Escuela Politécnica de Ingeniería
Grado de Ingeniería Informática en Tecnologías de la Información

Tecnologías Web

Tema 3

Tecnologías web de servidor: JSF 2.2 2/2

Índice

Archivos de propiedades e internacionalización i18N

- Archivo de etiquetas: archivos de texto plano con cadenas que puede ser referenciadas mediante etiquetas en JSF.
- El fin es:
  - Reutilizar cadenas. (“Nombre”, “Name”, “Apellidos”, “Surname” …)
  - Formato (archivo.properties)
  - Pares: Etiqueta=Valor
  - Ubicación: deben ir en src/…
  - Declaración del archivo **src/messages.properties**

```
<application>
 <resource-bundle>
  <base-name>messages</base-name>
   <var>msgs</var>
   </resource-bundle>
</application>
```

Ejemplo

```
registrationTitle=Registration
registrationText=Please enter your first name, last name, and email address.
firstNamePrompt=Enter first name
lastNamePrompt=Enter last name
emailAddressPrompt=Enter email address
buttonLabel=Register Me
successTitle=Success
successText=You registered successfully.
```

- En despliegue el archivo va en .../WEB-INF/classes/messages1.properties

```
<?xml version="1.0"?>
<faces-config ...>
<application>
<resource-bundle>
<base-name>messages1</base-name> <var>msgs1</var> </resource-bundle>
</application> </faces-config>
```

- Uso: #{msgs1.firstNamePrompt}

Ejemplo de uso

```
<!DOCTYPE ... >
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://java.sun.com/jsf/html">
<h:head><title>#{msgs1.registrationTitle}</title>
 <link href="./css/styles.css" rel="stylesheet" type="text/css"/>
 </h:head>
<h:body>
<table border="5" align="center">
<tr>
<th class="title">#{msgs1.registrationTitle}</th>
</tr>
</table>
<h3>#{msgs1.registrationText}</h3>
<h:form> #{msgs1.firstNamePrompt}: <h:inputText value="#{person1.firstName}"/> <br/>
#{msgs1.lastNamePrompt}: <h:inputText value="#{person1.lastName}"/> <br/>
#{msgs1.emailAddressPrompt}: <h:inputText value="#{person1.emailAddress}"/> <br/>
<h:commandButton value="#{msgs1.buttonLabel}"  action="#{person1.doRegistration}"/>
</h:form>
</h:body></html>
```

Strings parametrizados

- Más flexibles que los archivos de propiedades convencionales
- Permiten modificar en tiempo de ejecución los valores de las etiquetas
- Formato:
  - P.e., someName=blah {0} blah {1}
    Declaración igual que el ejemplo anterio, salvo que la etiqueta ahora es un Map.
- Mensajes de salida empleando h:outputFormat
  - <f:param> permite dar valores de sustitución a las etiquetas

```
<h:outputFormat value="#{msgs.someName}">
<f:param value="Literal value for 0th entry"/>
<f:param value="#{someBean.calculatedValForEnty1}"/>
</h:outputFormat>
```

Ejemplo de archivo de propiedades

registrationTitle=Registration

firstName=First Name

lastName=Last Name

emailAddress=Email Address

registrationText=Please Enter Your {0}, {1}, and {2}.

prompt=Enter {0}

buttonLabel=Register Me

successTitle=Success

successText=You Registered Successfully.

Ejemplo de uso

```
<!DOCTYPE ...>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:f="http://java.sun.com/jsf/core" xmlns:h="http://java.sun.com/jsf/html">
<h:head><title>#{msgs2.registrationTitle}</title>
... <h3>
<h:outputFormat value="#{msgs2.registrationText}">
<f:param value="#{msgs2.firstName}"/> Replaces {0} in registrationText
<f:param value="#{msgs2.lastName}"/> Replaces {1} in registrationText
<f:param value="#{msgs2.emailAddress}"/> Replaces {2} in registrationText
</h:outputFormat>
```

Ejemplo de uso 2

```
<!DOCTYPE ...>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://java.sun.com/jsf/html"> <h:head><title>#{msgs2.successTitle}</title> ...
</h:head>
<h:body>
<table border="5" align="center">
<tr><th class="title">#{msgs2.successTitle}</th></tr> </table>
<h3>#{msgs2.successText}</h3> <ul>
<li>#{msgs2.firstName}: #{person2.firstName}</li>
<li>#{msgs2.lastName}: #{person2.lastName}</li>
<li>#{msgs2.emailAddress}: #{person2.emailAddress}</li>
</ul>
</h:body>
</html>
```

Internacionalización (i18N)

- El propósito es web multilungua
- La idea es un archivos de propiedades para cada idioma (msg.properties, msg_es.properties, ….)
- Ante colisiones el último alfabéticamente gana
- El idioma lo escoge el usuario en su navegador

¿Cómo se hace?

- Crear un archivo de propiedades por idioma con las mismas etiquetas (blah.properties, blah_es.properties, blah_es_mx.properties)
- **Emplear f:view y el atributo locale**

  <f:view locale="#{facesContext.externalContext.requestLocale}">
  - Determina locale a partir de la configuración del navegador
  - O mediante código (uso de eventos)
  - Declarar el recurso **resource-bundle**
  - **El archivo de propiedades se selecciona automáticamente basado en locale**
- **La salida se hace:**
  - **h:outputFormat**
  - **Con el lenguaje de expresiones (#{msgs.etiqueta})**

Ejemplo

- **messages.properties**
  - company=JsfResort.com
  - feature=Our {0}:
  - pool=swimming pool
- **messages_es.properties**
  - feature=Nuestra {0}:
  - pool=piscina
- **messages_es_mx.properties**
  - pool=alberca

Ejemplo

```
<?xml version="1.0"?>
<faces-config ...>
<application>
<resource-bundle>
<base-name>messages</base-name> <var>msgs</var>
</resource-bundle>
</application>
</faces-config>
```

