JavaServer Faces

- Sesión 3: Ciclo de vida JSF. Conversores. Validadores. Eventos.

 © 2012-2013 Depto. Ciencia de la Computación e IA

Índice

- El ciclo de vida de una petición
- Validaciones
  - Custom validators
  - JSR 303
- Conversiones
  - Custom converters
- Gestión de eventos
  - Validación mediante eventos
  - Toma de decisiones

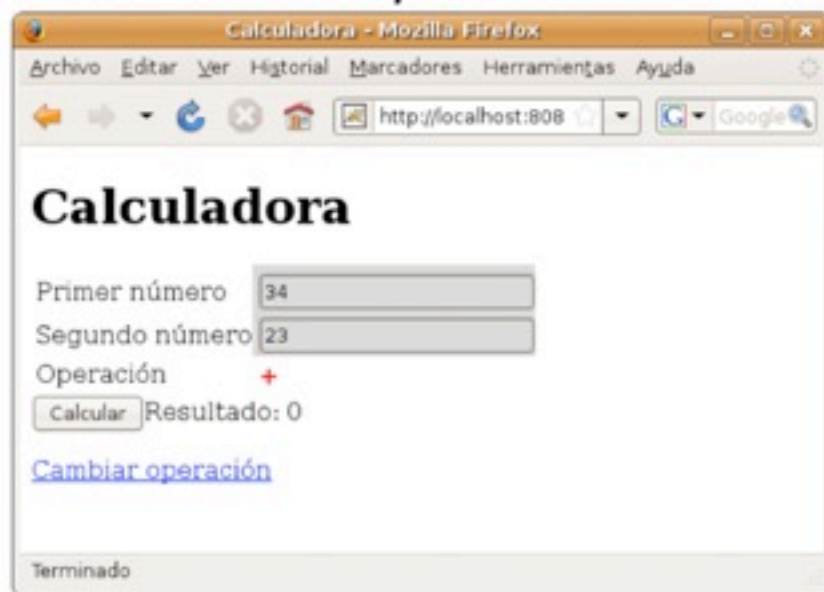
© 2012-2013 Depto. Ciencia de la Computación e IA

El ciclo de vida visto desde fuera

Declaración de la interfaz de usuario en JSF

```
<f:view>
  <h1>Calculadora</h1>
    <h:form id="calcForm">
      <h:panelGrid columns="3">
        <h:outputLabel value="Primer número"/>
        <h:inputText id="firstNumber"
           value="#{calculatorBB.firstNumber}"
            required="true"/>
        <h:message for="firstNumber"/>

        <h:outputLabel value="Segundo número"/>
        <h:inputText id="secondNumber"
           value="#{calculatorBB.secondNumber}"
...
```

Datos introducidos por el usuario



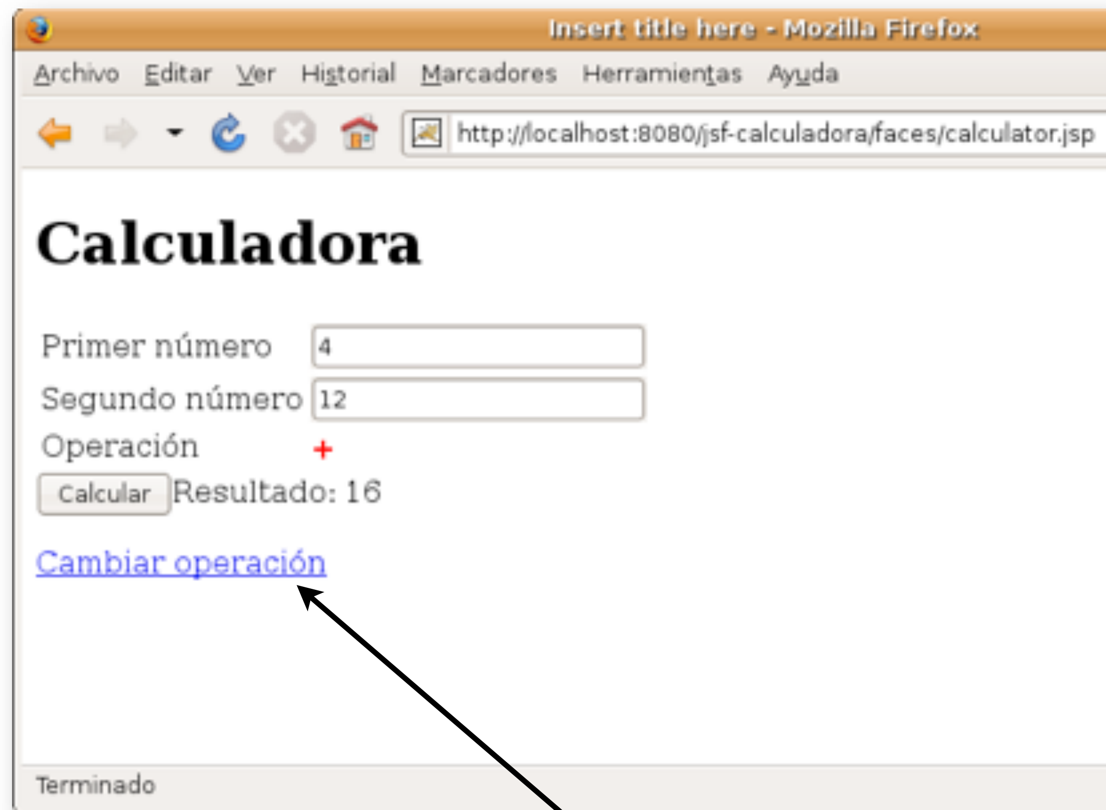
Página HTML

```
<!DOCTYPE html PUBLIC
   "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"
   content="text/html; charset=UTF-8">
<LINK href="/jsf-calculadora/css/mystyle.css"
     rel="stylesheet" type="text/css">

<title>Calculadora</title>
</head>
<body>

<h1>Calculadora</h1>

<form id="calcForm" name="calcForm"
  method="post"
  action="/jsf-calculadora/faces/calculator.jsp"
  enctype="application/x-www-form-urlencoded">
<input type="hidden" name="calcForm"
  value="calcForm" />

<table>
<tbody>
<tr>
<td><label>
Primer n&uacute;mero</label></td>
<td><input id="calcForm:firstNumber"
   type="text"
   name="calcForm:firstNumber"
   value="34" /></td>
<td></td>
</tr>
<tr>
<td><label>
Segundo n&uacacute;mero</label></td>
...
```

