IBM WebSphere Application Server V6

JavaServer Faces Architecture

e-business on demand.

© 2004 IBM Corporation
Converted to video July 7, 2015

This presentation will provide an overview of the JavaServer Faces architecture.

Goals

- Provide an overview of JavaServer Faces (JSF) Architecture
- Provide examples to showcase various JSF architectural design points

© 2004 IBM Corporation

The goals for this presentation are to provide and overview of the JavaServer Faces architecture and to provide examples that showcase various JSF architectural design points.

Agenda

- JavaServer Faces Runtime
- Architecture Details
- Summary and References

© 2004 IBM Corporation

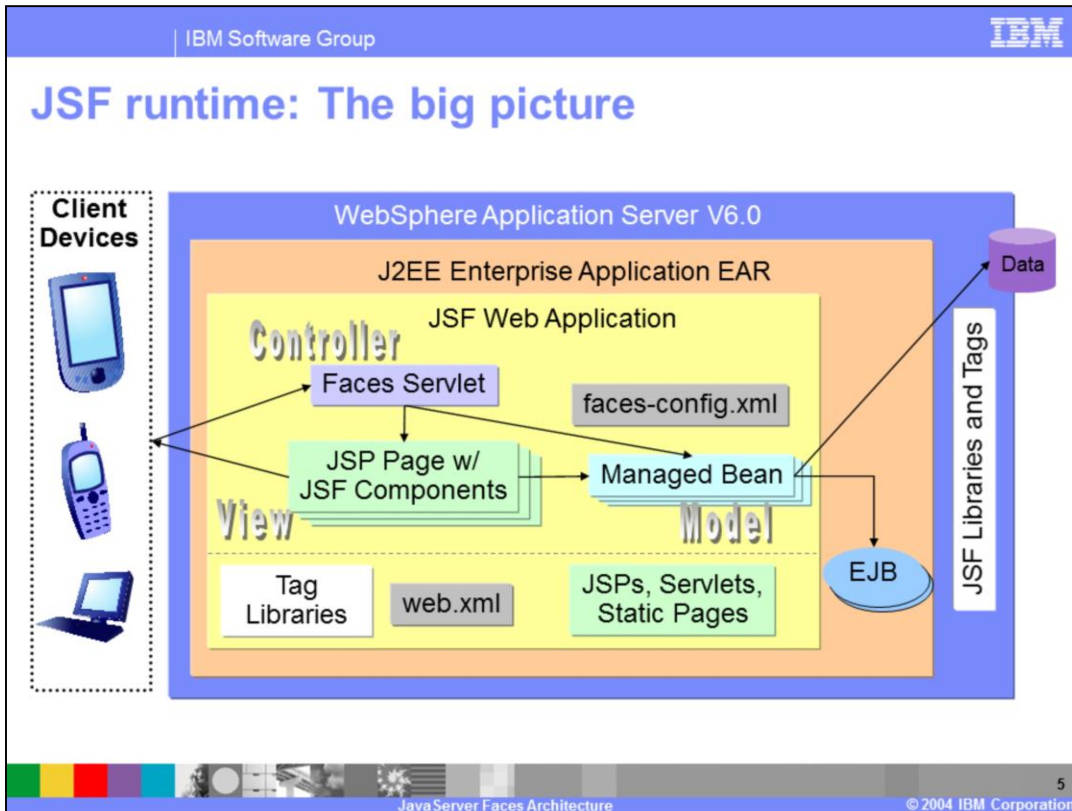
This presentation starts with a look at the JSF architecture. The second part of the presentation will provide examples of JSF concepts.

Section

JavaServer Faces runtime

© 2004 IBM Corporation

This section will discuss the JavaServer Faces runtime architecture.