Ejemplo

```
<!DOCTYPE ...>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:f="http://java.sun.com/jsf/core" xmlns:h="http://java.sun.com/jsf/html">
<f:view locale="#{facesContext.externalContext.requestLocale}">
...
<h1>#{msgs.company}</h1>
 <h2>
<h:outputFormat value="#{msgs.feature}">
 <f:param value="#{msgs.pool}"/>
</h:outputFormat>
</h2>
<img src="fancy-swimming-pool.jpg".../> ...
 </f:view>
</html>
```

Configurar locale (i18N) a partir de de la configuración de un usuario

- Dos pasos:
  - Invocar setLocale de view
    - <f:view locale="#{formSettings.currentLocale}">
    - También se puede obtener de la configuración del browser:
      - <f:view locale="#{facesContext.externalContext.requestLocale}">
  - Escribir locale de UIViewRoot
    - FacesContext.getCurrentInstance().getViewRoot() .setLocale(currentLocale);

Problema del uso de UIViewRoot().setLocale

- Cuando se llama a setLocale en view
  - Se invoca cada vez que se recarga la página jsf
  - No se invoca cuando se revisualiza la página despues de ejecutar el actionlistener (o envío de formulario)
- Si se obtiene Locale con UIViewRoot
  - Se invoca a setLocale cuando se revisualiza la página despues de ejecutar el actionlistener (o envío de formulario)
  - No se invoca cada vez que se recarga la página jsf
    - Ya que Locale se fija al valor por defecto
- Solución
  - Antes de navegar a la página ejecutar: FacesContext.getCurrentInstance().getViewRoot().setLocale(currentLocale);
  - En la propia página

```
<f:view locale="#{formSettings.currentLocale}">
```

Ejemplo Input Form

```
<!DOCTYPE ...>
 <html ...>
<f:view locale="#{formSettings.locale}"> ...
<h:commandButton value="#{msgs.switchLanguage}"
actionListener="#{formSettings.swapLocale1}"
immediate="true"/> ...
</f:view>
</html>
```

Bean para f:view

```
@ManagedBean
@SessionScoped
   public class FormSettings implements Serializable {
   private boolean isEnglish = true;
   private static final Locale ENGLISH = new Locale("en");
   private static final Locale SPANISH = new Locale("es");
   private Locale locale = new Locale("en");
   public Locale getLocale()
   { return(locale);
   }
```

...

Bean FormSettings para actionListener

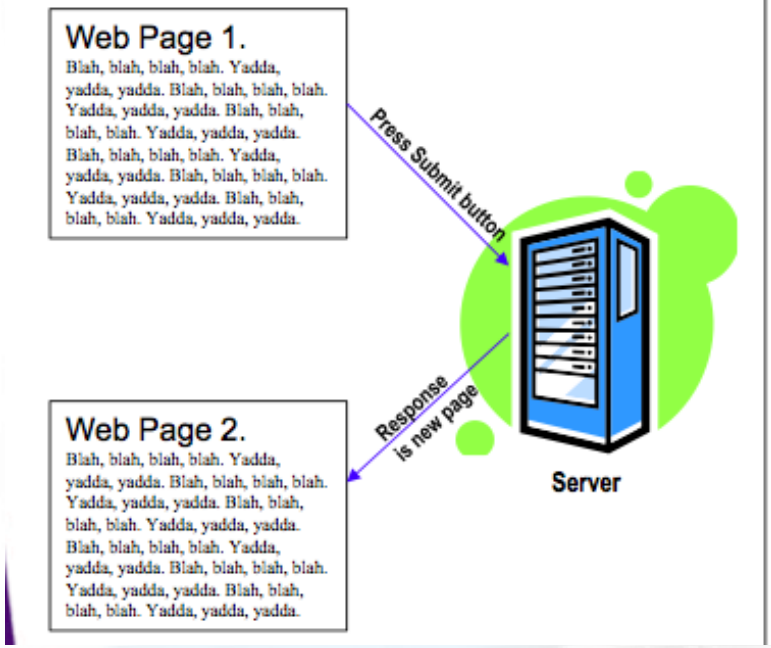
```
public void swapLocale(ActionEvent event)
{
   isEnglish = !isEnglish;
    if (isEnglish)
          locale = ENGLISH;
    else
          locale = SPANISH;
FacesContext.getCurrentInstance().getViewRoot() .setLoc
ale(locale);
 }
```

¿Qué es AjAX?

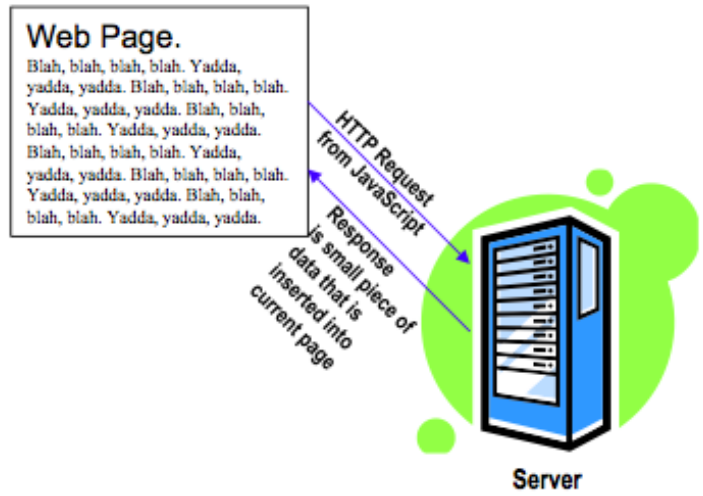
- El problema
  - Sincronicidad: Las solicitudes de una petición HTTP “convencional” suelen acarrear esperas.
  - La limitacion de opciones en los widget gráficos
- Contenido activo de cliente
  - Fallido: Java Applets
    - No soportado universalmente y no puede interactuar con HTML
  - Alternativa seria: Flash/Flex
    - No preinstalada en todos los PCS;
    - No disponible para iPhone/iPad
  - Poco probados
    - Microsoft Silverlight
    - JavaFX

AJAX = Asynchronous JavaScript And XML

Aplicaciones Web tradicinales
Cambios grandes infrecuentes



Aplicaciones Web AjaX
Cambios pequeños frecuentes



¿Porqué AJAX en JSF?