El ciclo de vida visto desde dentro

Response Complete
Response Complete
Faces Request
Restore View
Apply Request Values
Process Events
Process Validations
Process Events
Render Response
Response Complete
Response Complete
Faces Response
Render Response
Process Events
Invoke Application
Process Events
Update Model Values
Conversion Errors/ Render Response
Validation / Conversion Errors / Render Response

Programa ejemplo: calculadora



Componentes que se hacen visibles y se ocultan

Código de la vista

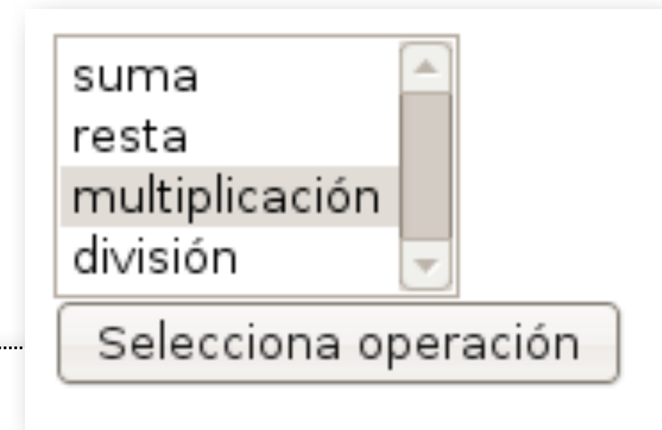
```
<f:view>
   <h:form id="calcForm">
      <h:panelGrid columns="3">
         <h:outputLabel value="Primer número"/>
         <h:inputText id="firstNumber"
                      value="#{calcBean.firstNumber}"
                      required="true"/>
         <h:message for="firstNumber"/>
         ...
         <h:commandButton value="Calcular"
                action="#{calculatorController.doOperation}"/>
         <h:outputText value="Resultado: #{calculatorBB.result}"/><br/>
         <p></p>
         <h:commandLink
                rendered="#{calculatorController.newOperationCommandRendered}"
                action="#{calculatorController.doNewOperation}"
                value="Cambiar operación"/>
          ...
   </h:form>
 </f:view>
```

Componentes (1)

Primer número 3
Segundo número 23
Operación +
Calcular Resultado: 26
Cambiar operación

```
<h:form id="calcForm">
   <h:panelGrid columns="3">
         <h:outputLabel value="Primer número"/>
         <h:inputText id="firstNumber"
                       value="#{calcBean.firstNumber}"
                       required="true"/>
         <h:message for="firstNumber"/>
         ...
         <h:commandButton value="Calcular"
                action="#{calculatorController.doOperation}"/>
         <h:outputText value="Resultado: #{calculatorBB.result}"/><br/>
         <p></p>
         <h:commandLink
                rendered="#{calculatorController.newOperationCommandRendered}"
                action="#{calculatorController.doNewOperation}"
                value="Cambiar operación"/>
          ...
</h:form>
```

