus;
    }

    public int getMaximum() {
        return (this.maximum);
    }

    public void setMaximum(int maximum) {
        this.maximum = maximum;
        this.maximumSet = true;
    }
}

```

```

public int getMinimum() {
    return (this.minimum);
}

public void setMinimum(int minimum) {
    this.minimum = minimum;
    this.minimumSet = true;
}

public void validate(FacesContext context, UIComponent component, Object
value)
    throws ValidatorException {
    if((context == null) || (component == null)) {
        throw new NullPointerException();
    }
    if(value != null) {
        try {
            int converted = intValue(value);
            if(maximumSet && (converted > maximum)) {
                if(minimumSet) {
                    throw new ValidatorException(MessageFactory.getMessage(context,
                        LongRangeValidator.NOT_IN_RANGE_MESSAGE_ID, new Object[] {
                            new Integer(minimum), new Integer(maximum),
                            MessageFactory.getLabel(context, component)
                        }));
                }
            } else {
                throw new ValidatorException(MessageFactory.getMessage(context,
                    LongRangeValidator.MAXIMUM_MESSAGE_ID, new Object[] {
                        new Integer(maximum),
                        MessageFactory.getLabel(context, component)
                    }));
            }
        }
        if(minimumSet && (converted < minimum)) {
            if(maximumSet) {
                throw new ValidatorException(MessageFactory.getMessage(context,
                    LongRangeValidator.NOT_IN_RANGE_MESSAGE_ID, new Object[] {
                        new Double(minimum), new Double(maximum),
                        MessageFactory.getLabel(context, component)
                    }));
            }
        } else {
            throw new ValidatorException(MessageFactory.getMessage(context,
                LongRangeValidator.MINIMUM_MESSAGE_ID, new Object[] {
                    new Integer(minimum),
                    MessageFactory.getLabel(context, component)
                }));
        }
    }
} catch(NumberFormatException e) {
    throw new ValidatorException(MessageFactory.getMessage(context,
        LongRangeValidator.TYPE_MESSAGE_ID, new Object[] {
            MessageFactory.getLabel(context, component)
        }));
}
}

private int intValue(Object attributeValue) throws NumberFormatException {
    if(attributeValue instanceof Number) {
        return (((Number)attributeValue).intValue());
    } else {

```

```

    }
    return (Integer.parseInt(attributeValue.toString()));
}
}
}

```

2.3 O Arquivo de configuração da aplicação

Um arquivo de configuração de aplicação, `faces-config.xml`, é utilizado para definir os beans, validadores, conversores, e regras de navegação. O Código a seguir mostra este arquivo para a aplicação `guessNumber`.

Código exemplo: `faces-config.xml`

```

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE faces-config PUBLIC
    "-//Sun Microsystems, Inc.//DTD JSF Config 1.1//EN"
    "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">
<faces-config>
    <application>
        <locale-config>
            <default-locale>en</default-locale>
            <supported-locale>de</supported-locale>
            <supported-locale>fr</supported-locale>
            <supported-locale>es</supported-locale>
        </locale-config>
    </application>
    <navigation-rule>
        <description>
            The decision rule used by the NavigationHandler to
            determine which view must be displayed after the
            current view, greeting.jsp is processed.
        </description>
        <from-view-id>/greeting.jsp</from-view-id>
        <navigation-case>
            <description>
                Indicates to the NavigationHandler that the response.jsp
                view must be displayed if the Action referenced by a
                UICommand component on the greeting.jsp view returns
                the outcome "success".
            </description>
            <from-outcome>success</from-outcome>
            <to-view-id>/response.jsp</to-view-id>
        </navigation-case>
    </navigation-rule>
    <navigation-rule>
        <description>
            The decision rules used by the NavigationHandler to
            determine which view must be displayed after the
            current view, response.jsp is processed.
        </description>
        <from-view-id>/response.jsp</from-view-id>
        <navigation-case>
            <description>
                Indicates to the NavigationHandler that the greeting.jsp
                view must be displayed if the Action referenced by a
                UICommand component on the response.jsp view returns
                the outcome "success".
            </description>
            <from-outcome>success</from-outcome>
            <to-view-id>/greeting.jsp</to-view-id>
        </navigation-case>
    </navigation-rule>

```