- ¿Porqué una librería de Ajax específica para JSF?
  - Existe decenas de ellas en el mercado (jQuery, DWR, GWT, etc).
- **Ventajas de una librería Ajax en JSF**
- Client side
  - Puedes configurar los elementos *JSF* (h:outputText, h:inputText, h:selectOneMenu, etc.)
  - Sería inmanejable si las actualizaciones Ajax fuera por un lado y los elementos gráficos por otro
  - No es necesario escribir código JavaScript
- Server side
  - Las llamadas Ajax puede ver los beans gestionados
    - incluyendo getters/setters de las propiedades
  - No hay que crear servlets y parsear los parámetros

Manejo de eventos vs. Ajax

- Ajax: ***A****synchronous* ***J****avaScript* ***A****nd* ***X****ML*
- La gestión de eventos en JSF es su fuerte comparado a Struts2 y MVC convencional
- Ajax suele ser más adecuado en más situaciones que los manejadores de eventos
- Ajax confiere una mejor experiencia al usuario
- Manejadores de eventos adecuado cuando se envia la página entera
- Ajax cuando se envía solo parte de la página

Formulario simple

- **Sintaxis**

```
<h:commandButton ... action="...">
   <f:ajax render="id1"/>
</h:commandButton>
<h:outputText ... value="#{...}" id="id1"/>
```

- **Interpretación**
  - Cuando se pincha el botón se ejecuta la action en el servidor se calcula el valor del elemento etiquetado como id1 (el setter de la propiedad referenciada en value=) y se retorna el valor al cliente sustituyendo el elemento DOM con el nuevo valor
  - Si el atributo “value” resulta un nuevo valor cada vez, entonces la action no necesita valor

```
<h:commandButton value="Update Time" action="nothing"> <f:ajax
render="timeResult"/>
</h:commandButton>
<h:outputText value="#{dateBean.time}" id="timeResult"/>
```

Formulario general

- **Sintaxis**

```
<h:commandButton ... action="...">
<f:ajax render="id1 id2" execute="id3 id4"
event="blah" onevent="javaScriptHandler"/> </h:commandButton>
```

- **Atributos**
  - Render
    - Los elementos a repintar en la página. Normalmente h:outputText
    - El objetivo de render debe estar en el mismo f:form que f:ajax
  - execute
    - Los elemento a enviar al servidor para ser procesados. Generalmente inputs elements como h:inputText or h:selectOneMenu.
  - event
    - El evento DOM al que responder (p.e., keyup, blur)
  - onevent
    - Una función JavaScript que se ejecuta cuando se dispara el evento (event)

Estructura de un página Facelets con Ajax

- **Declarar el f:namespace**
  - En la página xhtml incluir

    xmlns:f="http://java.sun.com/jsf/core
- Usar **h:head**
  - Cuando se usa f:ajax, el sistema inserta la etiqueta <script> en la sección <head>. Y no podrá encontrar la sección head a no ser que incluyas h:head
  - Es buena práctica usar siempre h:head and h:body.
- Si el browser tiene deshabilitado JavaScript, Los botones Ajax se convertirán en botones no-Ajax con envios de formulario y revisualización de la misma página.

Signatura de los action controllers

- **No-Ajax**

  public String myActionControllerMethod() {

  ...

  return("some outcome"); }

- • **Ajax obligatorio**

  public void myActionControllerMethod() { ... }

- • **Ajax y No-Ajax**

  public String myActionControllerMethod() {

  ...

  return(null); // In non-Ajax apps, means to redisplay form }

  – Este método valdrá para formularios Ajax y No-Ajax

El atributo render de f:ajax

- **Sintaxis**
  <f:ajax render="elementId" ... />
- **Idea**
  - Id o lista de Ids separados por espacios cuyos valores debe retornarse desde el servidor y reemplacerse en el arbol DOM
  - Estos elementos deben estar en el mismo h:form que f:ajax
- **Details**
  - Hay 4 valores especiales: @this, @form, @none, and @all.
  - Los valores de render (execute y event) puede ser expresiones JSF en vez de literales.

Version No-Ajax vs Ajax

Versión NO-AJAX

```
<h:form>
<fieldset>
<legend>Random Number</legend>
<h:commandButton value="Show Number"
action="#{numberGenerator.randomize}”>
</h:commandButton><br/>
<h2>
<h:outputText
value="#{numberGenerator.number}”/>
</h2>
</fieldset>
</h:form>
```

Versión AJAX

```
<h:form>
<fieldset>
<legend>Random Number</legend>
<h:commandButton value="Show Number"
action="#{numberGenerator.randomize}”>
<f:ajax render="numField1"/>
</h:commandButton><br/>
<h2><h:outputText
value="#{numberGenerator.number}"
id=“numField1” /></h2>
</fieldset>
</h:form>
```

- V. Ajax: cuando se presiona el botón, Javascript realiza una solicitud al servidor sin enviar el formulario. En el servidor se obtiene el bean numerGenerator (scope session) y se ejecuta el método ramdomize y a continuación se calcula el médodo getNumer. Se envía el valor al cliente y se inserta en el componente DOM correspondiente (h:outputText).
- Hay que tener en cuenta que el elemento a actualizar (el target de render) debe de estar en el mismo h:form al que se refiere la etiqueta f:ajax.
- Si Javascript estuviera deshabilitado en el navegador, se obviarían los elemento ajax y se ejecutaría el formulario en la versión NO-Ajax.