Componentes (2)




```
<h:form rendered="#{calculatorController.selectOperationFormRendered}">
   <h:selectOneListbox id="operation"
                       required="true"
                       value="#{calculatorBB.operation}">
       <f:selectItem itemValue="+" itemLabel="suma"/>
       <f:selectItem itemValue="-" itemLabel="resta"/>
       <f:selectItem itemValue="*" itemLabel="multiplicación"/>
       <f:selectItem itemValue="/" itemLabel="división"/>
   </h:selectOneListbox><br/>
   <h:commandButton
        action="#{calculatorController.doSelectOperation}"
        value="Selecciona operación"/>
</h:form>
```

Renderizado de los componentes

- Se define con las propiedades booleanas `newOperationRendered` y `selectOperationFormRendered` en el bean `calculatorController`
- Las acciones modifican el valor booleano de esas propieades

```
<f:view>
   <h:form id="calcForm">
      ...
      <h:commandLink
              rendered="#{calculatorController.newOperationCommandRendered}"
              action="#{calculatorController.doNewOperation}"
              value="Cambiar operación"/>
      ...
   </h:form>
   <h:form rendered="#{calculatorController.selectOperationFormRendered}">
      <h:commandButton
            action="#{calculatorController.doSelectOperation}"
            value="Selecciona operación"/>
     </h:form>
 </f:view>
```

¿Cuándo se crea el árbol de componentes?

- En la primera petición (*http://localhost:8080/calculator*) se crea el árbol de componentes a partir del fichero calculator.xhtml
- El árbol de componentes (vista) se guarda en el servidor.
- En la segunda petición (cuando el usuario pincha en el enlace “calcular” y se envía el formulario al servidor), JSF obtiene el árbol creado anteriormente, lo guarda en la petición y le aplica el ciclo de vida a la petición.

Controlador (1)

Propiedades

Mostrando y ocultando

```
public class CalculatorController {
  private CalculatorBB numbers;
  private CalculatorBO calculator = new CalculatorBO();
  private boolean selectOperationFormRendered=false;
  private boolean newOperationCommandRendered=true;

  ...
  // getters y setters

  public String doNewOperation() {
     selectOperationFormRendered=true;
     newOperationCommandRendered=false;
     return null;
  }

  public String doSelectOperation() {
     selectOperationFormRendered=false;
     newOperationCommandRendered=true;
     doOperation();
     return null;
  }
  ...
```

Controlador (2)

Obtenemos los valores introducidos por el usuario leyéndolos del backing bean,

llamamos a la capa de negocio

y ponemos el resultado en el backing bean.

```
...
public String doOperation() {
    String operation = numbers.getOperation();
    int firstNumber = numbers.getFirstNumber();
    int secondNumber = numbers.getSecondNumber();
    int result = 0;
    String resultStr = "OK";

    if (operation.equals("+"))
       result = calculator.add(firstNumber, secondNumber);
    else if (operation.equals("-"))
       result = calculator.substract(firstNumber, secondNumber);
    else if (operation.equals("*"))
       result = calculator.multiply(firstNumber, secondNumber);
    else if (operation.equals("/"))
       result = calculator.divide(firstNumber, secondNumber);
    else
       resultStr="not-OK";
    numbers.setResult(result);
    return resultStr;
  }
```

Conversión de datos

- En una aplicación web, todos los datos se introducen como texto
- JSF convierte:
  - los datos a tipos java en la fase *Apply Request Values*
  - los tipos java a String en la fase *Render Response*
- JSF usa una serie de conversores por defecto para tipos básicos, aunque podemos escogerlos.

```
<h:outputText value="Fecha de salida: #{bean.fechaSalida}">
   <f:convertDateTime dateStyle="short"/>
</h:outputText>
```

Tipo	Formato
default	Sep 9, 2003 5:41:15 PM
short	9/9/03 5:41 PM
medium	Sep 9, 2003 5:41:15 PM
long	September 9, 2003 5:41:15 PM PST
full	Tuesday, September 9, 2003 5:41:15 PM PST

Custom converters

- En ocasiones podemos necesitar conversores más específicos (DNI, Tarjetas de crédito, ...)
- Para esos casos, podemos crearnos nuestros propios conversores.
- Deben implementar la interfaz `javax.faces.convert.FacesConverter`.
- Proporciona los métodos:
  - *Object getAsObject(FacesContext context, UIComponent component, String newValue)*:

    String ⟶ Objeto.

