



JavaServer Faces

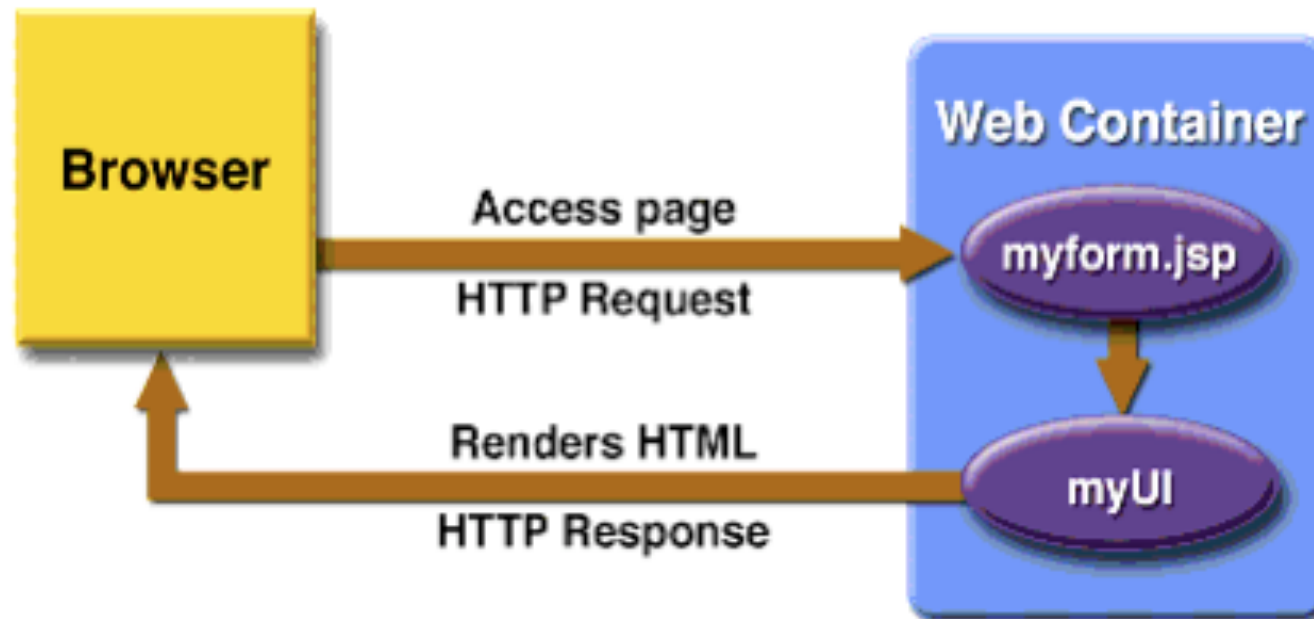
- Sesión 2: MVC en JSF



Índice

- **Vista:** componentes para la GUI
- **Modelo:** beans de respaldo: *backing* beans
 - Validación, conversión y tratamiento de errores
- **Controlador:** Acciones y navegación entre páginas
- Expresiones EL

Funcionamiento básico de JSF



Un primer ejemplo

The screenshot shows a web browser window with the title "Formulario de matrícula". The address bar displays the URL "http://localhost:8080/jsf-ejemplo1/". The page content includes a label "Dirección e-mail:" followed by a text input field. Below this is a label "Tecnologías Java de interés" followed by four checkboxes: "Servlets y JSP", "Struts", "JSF", and "JPA". At the bottom left of the form is a button labeled "Enviar".