Código del M. Bean

```
@ManagedBean
public class NumberGenerator {
private double number = Math.random();
private double range = 1.0;
public double getRange() { return(range); }
public void setRange(double range) { this.range = range; }
public double getNumber() { return(number); }
//Si randomize se usara sólo en Ajax no necesitaría retornar nada (void)
public void randomize() { ... }
//pero es mejor retornar null para que se válido para invocaciones No-Ajax
public String randomize()
 { number = range * Math.random(); return(null); }
}
```

El atributo excute de f:ajax

- **Sintaxis**
  - <f:ajax render="..." execute="..." ... />
- **Idea**
  - Definir un ID o conjuntos de ID a **enviar** al servidor
    - h:inputText se procesa normalmente (setters, validation, etc.)
- **Detalles**
  - 4 valores especiales: @this, @form, @none, @all
  - @this. Se envía el elemento indicado en f:ajax. Por defecto.
  - @form. Se envía todos los elementos del formulario en el que aparece f:ajax. Adecuado si hay que enviar todos los campos.
  - @none. No se envía nada. Útil si el elemento a pintar modifica su valor cada vez que es evaluado.
  - @all. Todos los elemento JSF UI en la página.

Ejemplo de execute

Fágina Facelet

```
<h:form> <fieldset>
<legend>Random Number (with execute="someId")</legend> Range:
<h:inputText value="#{numberGenerator.range}"
id="rangeField"/>
<br/>
<h:commandButton value="Show Number" action="#{numberGenerator.randomize}"> <f:ajax execute="rangeField" render="numField3"/>
 </h:commandButton><br/> <h2><h:outputText value="#{numberGenerator.number}"
id="numField3"/>
</h2>
</fieldset>
</h:form>
```

Bean

```
@ManagedBean
public class NumberGenerator {
private double number = Math.random(); private double range = 1.0;
public double getRange() { return(range); }
public void setRange(double range) { this.range = range;
}
public double getNumber() { return(number); }
public String randomize() { number = range * Math.random(); return(null);
} }
```

En este caso tenemos un atributo que se trae desde el servidor (render="numField3") y otro que se envía al servidor (execute="rangeField"). Este último supone la ejecución del método setRange antes de que se ejecutado el action controller randomize.

Uso de execute="@form"

- **Sintaxis**

  <f:form>

  <f:ajax render="Id" execute="@form"/>

  <h:form/>

- **Idea**
  - Envía todos los elementos del formulario al servidor
    - Se procesa normalmente todos los elementos JSF (setters, validación, etc.)
- **Detalles**
  - Útil si no quieres listar todos los campos de entrada en el atributo render de f:ajax. Tampoco es necesario indicar los ids en los campos de entrada del formulario.

Ejemplo

```
<h:form>
<fieldset>
<legend>Bank Customer Lookup (with execute="@form")
</legend>
 Customer ID:
<h:inputText value="#{bankingBeanAjax.customerId}"/><br/>
 Password:
<h:inputSecret value="#{bankingBeanAjax.password}"/><br/>
<h:commandButton value="Show Current Balance"
action="#{bankingBeanAjax.showBalance}">
<f:ajax execute="@form" render="ajaxMessage1"/>
</h:commandButton>
<br/>
<h2>
<h:outputText value="#{bankingBeanAjax.message}"
</fieldset>
</h:form>
```

Uso del atributo event de f:ajax

- **Sintaxis**

  <f:ajax render="..." event="..." ... />
- **Idea**
  - Especifica el nombre del evento DOM al que responder. No se debe incluir “on”.
    - http://www.w3schools.com/jsref/dom_obj_event.asp
- **Detalles**
  - Sino se especifica se usa el evento por defecto.
  - Eventos de alto nivel
    - JSF añade 2 extras: action & valueChange.
  - Envolver elementos con f:ajax
    - <f:ajax render="...">a bunch of components</f:ajax>

Eventos por defecto

- **Action**
  - h:commandButton, h:commandLink
  - "action" es parte de JSF, y no nativo de JavaScript/DOM. Lo cual significa que el botón/enlace se invoca de muchas formas (pulsando en él, INTRO si tiene el foco, acelerador de teclado, etc…)
- **valueChange**
  - h:inputText, h:inputSecret, h:inputTextarea, all radio button, checkbox, & menu items (h:selectOneMenu, etc.)
    - También este evento es suministrado por JSF y no es nativo de DOM. Diferentes navegadores manejan el cambio de un campo de forma diferente, de modo que este evento unifica el comportamiento

Ejemplo: Cambio de temperatura

Facelet

```
<h:form> <fieldset> <legend>On-the-Fly
Temperature Converter ...</legend>
Temperature in Fahrenheit:
<h:inputText
value="#{temperatureConverter.fTemp}">
<f:ajax event="keyup" render="cField
kField"/>
</h:inputText><br/> <h2> Temperature in
Celsius:
<h:outputText
value="#{temperatureConverter.cTemp}"
id="cField"/><br/>
Temperature in Kelvin: <h:outputText
value="#{temperatureConverter.kTemp}"
id="kField"/><br/>
</h2> </fieldset></h:form>
```

M.Bean

```
@ManagedBean
public class TemperatureConverter {
private String cTemp, kTemp; public String
getcTemp()
{ return(cTemp); }
public String getkTemp() { return(kTemp); }
public String getfTemp()
{ return(""); }
```

Ejemplo: combos encadenados 1

```
<h:form>
<fieldset> <legend>Population Lookup (with chained comboboxes)</legend>
1. State: <h:selectOneMenu
   value="#{locationBean.state}">
 <f:selectItems value="#{locationBean.states}"/>
<f:ajax render="cityList"/>
</h:selectOneMenu>
 <br/>
2. City: <h:selectOneMenu
   value="#{locationBean.city}"
   disabled="#{locationBean.cityListDisabled}"
   id="cityList">
<f:selectItems value="#{locationBean.cities}"/>
<f:ajax render="population"/>
</h:selectOneMenu> <br/>
3. Population: <h:outputText
   value="#{locationBean.city}" id="population"/>
   </fieldset>
</h:form>
@ManagedBean
@SessionScoped
public class LocationBean implements Serializable {
private String state, city;
private boolean isCityListDisabled = true;
public String getState() { return (state); } Make city list disabled (grayed out) initially. Enable it when the state is selected.
public void setState(String state) { this.state = state;
isCityListDisabled = false;
}
public String getCity() { return(city); }
public void setCity(String city) { this.city = city; }
public boolean isCityListDisabled()
{ return(isCityListDisabled);
}
```