    Si no puede convertir ⟶ ConverterException
  - *String getAsString(FacesContext context, UIComponent component, Object value)*:

    Objeto ⟶ String

Custom converters

- Un conversor se define mediante la anotación @FacesConverter("ID_Conversor")

```
@FacesConverter("conversorDni")
public class DniConverter implements Converter {

  public Object getAsObject(FacesContext context, UIComponent component, String value)
                 throws ConverterException {
    boolean situacionDeError = false;
    DniBean dni = new DniBean();
    dni.setNumero(value.substring(0, 8));
    dni.setLetra(value.substring(8, 1));

    if (situacionDeError) {
      FacesMessage message = new FacesMessage(FacesMessage.SEVERITY_ERROR,
                                 "Se ha producido un error en la conversión",
                                 "Detalle del error");
      throw new ConverterException(message);
    }

    return dni;
  }

  public String getAsString(FacesContext context, UIComponent component, Object value) throws ConverterException {
    DniBean dni = (DniBean) value;
    return dni.getNumero() + dni.getLetra();
  }
}
```

Custom converters

- Haremos uso de un custom converter mediante el tag <f:converter>

```
<h:inputText value="#{usuario.dni}">
    <f:converter converterId="conversorDni"/>
</h:inputText>
```

```
<h:inputText value="#{usuario.dni}" converter="conversorDni"/>
```

Custom converters

- Podemos obviar el uso del tag <f:converter> si especificamos en la clase conversora que se aplique siempre para un tipo de objeto.

```
@FacesConverter(forClass=Dni.class)
...
```

- Así, será la implementación de JSF quien busque conversores para este tipo de objeto.

© 2012-2013 Depto. Ciencia de la Computación e IA

Custom converters

- Podemos pasar atributos a nuestro conversor mediante el tag <f:attribute>

```
<h:outputText value="#{usuario.dni}">
   <f:converter converterId="org.especialistajee.Dni"/>
   <f:attribute name="separador" value="-"/>
</h:outputText>
```

- El código en el conversor será

```
String separator = (String) component.getAttributes().get("separator");
```

Validadores

- Se encargan de que los datos introducidos tengan los valores esperados.

Tag	Validator	Atributos	Descripción
f:validateDoubleRange	DoubleRangeValidator	minimum, maximum	Un valor double, con un rango opcional
f:validateLongRange	LongRangeValidator	minimum, maximum	Un valor long, con un rango opcional
f:validateLength	LengthValidator	minimum, maximum	Un String, con un mínimo y un máximo de caracteres
f:validateRequired	RequiredValidator		Valida la presencia de un valor
f:validateRegex	RegexValidator	pattern	Valida un String contra una expresión regular
f:validateBean	BeanValidator	validation-Groups	Especifica grupos de validación para los validadores

Uso de los validadores

```
<h:outputLabel value="Primer número"/>
<h:inputText id="firstNumber"
   value="#{calculatorBean.firstNumber}"
   required="true">
      <f:validateLongRange minimum="0"/>
</h:inputText>
<h:message for="firstNumber"/>
```

```
<h:inputText id="card" value="#{usuario.dni}" required="true"
      requiredMessage="El DNI es obligatorio"
      validatorMessage="El DNI no es válido">
   <f:validateLength minimum="9"/>
</h:inputText>
```

Sobreescribiendo los mensajes de error

- Fichero de mensajes (veremos los Message Bundles en la próxima clase).

Resource ID	Texto por defecto
javax.faces.component.UIInput.REQUIRED	{0}: Validation Error: Value is required
javax.faces.validator.DoubleRangeValidator.NOT_IN_RANGE; javax.faces.validator.LongRangeValidator.NOT_IN_RANGE	{2}: ValidationError: Specified attribute is not between the expeced values of {0} and {1}
javax.faces.validator.DoubleRangeValidator.MAXIMUM; javax.faces.validator.LongRangeValidator.MAXIMUM	{1}: Validation Error: Value is greater than allowable maximum of {0}
javax.faces.validator.DoubleRangeValidator.MINIMUM; javax.faces.validator.LongRangeValidator.MINIMUM	{1}: Validation Error: Value is less than allowable minimum of {0}
javax.faces.validator.DoubleRangeValidator.TYPE; javax.faces.validator.LongRangeValidator.TYPE	{1}: Validation Error: Value is not of the correct type
javax.faces.validator.LengthValidator.MAXIMUM	{1}: ValidationError: Value is greater than allowable maximum of {0}
javax.faces.validator.LengthValidator.MINIMUM	{1}: ValidationError: Value is less than allowable maximum of {0}
javax.faces.valiadtor.BeanValidator.MESSAGE	{0}

© 2012-2013 Depto. Ciencia de la Computación e IA

JSR 303

- JSF2 incluye soporte para JSR 303 (Bean Validation Framework).
- Podemos incluir anotaciones en los objetos para indicar las validaciones a realizar en lugar de en la vista.
- Centralizamos las validaciones en la clase.
- Si introducimos una nueva validación, no tenemos que ir vista por vista.