Vista

- En Struts la vista se define mediante páginas y la aplicación se mueve de una página a otra.
- En JSF la vista se define mediante vistas (**view**) que contienen componentes.
- El controlador puede decidir qué componentes son visibles, e ir mostrando y escondiendo componentes.
- La vista se define mediante una página HTML con etiquetas especiales `<f :>` (componentes núcleo de JSF) y `<h :>` (componentes HTML)

Vista: página JSF

```
<f:view>
  <h:form>
    <table>
      <tr>
        <td>Dirección e-mail:</td>
        <td><h:inputText value="#{selecCursosBean.email}" /></td>
      </tr>
      <tr>
        <td>Tecnologías Java de interés</td>
        <td><h:selectManyCheckbox
          value="#{selecCursosBean.cursosId}">
          <f:selectItem itemValue="JSP" itemLabel="Servlets y JSP" />
          <f:selectItem itemValue="Struts" itemLabel="Struts" />
          <f:selectItem itemValue="JSF" itemLabel="JSF" />
          <f:selectItem itemValue="JPA" itemLabel="JPA" />
        </h:selectManyCheckbox></td>
      </tr>
    </table>
    <h:commandButton value="Enviar"
      action="#{selecCursosHandler.grabarDatosCursos}" />
  </h:form>
</f:view>
```

Beans gestionados

Fichero selec-cursos.xhtml



Modelo: Beans gestionados

```
public class SelecCursosBackingBean {  
    private String email;  
    private String[] cursosId;  
  
    public String getEmail() {  
        return email;  
    }  
  
    public void setEmail(String email) {  
        this.email = email;  
    }  
  
    public String[] getCursosId() {  
        return cursosId;  
    }  
  
    public void setCursosId(String[] cursosId) {  
        this.cursosId = cursosId;  
    }  
}
```



Declaración en faces-config.xml

```
...
<managed-bean>
  <managed-bean-name>selecCursosBB</managed-bean-name>
  <managed-bean-class>
    jtech.jsf.SelecCursosBackingBean
  </managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
<managed-bean>
  <managed-bean-name>selecCursosHandler</managed-bean-name>
  <managed-bean-class>jtech.jsf.SelecCursosHandler
  </managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
  <managed-property>
    <property-name>datosCursos</property-name>
    <value>#{selecCursosBB}</value>
  </managed-property>
</managed-bean>
...
```


Declaración en JSF 2.0 (contenedor web)

- Anotaciones en la propia clase Java

```
@ManagedBean
@SessionScoped
public class SelecCursosBean {
    private String email;
    private String[] cursosId;
    ...
}
```

Nombre por defecto:
selecCursosBean

Definimos nombre del
bean

```
@ManagedBean(name="selecCursos")
@SessionScoped
public class SelecCursosBean {
    private String email;
    private String[] cursosId;
    ...
}
```

Declaración en JSF 2.0 (serv. aplicaciones)

- Anotaciones en la propia clase Java

```
@Named
@SessionScope
public class SelecCursosBean {
    private String email;
    private String[] cursosId;
    ...
}
```

Nombre por defecto:
selecCursosBean

Definimos nombre del
bean

```
@Named("selecCursos")
@SessionScope
public class SelecCursosBean {
    private String email;
    private String[] cursosId;
    ...
}
```

Ámbito de vida de los beans gestionados (I)

- **Petición:** Se define con el valor **request** en el `faces-config.xml` o con la anotación **@RequestScoped** en la clase. El bean se asocia a una petición HTTP. Cada nueva petición crea un nuevo bean y lo asocia con la página. Se usa para el paso de mensajes que no sea necesario propagar a lo largo de la aplicación.
- **Sesión:** Se define con el valor **session** en el `faces-config.xml` o con la anotación **@SessionScoped** en la clase. El bean se asocia a una sesión definida con el API de Servlets. Se usa para conservar elementos que queremos mantener a lo largo de la aplicación como, por ejemplo: un usuario logueado.
- **Aplicación:** Se define con el valor **application** y con la anotación **@ApplicationScoped**. Los beans con este ámbito viven asociados a la aplicación. Definen singletons que se crean e inicializa sólo una vez, al comienzo de la aplicación. Se usa para guardar características comunes compartidas y utilizadas por el resto de los beans de la aplicación (ej: valor del IVA en un e-commerce)

Ámbito de vida de los beans gestionados (II)

- **Vista:** Se define con el valor **view** en el `faces-config.xml` o la anotación **@ViewScoped** en la clase. Un bean en este ámbito persistirá mientras no naveguemos a otra vista. Se suele usar en páginas Ajax.
- **Custom:** Se define con la anotación **@CustomScoped("#{expt}")** en la clase. Definimos un mapa en el que nuestra aplicación será la responsable de la eliminación de los elementos. Ejemplo: un ámbito “registro”, donde al finalizar el proceso de registro se eliminarán los datos introducidos por el usuario.
- **Conversación:** Se define con la anotación **@ConversationScoped**. Ligado a una ventana o pestaña concreta del navegador. Una sesión puede mantener varias conversaciones en distintas páginas. Es propio de CDI, no de JSF.