Ejemplo: combos encadenados 2

```
public List<SelectItem> getStates() {
List<SelectItem> states = new
ArrayList<SelectItem>();
states.add(new SelectItem("--- Select State
---"));
for(StateInfo stateData:
StateInfo.getNearbyStates()) {
states.add(new
SelectItem(stateData.getStateName()));
}
return(states); }
```

```
public SelectItem[] getCities() { SelectItem[]
cities = { new SelectItem("--- Choose City
---")}; if(!isCityListDisabled && (state != null))
{
for(StateInfo stateData:
StateInfo.getNearbyStates())
{ if(state.equals(stateData.getStateName())) {
cities = stateData.getCities();
break; }
} }
return(cities); } }
```

Validación de la entrada de usuario

- Tipos de validación
  - Validation manual
    - Validación en el action controller
  - Validación manual implícita
    - Conversión de tipos y el atributo “required”
  - Validación automática explícita
    - Uso de f:validateLength, f:validateRegex, etc.
  - Definición de métodos de validación propios
    - Uso del atributo “validator”

La necesidad de la validación

- Dos tareas que una aplicación Web necesita realizar:
  - Comprobar que todos los campos obligatorios están rellenados y en el formato adecuado
  - Revisualizar el formulario cuando los campos están mal formados y no están
    - Mostrando los mensajes de error oportunos
    - Y mantiendo los valores válidos en formulario
- Debilidad de JSP 2.0

Aproximaciones de validación

- **Validación manual**
  - Emplear propiedades string en el bean
  - Realizar la validación en los métodos setter y/o en el action controller
  - Retornar null para revisualizar el formulario
  - Crear mensajes de error personalizados y almacenarlos en FacesMessage
  - Usar h:messages para mostrar la lista de errores
- **Validación automática implícita**
  - Usar propiedades del bean int, double, etc. o añadir required. –
  - El sistema revisualiza el formulario si hay errores de conversión
  - Usar h:message para mostrar el error específico
- **Validación automática específica**
  - Usar f:validateLength, f:validateDoubleRange, f:validateLongRange o f:validateRegex
  - El sistema revisualiza el formulario si hay fallos; Usar h:message
- **Métodos de validación propios**
  - Crear FacesMessage envuelto en ValidatorException

Validación manual

- Los métodos setter **sólo aceptan strings**
  - Realizar la conversión explícita a los otros tipos en código
  - Usar bloques try/catch para manejar datos ilegales
  - Puede requerir lógica de negocio específica
- **El action controller checkea los valores**
  - Si los valores son válidos
    - Retornará salidas normales
  - Si los valores son inválidos o están vacíos
    - Almacenar los mensajes de error empleando FacesContext.addMessage
    - Retornar null para revisualizar el formulario
- El formulario de entrada mostrará los mensajes de error
  - Usar h:messages
    - Si no hay mensajes no se generará salida alguna

Ejemplo de validación manual

```
<h:form>
<h:messages styleClass="error"/>
<h:panelGrid columns="2" styleClass="formTable">
  User ID:
  <h:inputText value="#{bidBean1.userID}"/>

  Keyword:
  <h:inputText value="#{bidBean1.keyword}"/>

  Bid Amount: $
  <h:inputText value="#{bidBean1.bidAmount}"/>

  Duration:
  <h:inputText value="#{bidBean1.bidDuration}"/>
</h:panelGrid>
  <h:commandButton value="Send Bid!"
                   action="#{bidBean1.doBid}"/>
</h:form>
```

Ejemplo de validación manual 2

```
@ManagedBean
public class BidBean1 {
  private String userID = "";
  private String keyword = "";
  private String bidAmount;
  private double numericBidAmount = 0;
  private String bidDuration;
  private int numericBidDuration = 0;
  public String getUserID() { return(userID); }
   public void setUserID(String userID      {
   this.userID = userID.trim();
  }
  public String getKeyword() { return(keyword); }

  ublic void setKeyword(String keyword) {
   this.keyword = keyword.trim();
  }
  public String getBidAmount()
{ return(bidAmount); }

  public void setBidAmount(String bidAmount) {
   this.bidAmount = bidAmount;
    try {
     numericBidAmount =
Double.parseDouble(bidAmount);
    } catch(NumberFormatException nfe) {}
  }
  public double getNumericBidAmount() {
   return(numericBidAmount);
  }
  public String getBidDuration()
{ return(bidDuration); }
  public void setBidDuration(String bidDuration) {
   this.bidDuration = bidDuration;
   try {
     numericBidDuration =
Integer.parseInt(bidDuration);
   } catch(NumberFormatException nfe) {}
  }

  public int getNumericBidDuration() {
   return(numericBidDuration);
  }
```

Ejemplo de validación manual 3

```
public String doBid() {
  FacesContext context =
FacesContext.getCurrentInstance();
  if (getUserID().equals("")) {
    context.addMessage(null,
      new FacesMessage("UserID required"));
  }
  if (getKeyword().equals("")) {
    context.addMessage(null,
      new FacesMessage("Keyword required"));
  }
  if (getNumericBidAmount() <= 0.10) {
    context.addMessage(null,
      new FacesMessage("Bid amount must be at
least $0.10."));
  }
  if (getNumericBidDuration() < 15) {
    context.addMessage(null,
      new FacesMessage("Duration must be at least
15 days."));
  }
  if (context.getMessageList().size() > 0) {
    return(null);
  } else {
    doBusinessLogicForValidData();
    return("show-bid1");
  }
}
```

Ejemplo de validación manual 4 (Mostrando los errores)

```
<h:form>
<h:messages styleClass="error"/>
<h:panelGrid columns="2" styleClass="formTable">
User ID: <h:inputText value="#{bidBean1.userID}"/>
Keyword: <h:inputText value="#{bidBean1.keyword}"/>
Bid Amount: $ <h:inputText value="#{bidBean1.bidAmount}"/>
Duration: <h:inputText value="#{bidBean1.bidDuration}"/>
</h:panelGrid>
<h:commandButton value="Send Bid!"
action="#{bidBean1.doBid}"/>
</h:form>
```