http://download.oracle.com/otndocs/jcp/bean_validation-1.0-fr-oth-JSpec/

JSR 303

- Las anotaciones que ofrece son

Anotación	Atributos	Descripción
`@Null`, `@NotNull`	Ninguno	Comprueba que un valor sea nulo o no lo sea
`@Min`, `@Max`	El límite como `long`	Comprueba que un valor es, como máximo o como mínimo, el valor límite descrito. El tipo debe ser `int`, `long`, `short`, `byte`, o a de sus *wrappers* (`BigInteger`, `BigDecimal`, `String`)
`@DecimalMin`, `@DecimalMax`	El límite como `String`	Igual que la anterior, puede aplicarse a un `String`
`@Digits`	`integer`, `fraction`	Comprueba que un valor tiene, como máximo, el número dado de dígitos enteros o fraccionales. Se aplica a `int`, `long`, `short`, `byte`, o a de sus *wrappers* (`BigInteger`, `BigDecimal`, `String`)
`@AssertTrue`, `@AssertFalse`	Ninguno	Comprueba que un booleano es verdadero o false
`@Past`, `@Future`	Ninguno	Comprueba que una fecha esté en el pasado o en el futuro
`@Size`	`min`, `max`	Comprueba que el tamaño de una cadena, array, colección o mapa está en los límites definidos
`@Pattern`	`regexp`, `flags`	Una expresión regular, y sus flags opcionales de compilación

 © 2012-2013 Depto. Ciencia de la Computación e IA

JSR 303

- Para sobreescribir los mensajes de error, crearemos un fichero ValidationMessages.properties en la raíz del paquete.

```
javax.validation.constraints.Null.message=must be null
javax.validation.constraints.NotNull.message=must not be null
javax.validation.constraints.AssertTrue.message=must be true
javax.validation.constraints.AssertFalse.message=must be false
javax.validation.constraints.Min.message=must be greater than or equal to {value}
javax.validation.constraints.Max.message=must be less than or equal to {value}
javax.validation.constraints.Size.message=size must be between {min} and {max}
javax.validation.constraints.Digits.message= numeric value out of bounds (<{integer}
digits>.<{fraction} digits> expected)
javax.validation.constraints.Past.message=must be a past date
javax.validation.constraints.Future.message=must be a future date
javax.validation.constraints.Pattern.message=must match the following regular
expression: {regexp}
```

JSR 303

- Podemos definir valores específicos de error en nuestros validadores

```
@Size(min=9, max=9, message="{org.especialistajee.longitudDni}")
private String dni = "";
```

- En el fichero ValidationMessages.properties introduciremos la línea:

```
org.especialistajee.longitudDni = El DNI debe tener 9 caracteres
```

 © 2012-2013 Depto. Ciencia de la Computación e IA

Custom validators

- Podemos crear clases que realicen las validaciones que nosotros queramos.
  - 1. Crear una clase que implemente la interfaz `javax.faces.validator.Validator` e implementar el método `validate()` de esa interfaz.
  - 2. Anotarla con `@FacesValidator(“validator_id”)`
  - 3. Usar la etiqueta `<f:validator validatorId=”validator_id”/>` en las páginas JSF.

Custom validators

```
@FacesValidator("calculator.isPair")
public class PairNumberValidaotr implements Validator{
    public void validate(FacesContext context, UIComponent component, Object value)
                        throws ValidatorException {
        int number = ((Integer)value).intValue();

        if(number%2 != 0){
            FacesMessage message = new FacesMessage(FacesMessage.SEVERITY_ERROR,
                  "No es un número par", "No es un número par");
        }
    }
}
```

```
<h:inputText id="firstNumber" value="#{calcBean.firstNumber}" required="true">
    <f:validator validatorId="calculator.isPair"/>
</h:inputText>
```

FacesContext

- Toda petición JSF tiene asociado un contexto, en forma de una instancia de la clase FacesContext.
- Esta clase define un conjunto de métodos que nos permiten obtener y modificar sus elementos:
  - La cola de mensajes
  - El árbol de componentes
  - Objetos de configuración de la aplicación
  - Métodos de control del flujo del ciclo de vida

Ejemplo de código

```
package calculator.validator;
...
import javax.faces.component.UIComponentBase;
import javax.faces.component.UIViewRoot;
import javax.faces.context.FacesContext;

public class PairNumberValidator implements Validator {
    public void validate(FacesContext arg0,
                         UIComponent component,
                             Object value)
            throws ValidatorException {
        FacesContext context = FacesContext.getCurrentInstance();
        UIViewRoot viewRoot = context.getViewRoot();
        String ids = getComponentIds(viewRoot);
        FacesMessage message = new FacesMessage("Componentes: "+ ids);
        context.addMessage(null,message);
        ...
        }
    }

}
```