This diagram gives a high level view of some of the important parts that make up a JSF application. This section will focus on providing an overview of some of the items that are specific to a JSF application. These items are included above the dashed line in the box entitled "JSF Web Application". Note from this diagram that the JSF runtime JAR files and tag libraries are included with the WebSphere Application Server Version 6.0 runtime. Any JAR files and tag libraries that support JSF custom UI components are packaged with the individual JSF Web Application.

The following is a description of some of the resources that make up a JSF application (in addition to those listed above):

JSF Page: A JSP page with at least one JSF component (a JSF custom tag). The JSF page is the View component in the MVC paradigm.

FacesServlet: The FacesServlet is provided by the JSF runtime implementation. This is the controller servlet that manages all incoming requests for JSF pages and passes these requests to the lifecycle management implementation of the JSF runtime.

Managed Beans: Managed beans represent the model aspect of the MVC design aspect of JSF. A managed bean is a Java bean whose state is managed by the JSF Framework

Validators: Validators are java classes that implement methods used to validate UI input. These classes implement the Validator interface provided with the JSF APIs.

Event Handlers: Event handlers are classes that implement methods to handle certain UI events. These events may include such things as action events like button clicks, or value changed events which are associated with input components.

faces-config.xml: The faces-config.xml file contains the configuration necessary for the JSF framework to run a particular JSF application.

There are several other terms that are important to introduce at this time since they will seen in the remaining part of this presentation. These terms include:

FacesContext: The FacesContext is a Java object that contains all the information needed to processes a request to a faces resource. This object is passed between the various phases in the request processing lifecycle, and maybe updated during this time; for example, to add an error message.

Lifecyle: The Lifecycle is a Java object that is used to manage the request processing lifecycle for a faces request.

JSF tag libraries

- There are two JSF tag libraries
  - JSF Core
    - Provides the core tags that are independent of the renderer type
    - Allows you to register such things as validators and event listeners
    - ```
      <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
      ```
  - HTML Basic
    - Tag Library that supports component tags for the Standard HTML renderer
    - Components included in this library are: buttons, text fields, checkboxes, lists, etc…
    - ```
      <%@ taglib uri="http://java.sun.com/jsf/html" prefix="“h" %>
      ```
- IBM provides several tag libraries that support IBM custom UI Components

There are two standard JSF tag libraries. The first of these tag libraries is the JSF core tag library. This library includes the core JSF tags that are independent of the renderer type. The tags in this library are used to register such things as event listeners and validators for UI components. The second standard JSF tag library is the HTML Basic tag library. The tags in this group is used to support UI component tags for the standard HTML renderer. As an example of the components included in this library there are tags for buttons, text fields, checkboxes, and so on.

In addition to these libraries, there are several tag libraries that are provided to support the IBM custom UI components. This support is provided with the IBM Rational® Application Developer V6 product.

IBM Software Group | IBM

JSF configuration: faces-config.xml

- Default JSF configuration file bundled in a Web application
- The faces-config.xml file is the place to configure
  - Managed beans
  - Validators
  - Converters
  - Navigation rules
  - Custom components
  - Lifecycle customizations