Validación automática implícita

- Definir las propiedades de los bean como tipos simples.
  - int/Integer, long/Long, double/Double, boolean/Boolean, etc.
  - Los tipos envoltorio permiten valores vacíos
- El sistema intenta realizar la conversión automática
  - Se convierten al estilo jsp:setProperty
    - i.e. Integer.parseInt,Double.parseDouble,etc.
  - Si hay error se revisualiza el formulario
    - Los mensajes de error se almacenan automáticamente
    - Usa el atributo converterMessage para personalizar los mensajes de error
- Campos obligatorios
  - Se puede añadir el atributo **required** a cualquier campo de entrada
- Usa h:message para mostrar los mensajes
  - Retorna cadena vacía si no hay mensaje

Precedencia de los tests de validación

- **Required**
  - Si etiquetas un campo como “required” y no se rellena el campo se genera el error de campo obligatorio
  - Tipos/validators no checkeados
    – Warning: Si tu setter espera un String un espacio en blanco lo satisface. i.e. el espacio en blanco no es considerado vacío.
- **Tipos**
  - Si un campo pasa la validación “required”. Entonces JSF comprueba si la cadena puede convertise al tipo esperado.
- **Validators**
  - Si un campo pasa required/tipos entonces se comprueba cualquier validador explícito.
- **Saltarse la validación**
  - Si se usa immediate="true” en un botón se salta la validación

Ejemplo de validación automática implícita 1

```
<h:form> <table> <tr> <td>
User ID: <h:inputText
value="#{bidBean2.userID}"
required="true"
requiredMessage="You must enter a user ID"
id="userID"/></td>
<td><h:message for="userID"
styleClass="error"/></td></tr> <tr>
<td>
Keyword:
<h:inputText value="#{bidBean2.keyword}"
required="true"
requiredMessage="You must enter a
keyword" id="keyword"/></td>
<td><h:message for="keyword"
styleClass="error"/></td></tr>
<tr> <td>Bid Amount: $<h:inputText
value="#{bidBean2.bidAmount}"
required="true"
requiredMessage="You must enter an
amount"
converterMessage="Amount must be a
number" id="amount"/></td>
<td><h:message for="amount"
styleClass="error"/></td></tr>
<tr> <td>
Duration: <h:inputText
value="#{bidBean2.bidDuration}"
required="true"
requiredMessage="You must enter a
duration"
converterMessage="Duration must be a
whole number"
id="duration"/></td>
<td><h:message for="duration"
styleClass="error"/></td></tr>
<tr><th colspan="2">
<h:commandButton value="Send Bid!"
action="#{bidBean2.doBid}"/></th></tr>
</table> </h:form>
```

Ejemplo de validación automática implícita 2

```
@ManagedBean public class BidBean2
 {
private String userID;
 private String keyword;
private Double bidAmount;
private Integer bidDuration; ...
public Double getBidAmount()
 { return(bidAmount); }
public void setBidAmount(Double bidAmount)
{ this.bidAmount = bidAmount; }
public Integer getBidDuration() { return(bidDuration); }
public void setBidDuration(Integer bidDuration) {
this.bidDuration = bidDuration; }
public String doBid()
 {
  doBusinessLogicForValidData();
 return("show-bid2");
}
```

Validación automática explícita

- Definir las propiedades del bean con tipos simples
  - int/Integer, double/Double, boolean/Boolean, etc.
- Agregar f:validateBlah or f:convertBlah
  - <f:validateLength .../>
  - <f:validateLongRange .../>
  - <f:validateDoubleRange .../>
  - <f:validateRegex .../>
- El sistema checkea la validez
  - Después de comprobar los required (requiredMessage) y los tipos (converterMessage), comprueba si pasa la regla del validador
  - Sino la pasa almacena el mensaje de error y revisualiza el formulario
    - Usar el atributo validatorMessage para personalizar los mensajes de error

Sintaxis general de un validador

- Regla: tener entre 5 y 6 caracteres

```
<h:inputText value="#{someBean.someProperty}"
  required=true
  requiredMessage="…"
  converterMessage="…"
  validatorMessage="…"
  id="someID"/>
<f:validateLength minimum="5" maximum="6"/>
</h:inputText>
```

- Precedencia de los tests
  - Required
  - Conversión de tipos
  - Reglas de validación

Conversión vs. Validación

- Tanto f:convertBlah como f:validateBlah checkean el formato y revisualizan el formulario si algún campo es inválido
  - Pero f:convertBlah hace alguna cosa más:
    - Puede cambiar el formato en que se muestra un campo
    - Puede cambiar la forma en que un String se convierte a número
  - f:convertBlah importante para outputText
- Ejemplos

```
<h:outputText value="#{orderBean.discountCode}">
<f:convertNumber maxFractionDigits="2"/>
</h:outputText>
```

  - Muestra 0.75 or algo muy próximo pero (0.749)

```
<h:outputText value="#{orderBean.discountCode}">
 <f:convertNumber type="percentage"/>
</h:inputText>
```

  - Puedes meter “75%”, pero almacena 0.75

Principales atributos de validación y conversión

- **f:validateLength**
  - minimum
  - maximum
- **f:validateLongRange**
  - minimum
  - maximum
- **f:validateDoubleRange**
  - minimum
  - Maximum
- **f:validateRegex**
  - pattern
- **f:convertNumber**
  - currencyCode, currencySymbol
  - groupingUsed
  - integerOnly
  - locale
  - max(min)FractionDigits
  - max(min)IntegerDigits
  - pattern (ala DecimalFormat)
  - type
    - number, currency, percentage
- **f:convertDateTime**
  - type
    - date, time, both
  - dateStyle, timeStyle
    - default, short, medium, long, full
  - pattern (ala SimpleDateFormat)
  - Locale
  - timeZone