Inicialización de los beans (I)

- En el constructor de la clase Java

```
public class SelecCursosBean {  
    private String email;  
    private String[] cursosId;  
  
    public SelecCursosBean() {  
        email="Introduce tu e-mail";  
    }  
    ...  
}
```

- En el fichero faces-config con expresiones EL

```
<managed-bean>  
    <managed-bean-name>selecCursosBean</managed-bean-name>  
    <managed-bean-class>org.especialistajee.jsf.SelecCursosBean</  
managed-bean-class>  
    <managed-bean-scope>request</managed-bean-scope>  
    <managed-property>  
        <property-name>email</property-name>  
        <value>Introduce tu e-mail</value>  
    </managed-property>  
</managed-bean>
```

Inicialización de los beans (II)

- Con JSF 2.0 podemos combinar ambos métodos

```
@ManagedBean
@RequestScoped
public class SelecCursosBean {
    @ManagedProperty(value="Introduce tu e-mail")
    private String email;
    private String[] cursosId;
    ...
}
```

- Expresiones EL

```
@ManagedBean
@RequestScoped
public class SelecCursosBean {
    @ManagedProperty(value="#{1+2+3+4}")
    private String email;
    private String[] cursosId;
    ...
}
```

Relaciones entre beans (I)

- En el fichero de configuración, mediante expresiones EL

```
<managed-bean>
  <managed-bean-name>entradaBean</managed-bean-name>
  <managed-bean-
class>org.especialistajee.beans.EntradaBean</
managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
    <managed-property>
      <property-name>usuario</property-name>
      <value>#{usuarioBean}</value>
    </managed-property>
</managed-bean>
```

Relaciones entre beans (II)

- Las expresiones EL también permiten asociar un bean a otro en tiempo de inicialización

```
@ManagedBean
@RequestScoped
public class SelecCursosController {
    @ManagedProperty(value="#{selecCursosBean}")
    private SelecCursosBean datosCursos;

    public void setDatosCursos(SelecCursosBean datosCursos) {
        this.datosCursos = datosCursos;
    }

    public SelecCursosBean getDatosCursos() {
        return datosCursos;
    }
    ...
}
```


Relaciones entre beans (III)

- Si nuestro servidor tiene soporte de CDI, la opción recomendada es:

```
@ManagedBean
@RequestScoped
public class SelecCursosController {
    @Inject
    private SelecCursosBean datosCursos;

    public void setDatosCursos(SelecCursosBean datosCursos) {
        this.datosCursos = datosCursos;
    }

    public SelecCursosBean getDatosCursos() {
        return datosCursos;
    }
    ...
}
```

Relaciones entre beans (IV)

- JSF llama al método set (*inyecta* el bean dependiente) cuando se inicializa el bean propietario de la relación.
- El ámbito del bean propietario de la relación debe ser menor o igual que el del bean inyectado para estar seguros de que éste existe.

Cuando nuestro bean tiene el ámbito...	...su propiedades pueden ser beans de los ámbitos
none	none
application	none, application
session	none, application, session
view	none, application, session, view
request	none, application, session, view, request



Controlador

Fichero
selec-cursos.xhtml

```
<h:commandButton value="Enviar"  
    action="#{selecCursosHandler.grabarDatosCursos}"/>
```

Bean gestionado

```
public class SelecCursosHandler {  
    SelecCursosBackingBean datosCursos;  
  
    public SelecCursosBackingBean getDatosCursos() {  
        return datosCursos;  
    }  
  
    public void setDatosCursos(SelecCursosBackingBean datosCursos) {  
        this.datosCursos = datosCursos;  
    }  
  
    public String grabarDatosCursos() {  
        EstudianteBO estudianteBO = new EstudianteBO();  
        String email = datosCursos.getEmail();  
        String[] cursosId = datosCursos.getCursosId();  
        estudianteBO.grabarAsignaturas(email, cursosId);  
        return "OK";  
    }  
}
```