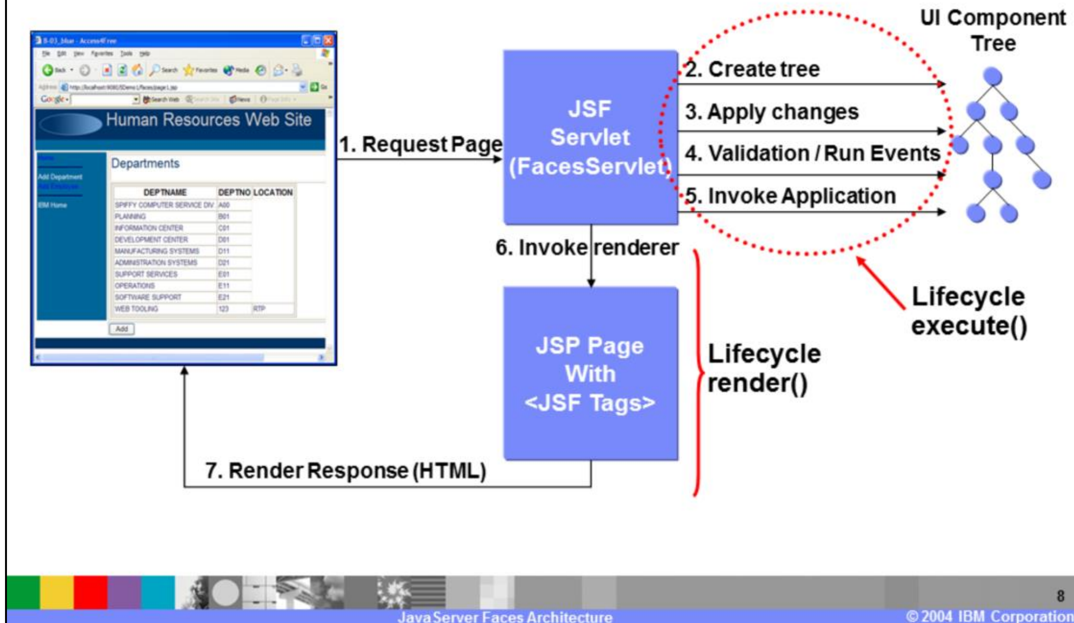
7

JavaServer Faces Architecture

© 2004 IBM Corporation

The faces-config.xml file is a configuration file used to store the configuration for a specific JSF application. This configuration file is packaged with the application WAR file.

This slide lists several of the configurable items available in the faces-config.xml file. The next section will illustrate examples highlighting various JSF features such as navigation, validators, converters, and so on. When appropriate in these examples, an example of the appropriate entry in the faces-config.xml will be highlighted.



The FacesServlet is provided by the JSF implementation, and represents the controller Servlet for the JSF application. The FacesServlet is responsible for managing all incoming requests and passing these requests to the lifecycle management process. Because the FacesServlet acts as the controller Servlet, an entry for it must appear in the web deployment descriptor for the application.

The FacesServlet is responsible invoking the appropriate Lifecycle management methods to manage the incoming request.

For each HTTP request the FacesServlet:

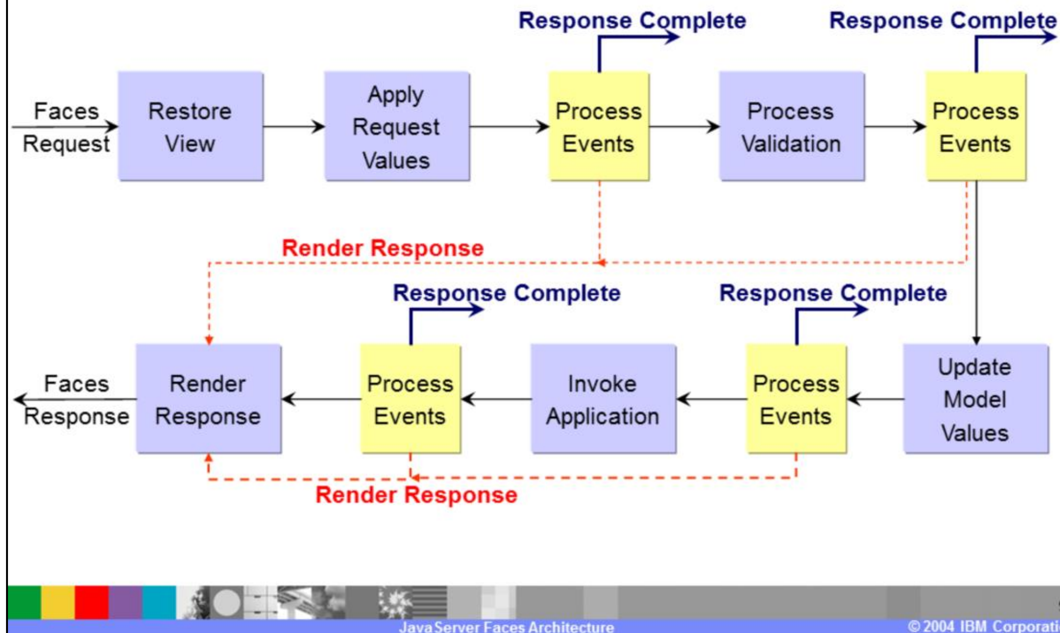
•Acquires a FacesContext and a Lifecycle instance