Ejemplo de validación aut. Explícita

```
<h:panelGrid columns="3"
styleClass="formTable">
User ID: <h:inputText
value="#{bidBean2.userID}"
required="true" requiredMessage="You
must enter a user ID"
validatorMessage="ID must be 5 or 6
chars" id="userID"> <f:validateLength
minimum="5" maximum="6"/>
</h:inputText>
<h:message for="userID"
styleClass="error"/>
Keyword:
<h:inputText
value="#{bidBean2.keyword}"
required="true"
requiredMessage="You must enter a
keyword" validatorMessage="Keyword
must be at least 3 chars" id="keyword">
<f:validateLength minimum="3"/>
</h:inputText>
<h:message for="keyword"
styleClass="error"/>
Bid Amount: $
<h:inputText
value="#{bidBean2.bidAmount}"
required="true"
requiredMessage="You must enter an
amount" converterMessage="Amount
must be a number"
validatorMessage="Amount must be
0.10 or greater" id="amount">
<f:validateDoubleRange
minimum="0.10"/>
</h:inputText>
<h:message for="amount"
styleClass="error"/>
```

Ejemplo de validación aut. Explícita 2

```
Duration:
<h:inputText value="#{bidBean2.bidDuration}"
required="true"
requiredMessage="You must enter a duration" converterMessage="Duration must be a whole number" validatorMessage="Duration must be 15 days or more" id="duration">
<f:validateLongRange minimum="15"/>
</h:inputText> <h:message for="duration" styleClass="error"/> </h:panelGrid>
<h:commandButton value="Send Bid!" action="#{bidBean2.doBid}"/>
</h:form>
```

Validación empleando método personalizados

- **Facelets**
  En los elementos input, indicar el método de validación esplícitamente
  - <h:inputText id="someID" validator="#{someBean.someMethod}"/>
- **Usar h:message**
  - <h:message for="someID"/>
- **Java**
  - Elevar una excepción ValidatorException con un FacesMessage si falla la validación. En caso contrario no hacer nada
- **Argumentos del método de validación:**
  - FacesContext (el contexto)
  - UIComponent (El componente a validar)
  - Object (el valor enviado, los tipos primitivos usan envoltorios)
- **Warning**
  - No emplear f:validateBlah & método personalizado para el mismo campo

Ejemplo con métodos personalizados

```
Bid Amount: $
<h:inputText
value="#{bidBean2.bidAmount}"
required="true"
requiredMessage="You must enter an
amount" converterMessage="Amount
must be a number"
validator="#{bidBean2.validateBidAmou
nt}" id="amount"/>
<h:message for="amount"
styleClass="error"/>
```

```
public void
validateBidAmount(FacesContext
context, UIComponent
componentToValidate,
Object value) throws ValidatorException
{
double bidAmount =
((Double)value).doubleValue(); double
previousHighestBid =
currentHighestBid();
 if (bidAmount <= previousHighestBid) {
FacesMessage message =
new FacesMessage("Bid must be higher
than current " +
"highest bid ($" + previousHighestBid +
").");
throw new
ValidatorException(message);
} }
```

Estructuras iterativas: ui: repeat

- **Sintaxis**

```
<ui:repeat var="someVar" value="#{someBean.someCollection}">
<someHTML>
#{someVar.someProperty} </someHTML>
</ui:repeat>
```

- **En Java**

```
for(SomeType someVar: someCollection) {
doSomethingWith(someVar);
}
```

- **Warnings**
  - **Tipos admitidos:** array o List (or ResultSet). No Map. Soporte para Map a partir de JSF 2.2.
  - No se puede usar int[] ni double[] ni otros arrays de tipos primitivos
  - Use Integer[] o Double[]

Estructuras condicionales: Texto condicional

- Alternativas
  - #{someCondition ? simpleVal1 : simpleVal2}

  <h:outputText value="#{someValue}" rendered="#{someCondition}"/>
  - En general se usa h:blah y el atributo “rendered

  <ui:fragment rendered="#{someCondition}"/> <someHTML>...</ someHTML>

  </ui:fragment>
- Para desarrolladores JSTL
  - c:if y c:choose no interactuan adecuadamente con ui:repeat porque se ejecutan cuando el arbol de componentes se crea y no cuando se visualiza
  - Así que no use etiquetas de evaluación condicional de JSTL

Plantillas: pasos

- Definir un archivo plantilla
  - El contenido que aparecerá en los clientes es insertado directamente
  - El contenido que puede ser sustituido en los archivos de los clientess se marca con ui:insert (con valores por defecto para el caso que no se sumistre valor)
- Definición de un cliente que usa una plantilla
  - Usar ui:composition para indicar la plantilla a usar
  - Usar ui:define para sobreescribir el contenido cada sección reemplazable de la plantilla (marcada en el template con ui:insert)
- Acceso al archivo cliente en el navegador
  - http://host/app/clientfile.jsf
    - Los usuarios nunca acceden a las plantillas directamente

Ejemplo sencillo

- **/templates/template-1.xhtml** ...

  <h:body>Content shared by all client files
  <h2>
  <*ui:insert name="title"*>Default Title</ui:insert></h2> More content shared by all clients
  <*ui:insert name="body"*>Default Body</ui:insert> </h:body> ...