Pantalla

Calculadora - Mozilla Firefox

Archivo Editar Ver Historial Marcadores Herramientas Ayuda

http://localhost:8080/jsf-calculadora/faces/calculator.jsp

Apache Tomcat/6.0.18 - Info... | Apache Tomcat/6.0.18 - Info... | Calculadora

Calculadora

Primer número [3] No es un numero par

Segundo número [4]

Operación +

[Calcular] Resultado: 0.0

Cambiar operación

- Componentes: javax.faces.ViewRoot (j_id_jsp_437901315_0) javax.faces.Form (calculadora) javax.faces.Panel (j_id_jsp_437901315_2) javax.faces.Output (j_id_jsp_437901315_3) javax.faces.Input (firstNumber) javax.faces.Message (j_id_jsp_437901315_5) javax.faces.Output (j_id_jsp_437901315_6) javax.faces.Input (secondNumber) javax.faces.Message (j_id_jsp_437901315_8) javax.faces.Output (j_id_jsp_437901315_9) javax.faces.Output (j_id_jsp_437901315_10) javax.faces.Output (j_id_jsp_437901315_11) javax.faces.Command (Status) javax.faces.Command (j_id_jsp_437901315_13) javax.faces.Output (j_id_jsp_437901315_14) javax.faces.Command (j_id_jsp_437901315_15) javax.faces.Messages (j_id_jsp_437901315_16) javax.faces.Form (j_id_jsp_437901315_17) javax.faces.SelectOne (operation) javax.faces.SelectItem (j_id_jsp_437901315_19) javax.faces.SelectItem (j_id_jsp_437901315_20) javax.faces.SelectItem (j_id_jsp_437901315_21) javax.faces.SelectItem (j_id_jsp_437901315_22) javax.faces.Command (j_id_jsp_437901315_23)
- No es un numero par

Terminado

Usando componentes en los beans

- Es posible ligar en la página JSF un componente JSF a una propiedad del bean gestionado
- El bean debe tener un campo con un tipo compatible con el componente

```
<h:panelGroup>
   <h:inputText binding="#{miBean.inputText}"
                size="30"/>
   <h:commandLink value="Añadir projecto"
                actionListener="#{miBean.addNewProject}"
                immediate="true"/>
</h:panelGroup>
...
<h:selectOneMenu id="project" required="true"
    value="#{miBean.project}">
    <f:selectItems value="#{miBean.projects}" />
</h:selectOneMenu>
```

Bean gestionado

```
public class MiBean {
    ...
    private UIInput inputText;
    ...

    public UIInput getInputText() {
        return inputText;
    }

    public void setInputText(UIInput inputText) {
        this.inputText = inputText;
    }

    public void addNewProject(ActionEvent event) {
        String newProject = (String)inputText.getSubmittedValue();
        inputText.setSubmittedValue(null);
        projects.add(newProject);
    }
    ...
}
```

Elementos JSF y componentes

Etiqueta	Clase Java
<h:column>	UIColumn
<h:commandButton>	UICommand
<h:commandLink>	UICommand
<h:dataTable>	UIData
<h:form>	UIForm
<h:graphicImage>	UIGraphic
<h:inputHidden>	UIInput
<h:inputSecret>	UIInput
<h:inputText>	UIInput
<h:inputTextarea>	UIInput
<h:message>	UIMessage
<h:messages>	UIMessages
<h:outputFormat>	UIOutput
<h:outputLabel>	UIOutput
<h:outputLink>	UIOutput
<h:outputText>	UIOutput
<h:panelGrid>	UIPanel
<h:panelGroup>	UIPanel
<h:selectBooleanCheckbox>	UISelectBoolean
<h:selectManyCheckbox>	UISelectMany
<h:selectManyListbox>	UISelectMany
<h:selectManyMenu>	UISelectMany
<h:selectOneListbox>	UISelectOne
<h:selectOneMenu>	UISelectOne
<h:selectOneRadio>	UISelectOne

 © 2012-2013 Depto. Ciencia de la Computación e IA

Componentes JSF

<<interface>>
StateHolder

<<interface>>
NamingContainer

<<interface>>
ValueHolder

UIComponent

<<interface>>
ActionSource

<<interface>>
EditableValueHolder

UIComponentBase

UIOutput
UIPanel
UICommand
UIForm
UIData
UIViewRoot
UIMessages
UIInput
UIGraphic
UIMessage
UISelectOne
UISelectMany
UIColumn
UISelectBoolean

Gestión de eventos

- JSF Soporta cuatro tipos distintos de eventos
  - Value change events
  - Action events
  - Phase events
  - System events (JSF 2.x)