•Calls the execute() method on the Lifecycle instance

•Calls the render() method on the Lifecycle instance, and

•Releases the FacesContext instance

During the Lifecycle.execute() call, the majority of the faces processing occurs. A detailed explanation is included on the next slide. It is during the Lifecycle.render() call that the response that is returned to the client is generated and passed back to the client browser.



This slide provides a detailed view of the JSF Request processing lifecycle. This diagram can be found in the JSF specification referenced at the end of this presentation. The following is a description of each step in the request phase:

Restore view

The primary task during this phase is to restore or create the UI component tree and store this information as the viewRoot property on the FacesContext. If the page being requested is being accessed for the first time during the current session, then the UI component tree will be created. However, if this page has already been accessed then the component tree is rebuilt from state information stored by the server.

Apply Request Values

During the Apply Request Values phase the current state of each component in the UI tree is updated with information included with the incoming request. Recall that this information comes in the form of parameters, cookies, headers, and so on. During this phase, these values are stored in the UI component and not on a business model object (this comes later). During this phase conversions are preformed and events that have been queued are handled. If any conversion errors occur during this process, these messages will be queued in the FacesContext. If any of the methods called during this phase to convert values or handle events calls responseComplete() on the FacesContext lifecycle processing is ended for this request. If any methods all renderResponse(), the remaining phases in the lifecycle are skipped and the faces response is rendered.

Process Validation

During the Process Validation phase all registered validators (zero or more) for each UI component is called to check for validation errors. If a validation error does occur, error messages can be queued in FacesContext and the valid property on the particular UI component is set to false.

Update Model Values

During this phase, business objects (model objects) are updated with the values stored on the UI component tree in the Apply Request changes.

Invoke Application

During the Invoke Application phase, application level events are handled. Examples of such events includes pressing a submit button on a form or clicking a command link.

Render Response

The primary task during the Render Response phase is to send a response to the client for the appropriate rendering. It is also during this phase that the state of the UI component tree is saved on the server in user session data for later requests.

IBM Software Group | IBM

Section

JavaServer Faces details

The next section will highlight some JavaServer Faces architectural details.

IBM Software Group

Development steps: Overview

- Create JSF pages with appropriate standard or custom component tags
  - Input/Output UI components
  - Command components
- Develop managed beans
- Configure attributes for each JSF component
- Add event listeners to listen and act on events
- Specify navigation rules for response

The following section will provide a number of examples that highlight many of the important JSF architecture details. Before jumping into these examples, it is important to discuss a basic set of JSF development steps so that you will better under stand some of the examples included in this section.

The first step to developing a JSF application is to create one or more JSF pages. Each page will generally include a number of input and output components and usually at least one command component of some sort, like a submit button. Also important in JSF development is creating managed beans. The data encapsulated in the managed bean will be used to bind to the various UI components on the JSF page. All UI components included on a JSF has many attributes associated with the component that can be set to define the configuration for those components. Generally a JSF page will include one or more event listener to act on various user events. Finally, if your application includes more than one JSF page, you will likely need to define one or more navigation rules in the faces-config.xml file.