Dependencias entre beans

- JSF no permite parámetros en las acciones
- ¿Cómo se accede a los datos introducidos por el usuario en la página?
- Solución 1: definir un único bean que contenga los datos y las acciones
- Solución 2: inyectar el bean con los datos en el controlador

```
<managed-bean>
  <managed-bean-name>selecCursosController</managed-bean-name>
  <managed-bean-class>jtech.jsf.controlador.SelecCursosController
</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
  <managed-property>
    <property-name>datosCursos</property-name>
    <value>#{selecCursosBean}</value>
  </managed-property>
</managed-bean>
```



Navegación

- La navegación se refiere al flujo de páginas a lo largo de la aplicación.
- Dos tipos de navegación:
 - Estática
 - Dinámica

Navegación estática

- Al hacer click sobre un botón/enlace, el destino será siempre el mismo.

```
<h:commandButton label="Login" action="welcome"/>
```

- La acción se transforma en un identificador de vista:
 - Si el nombre no tiene extensión, se le da la misma que tiene la vista actual
 - Si el nombre no empieza por /, se le asigna la misma ruta que la vista actual
- En el ejemplo, suponiendo que estemos en /home/index.xhtml, se redirigirá a /home/welcome.xhtml

Navegación dinámica

- El flujo depende de los datos introducidos
- El resultado de una acción debe ser una cadena
- JSF decide la siguiente vista a mostrar siguiendo las reglas del fichero `faces-config.xml`, donde se mapea esta cadena contra una vista

```
...  
<navigation-rule>  
  <from-view-id>/selec-cursos.xhtml</from-view-id>  
  <navigation-case>  
    <from-outcome>OK</from-outcome>  
    <to-view-id>/cursos-grabados.jsp</to-view-id>  
  </navigation-case>  
</navigation-rule>  
...
```

Navegación dinámica según acción


- Permite tener cadenas iguales que desemboquen en diferentes destinos

```
...
<navigation-rule>
  <from-view-id>/menu-principal.xhtml</from-view-id>
  <navigation-case>
    <from-action>#{loginController.cerrarSesion}</from-action>
    <from-outcome>success</from-outcome>
    <to-view-id>/index.xhtml</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-action>#{entradaController.anyadirEntrada}</from-action>
    <from-outcome>success</from-outcome>
    <to-view-id>/verEntrada.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
...
```


Navegación dinámica según condición

- Comparaciones simples usando EL

```
...  
<navigation-rule>  
  <from-view-id>login.xhtml</from-view-id>  
  <navigation-case>  
    <from-outcome>success</from-outcome>  
    <if>#{user.powerUser}</if>  
    <to-view-id>/index_power.xhtml</to-view-id>  
  </navigation-case>  
  <navigation-case>  
    <from-outcome>success</from-outcome>  
    <if>#{user.vipUser}</if>  
    <to-view-id>/index_vip.xhtml</to-view-id>  
  </navigation-case>  
  <navigation-case>  
    <from-outcome>success</from-outcome>  
    <to-view-id>/index_user.xhtml</to-view-id>  
  </navigation-case>  
</navigation-rule>  
...
```



```
if(user.isPowerUser()){  
    return "powerUser";  
} else if(user.isVipUser()){  
    return "vipUser";  
}  
return "user"
```

Navegación dinámica: Dynamic Target View IDs

- El elemento to-view-id puede ser una expresión EL, que se evalúa en tiempo de ejecución

```
<navigation-rule>
  <from-view-id>/main.xhtml</from-view-id>
  <navigation-case>
    <to-view-id>#{quizBean.nextViewID}</to-view-id>
  </navigation-case>
</navigation-rule>
```

