Fundamentals of Java Enterprise Components

Session: 8

JavaServer Faces as Web Pages

Objectives

- Develop Web pages using JSF
- Add various components to the Web pages through JSF tags
- Describe the use of converters, validators, and listeners to add additional functionality to the Web pages
- Develop server-side components such as managed beans and converters for JSF
- Explain how to connect all the components together to develop a comprehensive application

Using JavaServer Faces Technology in Web Pages

- JSF provides component tags to be used to create the Web page of an application.
- The component tags can be made functional by registering converters, validators, and listeners with the components.
- Managed beans are associated with the components in the Web page to implement the business logic of the application.

Web Page Setup Using JSF

A typical JSF Web page comprises the following components:

- Namespace declarations which are used to declare JSF tag libraries.
- HTML head and body tags. However, these are optional.
- Form tag which represents the user input components.

The tag libraries which will be used in the current JSF page have to be declared.

Following are the essential tag libraries whose access is required for all JSF pages:

- JSF HTML render kit
- JSF Core tag library
- HTML standard tag library

When multiple tag libraries are included for a certain Web page, then a tag library specific prefix has to be added to the tag.

HTML Tag Libraries in JSF Pages

- The HTML tag library in JSF is used to add HTML components to the Web page.
- Based on the client who is invoking the Web page, each component of the JSF can be rendered in a different way.
- Following table lists some of the important HTML tags used in JSF pages:

Tag	Description
`h:form`	Represents an input form to which other components can be added
`h:datatable`	Creates an HTML table which can be modified dynamically
`h:graphicImage`	Displays an image
`h:column`	Represents a column of data in a data table of the database
`h:commandLink`	Used to create a hyperlink
`h:inputText`	Used as text area which can accept user input
`h:message`	Used to display a localized message
`h.outputText`	Allows display of plain text
`h:selectOneListBox`	Allows selecting one element from a list of options
`h:selectOneMenu`	Allows selecting one item from a menu
`h.selectRadioOne`	Allows choosing one option from a list of options

Component Tag Attributes

Each component has certain attributes associated with it, following is the interpretation of these attributes:

- **binding** – This property binds the component with a bean, which will be invoked for any action with respect to the component.
- **id** – Uniquely identifies the component. It is used when the component has to be referred by another component.
- **immediate** – This attribute can only assume a value of true, it implies that the converters and validators associated with the component should act immediately.
- **rendered** - This attribute specifies the condition when the component should be rendered. This also assumes a value true, which implies that the component has to be rendered.
- **style** – specifies a CSS style for the component.
- **styleClass** – this attribute specifies a CSS class according to which the component has to be rendered.
- **value** – This specifies the value of the component in the form of a value expression.

Using Text Component Tags 1-2

Text components of a JSF page are essential to view and display text on the Web page.

There are three types of text components as follows:

- **Label** - Label is used to display read only text on the Web page.
- **Field** - Field allows the user to enter text data, which can be submitted along with the user form.
- **Password** - allows the user to enter data, but the data entered in the field is concealed through characters such as asterisks.

There are four types of input tags as follows:

- **h:inputHidden** - allows accepting a hidden variable in the Web page.
- **h:inputSecret** - used for text areas which accept password or any confidential values.
- **h:inputText** - used to accept text input in a single line.
- **h:inputTextArea** - used to accept input of multiple lines.

Using Text Component Tags 2-2

Output tags are used to include the output components in the JSF Web page. There are four types of output components:

- **h:outputFormat** - used to display formatted output
- **h:outputLabel** - used to display a read-only label
- **h:outputLink** - used to display a hyperlink
- **h:outputText** - used to display a one-line text string

Using Command Component Tags

The command component tags are used to perform actions and navigation.

- **h:commandButton** - used to add a button component.
- **h:commandLink** - adds a hyperlink

Apart from the default set of attributes to be added to the components, an action and actionListener attribute can also be added to the command component tags.

The action attribute can present a logical outcome or a reference to a bean which will result in a logical outcome.

The actionListener attribute refers to the bean method or listener which will process the component action.

Using Data Bound Table Components

The h:datatable component is used to add relational data to the Web page. This component enables binding to a collection of data objects.

The value attribute of the h:datatable tag is used to bind relational data to the component.

The data object which can be added to the data table component can be:

- Single bean
- list of beans
- Array of beans
- javax.faces.model.DataModel object
- java.sql.Resultset object
- javax.servlet.jsp.jstl.sql.Result object
- javax.sql.Rowset object

Using Core Tags

JSF has a set of core tags used to perform core actions.