Event and Listener Model

- The JSF framework provides support for creating and handling UI Events
- Event processing can occur at several points in the request processing lifecycle
- Two types of component events and corresponding listeners are available

Event	Listener	Comments
ActionEvent	ActionListener	Generated by UICommand components
ValueChangedEvent	ValueChangedListener	Generated by UIInput components

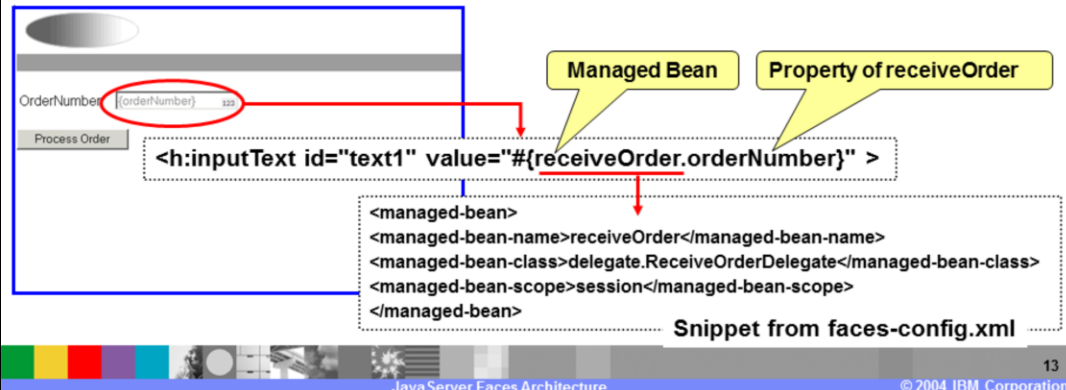
© 2004 IBM Corporation

The JSF framework provides support for creating and handling UI events. There are several points in the request processing lifecycle. Although the JSF framework provides an API for developers to build their own Events and Listeners to fit into the JSF framework, there are two standard events and corresponding listeners already provided with the JSF runtime. These events and listeners are listed on the table shown on this slide. In the next several slides, there are examples of the ActionEvent associated with a command button.

IBM Software Group IBM

Value binding expressions

- Allows attributes and properties to be dynamically calculated
- Syntax of value binding expressions are similar to Expression Language (EL) expressions defined in JSP 2.0



Each UI component included on a JSF page is typically bound to a particular data property associated with a business object on the server side. A value of a particular UI component is bound to this business object property through a value binding expression. A value binding expression is a way to allow attributes of a JSF UI component to be bound dynamically to a specific data source. The syntax of a value binding expressions are similar to Expression Language (EL) expressions defined in JSP 2.0.

The example on this slide illustrates a input component whose value is bound to a property (orderNumber) on a managed bean (receiveOrder). The value binding expression is the “value” attribute on the h:inputText tag. The example also highlights the managed bean definition found in the faces-config.xml file.

Method binding expressions

- Allows the invocation of the specified method of a particular object during request processing
  - Method must take a specified set of parameters
  - Returns result from the called method (if any)

OrderNumber: {orderNumber}

Process Order

Snippet from faces-config.xml

```
<managed-bean>
<managed-bean-name>receiveOrder</managed-bean-name>
<managed-bean-class>delegate.ReceiveOrderDelegate</managed-bean-class>
<managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