Value change events

- Los lanzan los elementos editables, cuando cambia el valor del componente

```
<h:selectOneMenu value="#{form.country}" onchange="submit()"
      valueChangeListener="#{form.countryChanged}">
   <f:selectItems value="#{form.countries}" var="loc"
      itemLabel="#{loc.displayCountry}" itemValue="#{loc.country}"/>
</h:selectOneMenu>
```

```
public void countryChanged(ValueChangeEvent event) {
   for (Locale loc : countries)
      if (loc.getCountry().equals(event.getNewValue()))
         FacesContext.getCurrentInstance().getViewRoot().setLocale(loc);
}
```

Value change events

- Métodos del objeto javax.faces.ValueChangeEvent
  - UIComponent getComponent(): devuelve el componente que disparó el evento
  - Object getNewValue(): devuelve el nuevo valor del componente, una vez convertido y validado
  - Object getOldValue(): devuelve el valor previo del componente,

© 2012-2013 Depto. Ciencia de la Computación e IA

Action events

- Los lanzan botones y enlaces.
- Se disparan durante la *Invoke Application Phase,* cerca del final del ciclo de vida.

```
<h:commandLink actionListener="#{bean.linkActivated}">
   ...
</h:commandLink>
```

- Los actionListeners no influyen en la navegación, sólo deben servir de apoyo a la lógica del controlador.
- Los actionListeners se invocan antes que las acciones

f:actionListener y f:valueChangeListener

- Tags análogos a los eventos que acabamos de ver.
- Ventaja: permiten asociar varios listeners a un mismo componente

```
<h:selectOneMenu value="#{form.country}" onchange="submit()">
   <f:valueChangeListener type="com.corejsf.CountryListener"/>
   <f:selectItems value="#{form.countryNames}"/>
</h:selectOneMenu>
```

```
<h:commandButton image="mountrushmore.jpg" action="#{rushmore.navigate}">
   <f:actionListener type="com.corejsf.RushmoreListener"/>
   <f:actionListener type="com.corejsf.ActionLogger"/>
</h:commandButton>
```

f:actionListener y f:valueChangeListener

- Con los tags, invocamos a una clase en lugar de a una acción
- Ésta debe implementar la interfaz ValueChangeListener o ActionListener

```
public class CountryListener implements ValueChangeListener {
   public void processValueChange(ValueChangeEvent event) {
      FacesContext context = FacesContext.getCurrentInstance();
      if ("ES".equals(event.getNewValue()))
         context.getViewRoot().setLocale(Locale.ES);
      else
         context.getViewRoot().setLocale(Locale.EN); }
}
```

f:setPropertyActionListener

- Hasta la especificación 1.2 de JSF, el paso de datos de la interfaz al componente era un tanto difícil.
- Con el tag f:setPropertyActionListener, invocamos al *setter* de nuestro Bean gestionado.

```
<h:commandLink immediate="true" action="#{localeChanger.changeLocale}">
   <f:setPropertyActionListener target="#{localeChanger.languageCode}" value="es"/>
   <h:graphicImage library="images" name="es_flag.gif" style="border: 0px"/>
</h:commandLink>

<h:commandLink immediate="true" action="#{localeChanger.changeLocale}">
   <f:setPropertyActionListener target="#{localeChanger.languageCode}" value="en"/>
   <h:graphicImage library="images" name="en_flag.gif" style="border: 0px"/>
</h:commandLink>
```

Paso de parámetros (bean gestionado)

```
public class LocaleChanger {

   private String languageCode;

   // Actions

   public String changeLocale() {
     FacesContext context = FacesContext.getCurrentInstance();
     String languageCode = getLanguageCode(context);
     context.getViewRoow().setLocale(new Locale(languageCode));
     return null;
   }

   // Setters

   public void setLanguageCode(String arg1) {
      languageCode = arg1;
   }
}
```

Eventos PhaseEvent

- Se producen antes y después de cada fase del ciclo de vida
- Se puede declarar un manejador propio en el fichero faces-config.xml:

```
<faces-config>
    <lifecycle>
        <phase-listener>es.ua.jtech.PhaseTracker</phase-listener>
    </lifecycle>
</faces-config>
```

- También, podemos declararlo en la raíz de la vista:

```
<f:phaseListener type="es.ua.jtech.PhaseTracker"/>
```

Eventos PhaseEvent

- Nuestros phase listeners deberán implementar la interfaz `javax.faces.event.PhaseListener`, que define los siguientes tres métodos:
  - `PhaseId getPhaseId()`. Dice a la implementación de JSF en qué fase enviar los eventos al listener. Estas fases pueden ser:

```
PhaseId.ANY_PHASE
PhaseId.APPLY_REQUEST_VALUES
PhaseId.INVOKE_APPLICATION
PhaseId.PROCESS_VALIDATIONS
PhaseId.RENDER_RESPONSE
PhaseId.RESTORE_VIEW
PhaseId.UPDATE_MODEL_VALUES
```