Following are the categories of core tags supported by JSF:

- Event handling core tags – actionListener, phaseListener, setPropertyActionListener, valueChangeListener are the event listening core tags
- Data Conversion core tags – converter, converterDateTime, convertNumber are data conversion core tags
- Facet Core tags - Facet represents a named section within a container, facet is created through facet tag
- Core tags used to represent items in a List – f:selectItem and f:selectItems tags are used to represent items in a list
- Validator core tags – These tags are used to validate input values or other values in JSF pages
- Miscellaneous core tags

All these core tags are inserted in the page definition with a prefix ‘f’.

Using Converters, Listeners, and Validators

Converters, Listeners, and Validators are also components of JSF pages which process the input data of the form.

Converters are used to convert the input data of the components.

Listeners capture the events occurring in the page and perform actions according to the events.

Validators are used to validate the data received from the input components.

Using Standard Converters 1-2

- The JSF standard implementation provides a set of Converter implementations that can be used to convert component data.
- The standard converter implementations present in javax.faces.Convert can be used to perform required conversions between the model view and presentation view.
- Following are the converter implementations present in javax.faces.Convert package:

BigDecimalConverter	EnumConverter
BigIntegerConverter	FloatConverter
BooleanConverter	IntegerConverter
ByteConverter	LongConverter
CharacterConverter	NumberConverter
DateTimeConverter	ShortConverter
DoubleConverter	

Using Standard Converters 2-2

In order to use a converter with a component, the converter has to be registered with the component first. Following are the ways in which the converter can be registered with a component:

- The converter tag is nested within the component tag. This is generally used for date and time converters.
- Bind the value attribute of the component with the managed bean property which in turn has the converter.
- Add a converter attribute to the component tag and refer to the converter from the converter attribute.
- A converter tag can be nested in a component tag. Then, one can use either a converter tag's converterId attribute or its binding attribute to reference the converter.

Using DateTimeConverter

The DateTimeConverter is used to convert the input text of a component into java.util.Date type. This conversion is possible by including a convertDateTime tag in the component tag.

The DateTimeConverter can have the following attributes:

- **binding** – It is used to bind the converter to a bean.
- **dateStyle** – This attribute defines the date format according to java.text.DateFormat.
- **for** – This attribute is used in case of composite components, in this case to bind the dateTimeConverter to certain component in the group.
- **locale** – This attribute is used to represent a date format which is part of predefined set of date styles.
- **pattern** – This attribute determines the formatting pattern of the date and time.
- **timeStyle** – This attribute is also used to define the format of the date and time.
- **timeZone** – This attribute of the DateTimeConverter defines the time zone in which the application is running.
- **type** – This attribute defines whether a string value has date, time, or both date and time.

Using NumberConverter

- Numberconverter tag is used to transform the data from the input component to java.lang.Number type object.
- NumberConverter also has several attributes which define the parameters of the conversion.
- Few of these attributes are binding, currencyCode, currencySymbol, for, groupingUsed, and so on.

Registering Listeners on Components

- Listener objects are used to capture events in the Web page.
- Listeners can be implemented as classes or as managed bean methods.
- The listener method is referenced through actionListener or valueChangeListener if it is implemented as a managed bean method or as a class.

Registering a ValueChangeListener on a Component

On a component which implements EditableValueHolder, the valueChangeListener can be registered by nesting a f:valueChangeListener tag within the component's tag.

The **ValueChangeListener** has two attributes namely, type and binding.

- type attribute specifies the implementation of **ValueChangeListener** along with the tag,
- binding attribute binds the component to a managed bean.

Following code displays the usage of type attribute:

```
<h:inputText id="name"
size="30"
value="#{ShoppingBean.name}"
required="true">
<f:valueChangeListener
type="Dealsstore.listeners.Us
erNameChanged" />
</h:inputText>
```

Registering an ActionListener on a Component

Can be registered on command components such as buttons by including a tag in the command component.

The actionListener also has type and binding attributes which are used to reference the user defined listener implementations.

The core tag library also has an setActionListener tag which can be used to register an actionListener with an ActionSource instance.