Resumen: vista

Vista: *selec-cursos.jsp*

```
...  
<h:input_text value=  
    "#{selecCursosBB.email}"/>  
...  
<h:commandButton value="Enviar"  
    action="#{selecCursosHandler.grabarDatosCursos}">  
...  
...
```

Formulario de matrícula

Dirección e-mail:

Tecnologías Java de interés ☐ Servlets y JSP ☐ Struts ☐ JSF ☐ JPA

Configuración: *faces-config.xml*

```
...  
<managed-bean>  
    <managed-bean-name>selecCursosBB  
    </managed-bean-name>  
    <managed-bean-class>  
        SelecCursosBackingBean  
    </managed-bean-class>  
    <managed-bean-scope>session  
    </managed-bean-scope>  
</managed-bean>  
...  
<managed-bean>  
    <managed-bean-name>selecCursosHandler  
    </managed-bean-name>  
    <managed-bean-class>SelecCursosHandler  
    </managed-bean-class>  
    <managed-bean-scope>session  
    </managed-bean-scope>  
    <managed-property>  
        <property-name>datosCursos</property-name>  
        <value>#{selecCursosBB}</value>  
    </managed-property>  
</managed-bean>  
...
```

Backing bean **selecCursosBB**

Backing bean **selecCursosHandler**

Resumen: modelo

Vista: *selec-cursos.jsp*

```
...  
<h:input_text value=  
    "#{selecCursosBB.email}" />  
...  
<h:commandButton value="Enviar"  
    action="#{selecCursosHandler.grabarDatosCursos}">  
...  

```

Formulario de matrícula

Dirección e-mail:

Tecnologías Java de interés ☐ Servlets y JSP ☐ Struts ☐ JSF ☐ JPA

http://localhost:8080/jsf-ejemplo1/ Google

SeleCursosBackingBean

```
getEmail() : String  
setEmail(String) : void  
getCursosId() : String[]  
setCursosId(String[]) : void
```

SeleCursosHandler

```
getDatosCursos() : SeleCursosBackingBean  
setDatosCursos(SeleCursosBackingBean)  
grabarDatosCursos() : String
```

Resumen: controlador

Vista: selec-cursos.jsp

```
...  
<h:input_text value=  
    "#{selecCursosBB.email}"/>  
...  
<h:commandButton value="Enviar"  
    action="#{selecCursosHandler.grabarDatosCursos}">  
...  

```

Formulario de matrícula

Dirección e-mail:

Tecnologías Java de interés ☐ Servlets y JSP ☐ Struts ☐ JSF ☐ JPA

SelecCursosHandler

```
public String grabarDatosCursos() {  
    EstudianteBO estudianteBO = new EstudianteBO();  
    String email = datosCursos.getEmail();  
    String cursosId[] = datosCursos.getCursosId();  
    estudianteBO.grabarAsignaturas(email, cursosId);  
    return "OK";  
}
```

Configuración: faces-config.xml

```
...  
<navigation-rule>  
    <from-view-id>/selec-cursos.jsp</from-view-id>  
    <navigation-case>  
        <from-outcome>OK</from-outcome>  
        <to-view-id>/cursos-grabados.jsp</to-view-id>  
    </navigation-case>  
</navigation-rule>  
...  

```

Regla de navegación



Expresiones EL

- Se utilizan para dar valor a atributos de los componentes JSF
- Sintaxis: `#{...}`
- Ampliación de las expresiones EL de JSP
- Se evalúan en la fase de *Apply Request Values*, cuando JSF llama al método *decode* del componente.
- Expresiones con semántica *getValue* y con semántica *setValue*

Ejemplos y usos de expresiones EL

```
#{foo.bar}  
#{foo[bar]}  
#{foo["bar"]}  
#{foo[3]}  
#{foo[3].bar}  
#{foo.bar[3]}  
#{customer.status == 'VIP'}  
#{(page1.city.fahrenheitTemp - 32) * 5 / 9}
```

- Las expresiones EL se pueden utilizar en múltiples atributos de los componentes JSF, no sólo en el atributo value.

```
<h:outputText  
    value="El total del pedido es: #{pedido.importe}"  
    style="#{pedido.cssStyle}"
```