- **client-file-1.xhtml**

  <ui:composition template="/templates/template-1.xhtml">
  <ui:define name="title">Title text</ui:define> <ui:define name="body">
  Content to go in "body" section of template
  </ui:define>
  </ui:composition>

Incluyendo archivos en templates

- Contenido compartido por todos los clientes
  - Incluir el contenido directamente en el template
- Contenido especifico para un cliente
  - Incluirlo en el archivo del cliente en el cuerpo de ui:define (reemplazando una sección del archivo template)
- Si hay contenido compartido por algunos clientes
  - Y tal vez insertado en diferentes lugares en cada cliente
  - Problemas
    - No usado por todos los cliente por lo que no puede ir en el template
    - Usado por más de un cliente, por lo que sería muy repetitivo especificar el contenido en cada cliente con ui:define
- Solución
  - Poner el contenido a reusar en archivos separados
  - En archivo cliente, emplear ui:include en el cuerpo de ui:define

Uso de ui:include

- Contenido compartido por todos los clientes
  - Incluir el contenido directamente en el template
- Contenido especifico para un cliente
  - Incluirlo en el archivo del cliente en el cuerpo de ui:definefrom the template file)
- Si hay contenido compartido por algunos clientes
  - Y tal vez insertado en diferentes lugares en cada cliente
  - Problemas
    - No usado por todos los cliente por lo que no puede ir en el template
    - Usado por más de un cliente, por lo que sería muy repetitivo especificar el contenido en cada cliente con ui:define
- Solución
  - Poner el contenido a reusar en archivos separados
  - En archivo cliente, emplear ui:include en el cuerpo de ui:define

Los archivos a incluir (Snippet)

- **Incluir el contenido en ui:composition**
  - Con el esquema apropiado
- Colocar los **snippets en carpetas separadas**
  - Evitar la confusión con ficheros públicos o plantillas
- Usar sintaxis XML
  - Esta es la diferencia con los archivos incluidos con jsp:include, donde los snippets son etiquetas huérfanas. Aquí son archivos XML legales
    - El contenido fuera de ui:composition es ignorado
- **Snippets pueden usar templates**
  - La etiqueta raiz será <ui:composition>, no <html>
    - Ya que esos templates crear snippets no HTML

Archivo cliente

- Usar ui:composition
  - Con el esquema referencia y el atributo “template”

<ui:composition xmlns="..." xmlns:ui="..." template="/templates/some-template.xhtml">

- Usar ui:define con ui:include para hacer referencia a los snippets

<ui:define name="section-name1-from-template-file"> <ui:include src="/snippets/some-snippet.xhtml"/>

</ui:define>

<ui:define name="section-name2-from-template-file"> Client-specific content

</ui:define>

- Los archivos cliente van donde cualquier página JSF
  - Deben ser públicos
    - • Si el archivo es blah.xhtml, se accede como blah.jsf

Ejemplo: Archivo template (/templates/eboats-template.xhtml)

```
<!DOCTYPE ...>
<html xmlns="http://www.w3.org/
1999/xhtml"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:ui="http://java.sun.com/jsf/
facelets"> <h:head>
<title><ui:insert name="title">Title</
ui:insert></title> <link rel="stylesheet"
type="text/css"
href="./css/styles.css"/> </h:head>
<h:body>
<ui:insert name="header">Header</
ui:insert> <p/>
<table border="5" align="center">
<tr><th class="title"> <ui:insert
name="title">Title</ui:insert> </th></
tr> </table> <p/>
<table width="75" align="left"
cellspacing="5"> <tr><td><ui:insert
name="menu">Menu</ui:insert></
td></tr> </table>
<p/>
<ui:insert name="body">Body</
ui:insert>
<br clear="all"/>
<hr/>
<ui:insert name="footer">Footer</
ui:insert> </h:body></html>
```

Ejemplo: Snippets

Header file (/snippets/header.xhtml)

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml" xmlns:ui="http://java.sun.com/jsf/facelets">
<table width="100%" class="dark"> <tr><th align="left">
<a href="welcome.jsf" class="white">Home</a>    
<a href="products.jsf" class="white">Products</a>    
<a href="services.jsf" class="white">Services</a>    
<a href="contact.jsf" class="white">Contact Us</a>
</th><th align="right">
<a href="cart.jsf" class="white">My Cart</a>    
<a href="logout.jsf" class="white">Logout</a>    
<a href="help.jsf" class="white">Help</a>
</th></tr></table> </ui:composition>
```

(Footer File 1 (/snippets/footer-full.xhtml)

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml" xmlns:ui="http://java.sun.com/jsf/facelets"> <div align="center"> <a href="welcome.jsf">Home</a> | <a href="contact.jsf">Contact</a> | <a href="privacy.jsf">Privacy</a>
</div>
</ui:composition>
```

Ejemplo: Cliente

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml" xmlns:ui="http://java.sun.com/jsf/facelets" template="/templates/eboats-template.xhtml">
<ui:define name="title"> Welcome to eboats! </ui:define> <ui:define name="header">
<ui:include src="/snippets/header.xhtml"/>
</ui:define>
<ui:define name="menu"> <ui:include src="/snippets/google-search-box.xhtml"/>
</ui:define>
<ui:define name="body">
<p/>
<img src="./images/yacht.jpg" width="240" height="367"
align="right" alt="Base-model yacht"/>
Looking for a hole in the water into which to pour your money? You've come to the right place! We offer a wide selection of reasonably priced boats for everyday use.
<h2>Yachts</h2>
... (more body content)
</ui:define>
<ui:define name="footer">
<ui:include src="/snippets/footer-no-home.xhtml"/>
</ui:define> </ui:composition>
```

Bibliografía

- **www.coreservlets.com**
- **JSF 2 Docs home page**
  – http://javaserverfaces.java.net/nonav/docs/2.1/
- **JSF 2 Java API**
  – http://javaserverfaces.java.net/nonav/docs/2.1/javadocs/
- **JSF2TagsAPI**
  – http://javaserverfaces.java.net/nonav/docs/2.1/vdldocs/facelets/
- **Java 6 API**
  – http://docs.oracle.com/javase/6/docs/api/
- **Java 7 API**
  – http://docs.oracle.com/javase/7/docs/api/

Bibliografía

- **www.coreservlets.com**
- **JEE 6**
  - http://docs.oracle.com/javaee/6/tutorial/doc/giepu.html
- **JSF 2 Docs home page**
  - http://javaserverfaces.java.net/nonav/docs/2.2/
- **JSF 2 Java API**
  - http://javaserverfaces.java.net/nonav/docs/2.2/javadocs/
- **JSF2TagsAPI**
  - http://javaserverfaces.java.net/nonav/docs/2.2/vdldocs/facelets/
- **Java 7 API**
  - http://docs.oracle.com/javase/7/docs/api/