Using Standard Validators

JSF provides a standard set of validators. Following are the predefined set of validators:

- **BeanValidator class** – used to register a bean validator for the component.
- **DoubleRangeValidator class** – class used to validate whether a certain float value of a component is within the specified range or not.
- **LengthValidator class** – This class is used to validate string inputs based on the length of the string.
- **LongRangeValidator** – This class is used to define validators which will check the range of floating type value.
- **Regex Validator** - This class is used to define a validator of an input component against a regular expression from the java.util.regex package.
- **RequiredValidator** – This class is used to define a validator which checks that the current value is not empty. This is applied on a component which accepts input.

Allows for user defined validators also by implementing the Validator interface.

The validations of the input data can also happen through bean validation.

Validating a Component's Value

A validator can be applied to a component through any one of the following three ways:

- Nest the validator tag within the corresponding component tag.
- Refer to a method which performs the validation as an attribute to the component tag.
- Nest the validator tag within the component tag and use the validatorId or binding attribute to refer to the validator.

Validation can be performed only on those components which implement EditableValueHolder.

Following is an example of using a validator tag:

```
<h:inputText id="check"
size="4"
value="#{cartItems.quantity}"
>
<f:validateLongRange
minimum="1"/>
</h:inputText>
```

Referencing a Managed Bean Method

- Components can be associated with managed beans through attributes. Following are attributes which enable referencing managed bean methods:
  - action
  - actionListener
  - validator
  - valueChangeListener
- Components implementing ActionSource interface can use the action and actionListener attributes.
- Components implementing EditableValueHolder interface can use the validator and valueChangeListener attributes.

Developing Applications with JSF Technology

Following are the steps involved in developing an application using JSF technology for implementing the converters, validators, and so on.

- Creating a managed bean
- Writing bean properties
- Writing managed bean methods

Creating a Managed Bean

Managed bean implements the business logic of the application

The managed bean properties can be bound to anyone of the following:

- Component value
- Component instance
- Converted value
- Listener instance
- Validator instance

A managed bean can handle the following functions in an application:

- validate component's data
- handle an event fired by a component
- define page navigation

Using EL to Reference Managed Beans

Expression language can be used with the component tags to reference managed beans.

Given code demonstrates how the values entered in the Web page are associated with the beans.

Apart from bean properties, these attribute expressions can also refer to implicit objects, lists, arrays, and maps.

```
<h:inputText id="payamount" size="30"
value="#{ShoppingBean.price}"
required="true">
converter = "#shoppingBean.IntegerConverter"
validator="#{shoppingBean.validateNumberRange}"
</h:inputText>
```

Writing Bean Properties

A component tag value can be bound with the managed bean property through the value attribute.

The component instance can be bound to the managed bean through binding attribute.

The 'binding' attribute of all the converters, listeners, and validators can be used to bind them to the managed bean.

Writing Properties Bound to Component Values

When the developer intends to write the component value as a bean property, then the data type of the bean property and the component value must be same.

Following are the component classes supported by javax.faces.components and acceptable types of their values:

- UIInput, UIOutput, UISelectItem, and UISelectOne for primitive data types in Java
- UIData for group of data
- UISelectBoolean
- UISelectItem
- UISelectMany

Writing Properties Bound to Component Instances

- When a property is bound to component instance, it will return a component instance instead of values.
- Following provides an example of binding a property with a component instance:

```
<h:selectBooleanCheckbox id="payment"
binding="#{payBean.paymentGateway}" />
<h:outputLabel for="payment" rendered="false"
binding="#{payBean.cardDetails}" />
</h:outputLabel>
```

- The converter, listener, and validator implementations are bound to managed bean attributes, which enables the managed bean to manipulate their implementations.

Writing Managed Bean Methods 1-2

Managed bean methods perform several application specific functions such as processes associated with the page navigation, handling action events, performing validation on the component values, and handling value change events.

Writing method to handle navigation

- An action method is bound to the action attribute of the component tag, determining the page to be displayed by the navigation system.

Writing a method to handle an action event

- A method that handles an action event should be public in the managed bean.
- This method will accept the action event as a parameter and return void.
- The action method handling an action event is referenced by the component actionListener **attribute**.
- The action method handling an action event can be implemented only by components that implement javax.faces.component.ActionSource.

Writing Managed Bean Methods 2-2

Writing a method to perform validation

- A managed bean method which performs the validation must accept the javax.faces.context.FacesContext, this object represents the component whose data must be validated.
- Referred through validator attribute of the component.

Writing a method to handle a value change event

- Value change events are handled through a method which accepts a value change event and returns void.

Configuring JSF Applications 1-2

Configuration of the application is essential to ensure appropriate functioning of the application.