Objetos implícitos JSF

- **requestScope**, **sessionScope**, **applicationScope**: permite acceder a las variables definidas `sdfasdfsdf asdfasdfsdf` en el ámbito de la petición, de la sesión y de la aplicación.
- **param**: para acceder a los valores de los parámetros de la petición.
- **paramValues**: para acceder a los arrays de valores de los parámetros de la petición.
- **header**: para acceder a los valores de las cabeceras de la petición.
- **headerValues**: para acceder a los arrays de valores de los parámetros de la petición.
- **cookie**: para acceder a los valores almacenados en las cookies en forma de objetos **`javax.servlet.http.Cookie`**
- **initParam**: para acceder a los valores de inicialización de la aplicación.
- **facesContext**: para acceder al objeto `javax.faces.context.FacesContext` asociado a la aplicación actual.
- **view**: para acceder al objeto **`javax.faces.component.UIViewRoot`** asociado a la vista actual.

```
#{view.children[0].children[0].valid}
```


Componentes HTML <h:> (I)

Etiqueta	Descripción
<code><h:commandLink></code>	Representa un comando que se renderiza como un enlace. Cuando el usuario pincha en el enlace, se ejecuta un código javascript que envía el formulario al que pertenece y se lanza un evento <code>ActionEvent</code> .
Ejemplo:	
<pre><h:form> <h:commandLink action="#{formController.save}"> <h:outputText value="Save" /> </h:commandLink> </h:form></pre>	

Etiqueta	Descripción
<code><h:commandButton></code>	Representa un comando que se renderiza como un botón HTML de tipo entrada. Cuando el usuario cliquea el botón, se envía el formulario al que pertenece y se lanza un evento <code>ActionEvent</code> .
Ejemplo:	
<pre><h:form> <h:commandButton value="Save" action="#{formController.save}" /> </h:form></pre>	

Componentes HTML <h:> (II)

Etiqueta	Descripción
<h:dataTable>	Se renderiza en un elemento HTML <table>. Los elementos hijo <h:column> son los responsables de renderizar las columnas de la tabla. El atributo value debe ser un array de objetos y se define una variable que hace de iterador sobre ese array. Se puede indicar el primer objeto a mostrar y el número de filas con los atributos first="first" y rows="rows". Los componentes de la tabla pueden declararse con la faceta header y footer.
Ejemplo:	
<pre><h:dataTable value="#{reportController.currentReports}" var="report"> <f:facet name="header"> <h:outputText value="Expense Reports" /> </f:facet> <h:column rendered="#{reportController.showDate}"> <f:facet name="header"> <h:outputText value="Date" /> </f:facet> <h:outputText value="#{report.date}" /> </h:column> ... </h:dataTable></pre>	

Etiqueta	Descripción
<h:column>	Se utiliza dentro de una etiqueta <h:dataTable> para representar una columna de datos tabulares. Podemos añadirle una etiqueta <f:facet name="header"> o <f:facet name="footer">.
Ejemplo:	
<pre>><h:dataTable value="#{reportController.currentReports}" var="report"> <h:column rendered="#{reportController.showDate}"> <f:facet name="header"> <h:outputText value="Date" /> </f:facet> <h:outputText value="#{report.date}" /> </h:column> ... </h:dataTable></pre>	



Componentes HTML <h:> (III)

Etiqueta	Descripción
<h:form>	Se renderiza como un elemento <form> con un atributo de acción definido por una URL que identifica la vista contenida en el formulario. Cuando se envía el formulario, sólo se procesan los componentes hijos del formulario enviado.
Ejemplo:	
<pre><h:form> <h:panelGrid columns="2"> <h:outputText value="First name:" /> <h:inputText value="#{user.firstName}" /> <h:outputText value="Last name:" /> <h:inputText value="#{user.lastName}" /> </h:panelGrid> </h:form></pre>	

Etiqueta	Descripción
<h:graphicImage>	Se renderiza como un elemento con un atributo src que toma como valor el valor del atributo value de la etiqueta.
Ejemplo:	
<pre><h:graphicImage value="/images/folder-open.gif" /></pre>	