```
<hx:commandExButton id="button1" action="#{receiveOrder.processOrder}">
```

Managed Bean

Method on receiveOrder

© 2004 IBM Corporation

A method binding expression is used when you want to bind a particular property to the output returned from the invocation of a specified method on a managed bean during request processing. Not all attributes on JSF components accept method binding expressions, and those that do are listed on the following slide. It is important to note that attributes that do accept method binding expressions must conform to a specific method signature.

The example on this slide shows a method binding expression used with the action attribute of a command button. For this example, there is a managed bean called receiveOrder defined in the faces-config.xml file, and the Java class represented by this managed bean includes a method called processOrder. As you will learn on the next slide, methods used in a method binding expression for the action attribute must take no parameters and return a string.

Method binding expressions (cont.)

- The following lists the standard JSF component properties that support method binding expressions

Component Property	Method Signature
action	public String methodName()
actionListener	public void methodName(javax.faces.event.ActionEvent)
validator	public void methodName(javax.faces.context.FacesContext, javax.faces.component.UIComponent, java.lang.Object)
valueChangeListener	public void methodName(javax.faces.event.ValueChangeEvent)

© 2004 IBM Corporation

This slide lists the standard component properties that accept method binding expressions.

You were introduced to the action property in the previous example. This property can be associated with a command component. In this case, the action property action specifies the value returned when a command component is clicked. The method bound to this property is called during the “Invoke Application” phase of the request processing lifecycle. The string result that is returned by this method is bound to the action property and is used in conjunction with the Navigation handler to decide what to do based on the return. An example of this will be shown later on in this section.

The actionListener and valueChangedListener are properties that can be set on particular UI components to register methods to handle an ActionEvent or ValueChangedEvent, respectively. If you choose to use this approach, then there is no need to write your own Listener class to implement these methods. In this case you can simply add these methods to a managed bean class and use the method binding expression.

The validator property can be used to specify a validation method without needing to implement the Validator interface. This can be used on input components to validate input. The following three examples illustrate the use of the validation, and in particular there is an example of using the validator property along with a method binding expression.

IBM Software Group | IBM

Validation Model: Standard Validators

OrderNumber {orderNumber}

Process Order

More than one validator can be registered with a given UI component

```
<h:inputText id="t1" value="#{receiveOrder.orderNumber}" styleClass="inputText" required="true">
  <f:convertNumber type="number"/>
  <f:validateDoubleRange minimum="0.0"></f:validateDoubleRange>
</h:inputText>
<h:message styleClass="message" id="message1" for="t1"></h:message>
```

OrderNumber: Validation Error: Value is required.

Process Order

Validation error messages are added to the FacesContext

JavaServer Faces Architecture 16 © 2004 IBM Corporation

The JSF framework provides support for validating user input during the “Process Validations” lifecycle phase. There are two types of validators. These validators include:

Standard: The JSF runtime comes with a set of standard classes and tags to do several common validation tasks for UI input components.

Custom: Custom validators are implemented by application developers. To build these validators, you must implement the javax.faces.validator.Validator interface, or provide a validator method as part of a managed bean and use the validator property as discussed on the previous slide.

This slide provides an example of some of the standard validators that are available with the base JSF implementation.

Validation Model: Custom Validator (Example 1)

OrderNumber {orderNumber}

Process Order

Use a method binding expression

```
<h:inputText id="t1" value="#{receiveOrder.orderNumber}"
styleClass="inputText" validator="#{receiveOrder.checkOrder}">
</h:inputText>
```

receiveOrder Managed Bean Class

```
public void checkOrder(javax.faces.context.FacesContext context,
                       javax.faces.component.UIComponent component, java.lang.Object value) {
...
   if(!valid) {
     component.setValid(false);
     context.addMessage(component.getClientId(context)), errMsg);
   }
...
}
```


© 2004 IBM Corporation

This examples shows how to set a custom validator using a method binding expression defined on an inputText component. In this example, there is no need to add any validator configuration in the faces-config.xml file. The only thing that needs to be done is to include the validator attribute with a method binding expression on the inputText field and also add the appropriate method to the managed bean referenced in the method binding expression. Note that the method that is added to the managed bean must match the method signature shown above.

Validation Model: Custom Validator (Example 2)

OrderNumber {orderNumber}

Process Order

Custom Validator implementation

```
<h:inputText id="t1" value="#{receiveOrder.orderNumber}" styleClass="inputText" >
  <f:validator id="orderNumValidator"></f:validator>
</h:inputText>
```