Following are the tasks required to configure the applications:

- Registering the managed beans with the application.
- Configuring the managed beans.
- Defining the navigation rules among the pages of the application.
- Packaging the application to include all the resource files in the application.

JSF provides a portable configuration document which is an XML file to configure the resources of the application.

faces-config.xml file is one such configuration file.

Configuring JSF Applications 2-2

Each configuration file must include the following information in the given order:

- XML version number with an encoding attribute.
- A faces-config tag with all the required declarations of the tag libraries and name spaces.

The application can have more than one configuration file which can be located in any one of the following ways:

- Locating a resource /META-INF/faces-config.xml in the application's /WEB-INF/lib/ directory.
- A context initialization parameter javax.faces.application.CONFIG_FILES in the Web deployment descriptor specifies the path to the multiple configuration files of the Web application.
- A resource named faces-config.xml is the configuration file for simple Web applications.

It is important to define an order in which multiple configuration files are accessed. The order of access is determined by an orderingXML element.

Using Converters and Validators in a JSF Application 1-3

Following demonstrates the usage of converters and validators in a JSF page:

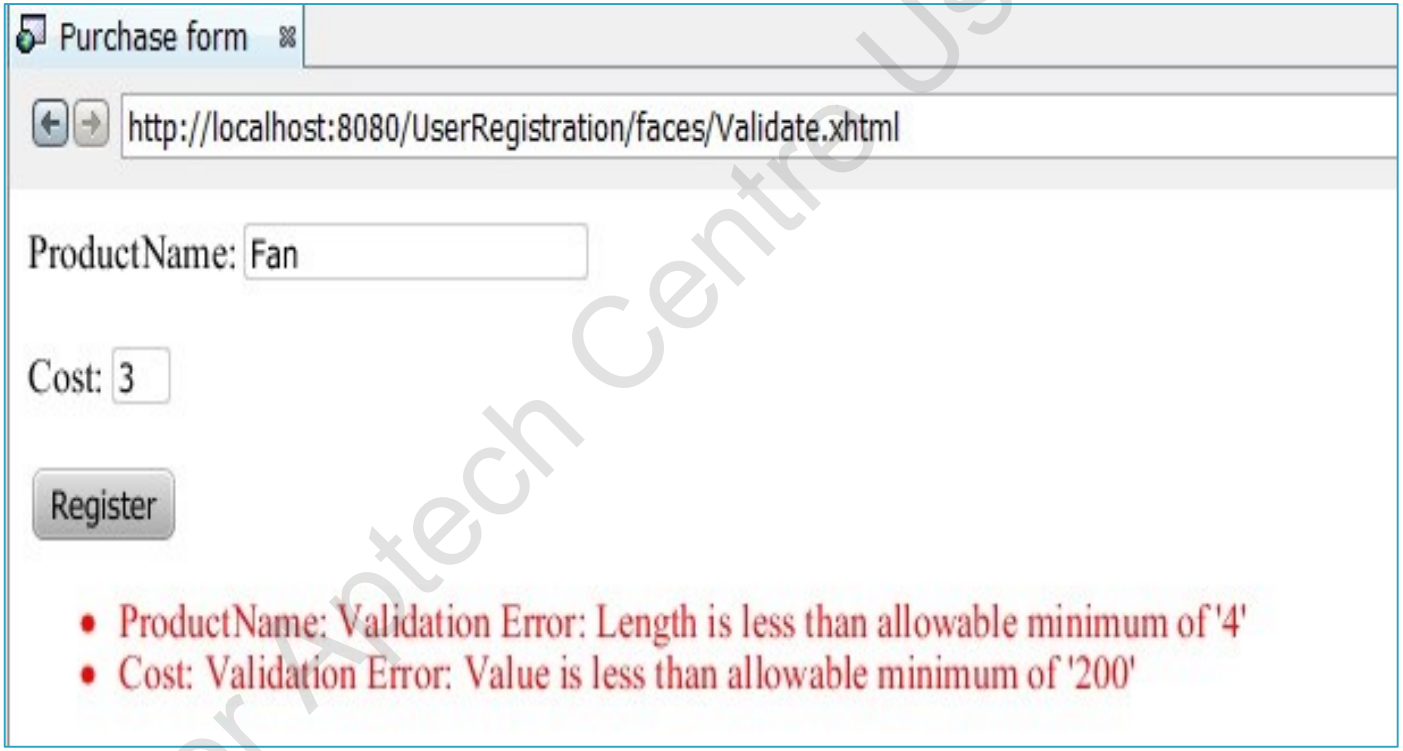
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
    <h:head>
        <title>Purchase form</title>
    </h:head>
    <h:body>
        <h:form>
            <f:view>
         <h:outputLabel for="Name" value="ProductName:"/>
        <h:inputText id="Name" label="ProductName" required="true" >
            <f:validateLength minimum="4"></f:validateLength>
        </h:inputText>
```

Using Converters and Validators in a JSF Application 2-3