```

<managed-bean>
  <description>
    The "backing file" bean that backs up the guessNumber webapp
  </description>
  <managed-bean-name>UserNumberBean</managed-bean-name>
  <managed-bean-class>guessNumber.UserNumberBean</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
  <managed-property>
    <property-name>minimum</property-name>
    <property-class>int</property-class>
    <value>0</value>
  </managed-property>
  <managed-property>
    <property-name>maximum</property-name>
    <property-class>int</property-class>
    <value>10</value>
  </managed-property>
</managed-bean>
</faces-config>

```

A tarefa de definir as regras de validação envolve a definição de quais páginas serão mostradas após a ação do usuário. Cada elemento “<navigarion-rule>” define como ir de uma página, como definido em “<from-view-id>” para as outras páginas da aplicação. Um elemento “<navigarion-rule>” contém um número indeterminado de elementos “<navigation-case>” que definem as páginas a serem utilizadas com “<to-view-id>” baseadas numa condição lógica definida por “<from-outcome>”. Este “outcome” é definido pelo atributo “action” do componente que efetua o “submit” no formulário (tais como “commandButton” no código dos exemplos acima).

Adicionalmente, os beans necessitam serem configurados neste arquivo assim a implementação pode criar automaticamente novas instâncias deste beans quando necessário. O elemento “<managed-bean>” é usado para criar o mapeamento entre um bean e uma classe. A primeira vez que o bean “UserNumberBean” é referenciado, um objeto é criado e armazenado num escopo apropriado (sessão, aplicação ou requisição).

Finalmente, de forma a utilizar o framework JSF em suas aplicações web, você necessita declarar o “FaceServlet” e seu mapeamento no arquivo de publicação (deployment descriptor file) “web.xml”. Este servlet atua como o controlador central e manipula todas as requisições relacionadas com JSF. O Código de exemplo a seguir mostra este o arquivo “web.xml” para a aplicação “guessNumber”. Uma coisa importante a ser notada é o parâmetro “javax.faces.STATE_SAVING_METHOD”, que é utilizado para especificar onde o estado deve ser salvo (o cliente neste caso). Se você quer salvar o estado no servidor (este é o valor default na implementação de referência JSF), então especifique “server” ao invés de “client” no “param-value”. Note que se o estado é salvo no cliente, o estado da página inteira é embutida num campo escondido (hidden).

Código exemplo: web.xml

```

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE web-app PUBLIC
  "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <display-name>JSF Guess Number Sample Application</display-name>

```

```

<description>
    JSF Guess Number Sample Application
</description>
<context-param>
    <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
    <param-value>client</param-value>
</context-param>
<context-param>
    <param-name>com.sun.faces.validateXml</param-name>
    <param-value>true</param-value>
    <description>
        Set this flag to true if you want the JSF
        Reference Implementation to validate the XML in your
        faces-config.xml resources against the DTD. Default
        value is false.
    </description>
</context-param>
<context-param>
    <param-name>com.sun.faces.verifyObjects</param-name>
    <param-value>true</param-value>
    <description>
        Set this flag to true if you want the JSF
        Reference Implementation to verify that all of the application
        objects you have configured (components, converters,
        renderers, and validators) can be successfully created.
        Default value is false.
    </description>
</context-param>
<!-- Faces Servlet -->
<servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup> 1 </load-on-startup>
</servlet>
<!-- Faces Servlet Mapping -->
<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>/guess/*</url-pattern>
</servlet-mapping>
<security-constraint>
    <!-- This security constraint illustrates how JSP pages
        with JSF components can be protected from
        being accessed without going through the Faces Servlet.
        The security constraint ensures that the Faces Servlet will
        be used or the pages will not be processed. -->
    <display-name>Restrict access to JSP pages</display-name>
    <web-resource-collection>
        <web-resource-name>
            Restrict access to JSP pages
        </web-resource-name>
        <url-pattern>/greeting.jsp</url-pattern>
        <url-pattern>/response.jsp</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <description>
            With no roles defined, no access granted
        </description>
    </auth-constraint>
</security-constraint>
</web-app>

```

Bibliografia:

- Developing Web Applications with JavaServer Faces
<http://java.sun.com/developer/technicalArticles/GUI/javaServerFaces/>

Prof.: Wilson J. Santana – 2007/07