```
<validator>
<validator-id>orderNumValidator</validator-id>
<validator-class>validators.OrderNumValidator</validator-class>
</validator>
```

faces-config.xml

User provided class that implements the javax.faces.validator.Validator interface

© 2004 IBM Corporation

This examples shows another alternative to using a custom validator. Here you provide a class that implements the javax.faces.validator.Validator interface. In this case, rather than using the validator property on the inputText component, the <f:validator> tag is used to specify the custom validator. Unlike the last example, in this case it is necessary to add a validator configuration item in the faces-config.xml file. Note that it is also possible to define a custom tag to represent your custom validator and use this in place of the <F:validator> tag.

IBM Software Group | IBM

Conversion Model

- Supports type conversion between presentation layer and the underlying business objects
- A standard set of converter implementations are available
- Custom converters can be developed by implementing the javax.faces.convert.Converter interface

Example

```
<h:inputText styleClass="inputText" id="text2"> <f:convertDateTime /> </h:inputText>
```

The conversion model is a way to convert user input from the presentation view to the model view. These two views typically work with a different form of the data. The model will often deal with Java objects, while the view may need a particular way to display the data (like a string). Converters are used to transform data between the model and presentation views. A converter is attached to a UI component that is a value holder of some sort.

The JSF runtime includes a standard set of converter implementations. In addition to this, there is also support for developing a custom converter by implementing the javax.faces.convert.Converter interface. Custom converters are defined in the faces-config.xml file.

IBM Software Group | IBM

Navigation Model

```
<navigation-rule>   ← Page Navigation Rule
  <from-view-id>/supply.jsp</from-view-id>
  <navigation-case>
    <from-action>#{receiveOrder.processOrder}</from-action>
    <from-outcome>success</from-outcome>
    <to-view-id>/confirmation.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-action>>#{receiveOrder.processOrder}</from-action>
    <from-outcome>failure</from-outcome>
    <to-view-id>/failure.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
<navigation-rule>   ← Global Navigation Rule
  <navigation-case>
    <from-outcome>error</from-outcome>
    <to-view-id>error.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

Snippet from faces-config.xml

receiveOrder Managed Bean Class

```
…
public static String processOrder() {
  try {
    doProcessing();
    return "success";
  } catch (ProcessingException pe) {
    return "failure";
  } catch (Exception e) {
    return "error";
  }
}
…
```

Navigation rules are selected based on the return values from method bindings specified for the action component property

20

JavaServer Faces Architecture © 2004 IBM Corporation

Navigation rules are specified in the faces-config.xml file. At runtime, these rules are selected based on the current page (from-view-id) and the return values from method bindings specified for the action component property on a command UI component (command button or command link).

It is important to point out that there are two types of navigation rules. These navigation types are:

Global: Application wide navigation rule indicated when <from-tree-id> is omitted, and

Page-specific: A navigation rule that only applies when the request page matches the <from-tree-id> element

Section

Summary and Reference

© 2004 IBM Corporation

The next section will provide a summary and references for this presentation.

Summary

- JSF is a new framework for rapid Web application development
  - Tool-based
  - Framework to provide structure for large development teams and ease of maintenance
  - Isolates developers from heavy development work
- Based on a Model-View Controller architecture and Event-Driven model
- Numerous extension points for adding custom validators, event handlers, and navigators
- Future programming model for developers of J2EE applications

© 2004 IBM Corporation

In summary, this presentation has focused on providing an overview for the JavaServer Faces framework. This technology is intended to be incorporated with tool support to enable rapid application development to J2EE web development.

The JSF framework is based upon a Model-View-Controller based design paradigm, and also includes an event driven model that provides an easy framework for to handle user events when building web based applications.

The JSF framework is extremely flexible and includes many extension points for adding such things as validators, event handlers, and converters.

References

- JSR 127

  http://java.sun.com/j2ee/javaserverfaces/

- JSF and Struts

  http://www-106.ibm.com/developerworks/java/library/j-integrate/

© 2004 IBM Corporation