  - void `afterPhase(PhaseEvent)`
  - void `beforePhase (PhaseEvent)`

Eventos PhaseEvent

- También, podemos invocar beans que implementan métodos del tipo `void listener(javax.faces.event.PhaseEvent)`

```
<f:view beforePhase="#{backingBean.beforeListener}">
    ...
</f:view>
```

System Events

Clase del evento	Descripción	Origen
PostConstructApplicationEvent; PreDestroyApplicationEvent	Inmediatamente después del inicio de la aplicación; inmediatamente antes del apagado de la aplicación	Application
PostAddToViewEvent; PreRemoveFromViewEvent	Después de que un componente haya sido aladido al árbol de la vista; justo antes de que vaya a ser eliminado	UIComponent
PostRestoreStateEvent	Después de que el estado de un componente haya sido restaurado	UIComponent
PreValidateEvent; PostValidateEvent	Antes y después de que un componente haya sido validado	UIComponent
PreRenderViewEvent	Antes de que la vista raíz vaya a renderizarse	UIViewRoot
PreRenderComponentEvent	Antes de que vaya a renderizarse un componente	UIComponent
PostConstructViewMapEvent; PreDestroyViewMapEvent	Después de que el componente raíz ha construído el mapa de ámbito vista; cuando el mapa de la vista se limpia	UIViewRoot
PostConstructCustomScopeEvent; PreDestroyCustomScopeEvent	Tras la construcción de un ámbito de tipo *custom*; justo antes de su destrucción	ScopeContext
ExceptionQueuedEvent	Después de haber encolado una excepción	ExceptionQueuedEventContext

System Events - Invocación

- Mediante el tag f:event
  - Listening a nivel de componente o vista

```
<h:inputText value="#{...}">
        <f:event name="postValidate" listener="#{bean.method}"/>
    </h:inputText>
```

- El método del bean tendrá la forma `public void listener(ComponentSystemEvent) throws AbortProcessingException`

System Events - Invocación

- Mediante el anotaciones para clases del tipo UIComponent o Renderer.

```
@ListenerFor(systemEventClass=PreRenderViewEvent.class)
```

- Muy útil para el desarrollo de componentes

System Events - Invocación

- Declaración en el faces-config.xml

```
<application>
    <system-event-listener>
        <system-event-listener-class>listenerClass</system-event-listener-class>
        <system-event-class>eventClass</system-event-class>
    </system-event-listener>
</application>
```

- Muy útil para hacer *listening* a nivel de aplicación

System Events - Invocación

- Llamada al método subscribeToEvent de las clases UIComponent o Application.
- Muy útil para el desarrollo de *frameworks*

Validación mediante el tag f:event

```
<h:panelGrid id="date" columns="2">
    <f:event type="postValidate" listener="#{bb.validateDate}"/>
    Día: <h:inputText id="day" value="#{bb.day}" size="2" required="true"/>
    Mes: <h:inputText id="month" value="#{bb.month}" size="2" required="true"/>
    Año: <h:inputText id="year" value="#{bb.year}" size="4" required="true"/>
</h:panelGrid>
<h:message for="date" styleClass="errorMessage"/>
```

Validación mediante el tag f:event

```
public void validateDate(ComponentSystemEvent event) {
    UIComponent source = event.getComponent();
    UIInput dayInput = (UIInput) source.findComponent("day");
    UIInput monthInput = (UIInput) source.findComponent("month");
    UIInput yearInput = (UIInput) source.findComponent("year");

    int d = ((Integer) dayInput.getLocalValue()).intValue();
    int m = ((Integer) monthInput.getLocalValue()).intValue();
    int y = ((Integer) yearInput.getLocalValue()).intValue();

    if (!isValidDate(d, m, y)) {
        FacesMessage message = es.ua.jtech.util.Messages.getMessage(
          "es.ua.jtech.messages", "invalidDate", null);
        message.setSeverity(FacesMessage.SEVERITY_ERROR);
        FacesContext context = FacesContext.getCurrentInstance();
        context.addMessage(source.getClientId(), message);
        context.renderResponse();
    }
}
```

Toma de decisiones

```
<f:view>
    <f:event type="preRenderView" listener="#{user.checkLogin}"/>
    <h:head>
        <title>...</title>
    </h:head>
    <h:body>
        ...
    </h:body>
</f:view>
```

Toma de decisiones

```
public void checkLogin(ComponentSystemEvent event) {
    if (!loggedIn) {
        FacesContext context = FacesContext.getCurrentInstance();
        ConfigurableNavigationHandler handler =
          (ConfigurableNavigationHandler)context.getApplication().getNavigationHandler();
        handler.performNavigation("login");
    }
}
```

¿Preguntas?