```
        <h:message for="ProductName" />
        <p></p>
         <h:outputLabel for="cost" value="Cost:"/>
         <h:inputText id="cost" label="Cost" size="2" >
             <f:validateLongRange minimum="200" />
              <f:convertNumber maxFractionDigits ="2" />
          </h:inputText>
         <p></p>
         <h:commandButton id="register" value="Register"
                action="confirm" />
        </f:view>
        </h:form>
  </h:body>
</html>
```

Using Converters and Validators in a JSF Application 3-3

Following figure shows the resultant JSF page when the defined converters and validators are activated:



Implementing Converters in a JSF Application through Managed Bean 1-4

Following code demonstrates the converter implementation through a managed bean:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!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://xmlns.jcp.org/jsf/html"
      <h:head></h:head>
    <h:body>
        <h:form>
            <h3> Currency Converter </h3>
            <p></p>
   <h:outputLabel for="rupees" value="Amount in Rupees:"/>
   <h:inputText id="rupees" value="#{currencyBean.rupees}"/>
             <h:commandButton  id="submit" value="Submit" action =
"response"/>
        </h:form>
    </h:body>
</html>
```

Implementing Converters in a JSF Application through Managed Bean 2-4

Following code demonstrates the managed bean associated with the JSF page:

```
package converter;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
@ManagedBean (name = "currencyBean")
@SessionScoped
public class CurrencyBean {
int rupees;
int dollars;
   public CurrencyBean() {    }
    public int getRupees() {
        return rupees;
    }
    public void setRupees(int rupees) {  this.rupees = rupees;  }
    public int getDollars() { this.dollars = ((this.rupees)/60);
        return dollars;   }
    public void setDollars(int dollars) {
        this.dollars = dollars;    }   }
```

Implementing Converters in a JSF Application through Managed Bean 3-4

Following code displays the implementation of the response JSF page:

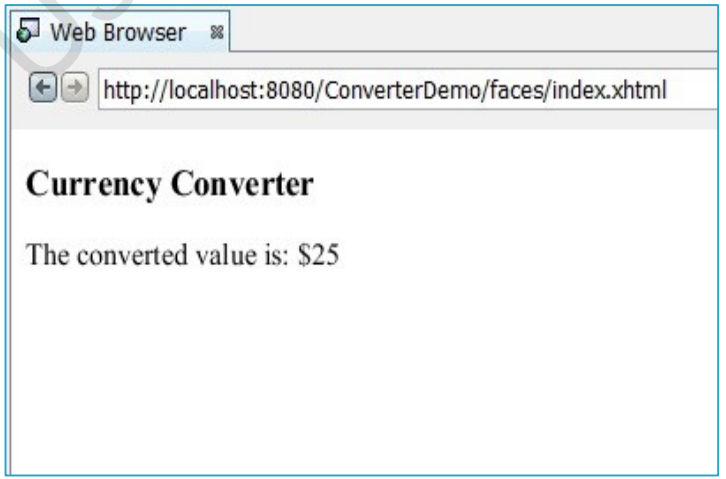
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
    </h:head>
    <h:body>
        <h3> Currency Converter </h3>
        <h:outputText id="output"  value="The converted value is:
$"/>
         <h:outputText id="out"  value="#{currencyBean.dollars}"/>

    </h:body>
</html>
```

Implementing Converters in a JSF Application through Managed Bean 4-4

Following figures show the execution of the application:

Index Page



Response Page

Summary

- JSF tags can implement what HTML tags can along with additional functionality through managed beans.
- Each tag has various attributes to define the behavior of the components associated with the tag.
- Apart from the HTML tags, JSF also has a set of core tags, which are used for event handling, data conversion, and so on.
- JSF defines a standard set of validators, listeners, and converters which can be used to act along with the components.
- Validators, listeners, and converters can be implemented through managed bean methods.
- Methods and properties of the components can be implemented through managed beans.
- The configuration information of JSF applications is stored in XML files. Each application can have multiple configuration files.