Componentes HTML <h:> (IV)

Etiqueta	Descripción
<h:inputHidden>	Se renderiza como un elemento <input> con un atributo type definido como hidden.
Ejemplo:	
<pre><h:form> <h:inputHidden value="#{user.type}" /> </h:form></pre>	

Etiqueta	Descripción
<h:inputSecret>	Se renderiza como un elemento <input> con un atributo type definido como password.
Ejemplo:	
<pre><h:form> <h:inputSecret value="#{user.password}" /> </h:form></pre>	

Etiqueta	Descripción
<h:inputText>	Se renderiza como un elemento <input> con un atributo type definido como text.
Ejemplo:	
<pre><h:form> <h:inputText value="#{user.email}" /> </h:form></pre>	

Etiqueta	Descripción
<h:inputTextarea>	Se renderiza como un elemento <textarea>.
Ejemplo:	
<pre><h:form> <h:inputTextarea value="#{user.bio}" /> </h:form></pre>	



Componentes HTML <h:> (V)

Etiqueta	Descripción
<h:message>	Este elemento obtiene el primer mensaje encolado para el componente identificado por el atributo <code>for</code> .
Ejemplo:	
<pre><h:form> <h:inputText id="firstName" value="#{user.firstName}" /> <h:message for="firstName" errorStyle="color: red" /> </h:form></pre>	

Etiqueta	Descripción
<h:messages>	Este elemento obtiene todos los mensajes encolados.
Ejemplo:	
<pre><h:messages/> <h:form> <h:inputText id="firstName" value="#{user.firstName}" /> <h:message for="firstName" errorStyle="color: red" /> </h:form></pre>	



Componentes HTML <h:> (VI)

Etiqueta	Descripción
<h:outputFormat>	Define un mensaje parametrizado que será rellenado por los elementos definidos en parámetros <f:param>
Ejemplo:	
<pre><f:loadBundle basename="messages" var="msgs" /> <h:outputFormat value="#{msgs.sunRiseAndSetText}"> <f:param value="#{city.sunRiseTime}" /> <f:param value="#{city.sunSetTime}" /> </h:outputFormat></pre>	

Etiqueta	Descripción
<h:outputLabel>	Define un elemento HTML <label>.
Ejemplo:	
<pre><h:inputText id="firstName" value="#{user.firstName}" /> <h:outputLabel for="firstName" /></pre>	

Etiqueta	Descripción
<h:outputLink>	Se renderiza como un elemento <a> con un atributo href definido como el valor del atributo value.
Ejemplo:	
<pre><h:outputLink value="../../../logout.jsp" /></pre>	

Etiqueta	Descripción
<h:outputText>	Se renderiza como texto.
Ejemplo:	
<pre><h:outputText value="#{user.name}" /></pre>	

Componentes HTML <h:> (VII)

Etiqueta	Descripción
<h:panelGrid>	Se renderiza como una tabla de HTML, con el número de columnas definido por el atributo <code>columns</code> . Los componentes del panel pueden tener las facetas <code>header</code> y <code>footer</code> .
Ejemplo:	
<pre><h:form> <h:panelGrid columns="2"> <h:outputText value="First name:" /> <h:inputText value="#{user.firstName}" /> <h:outputText value="Last name:" /> <h:inputText value="#{user.lastName}" /> </h:panelGrid> </h:form></pre>	

Etiqueta	Descripción
<h:panelGroup>	El componente actúa como un contenedor de otros componentes en situaciones en las que sólo se permite que exista un componente, por ejemplo cuando un grupo de componentes se usa como una faceta dentro de un <code>panelGroup</code> . Se renderizará como un elemento <code></code> . Si le ponemos el atributo <code>layout="block"</code> , se renderizará como un <code>>div<</code>
Ejemplo:	
<pre><h:form> <h:panelGrid columns="2"> <f:facet name="header"> <h:panelGroup> <h:outputText value="Sales stats for " /> </h:outputText value="#{sales.region}" style="font-weight: bold" /> </h:panelGroup> </f:facet> <h:outputText value="January" /> <h:inputText value="#{sales.jan}" /> <h:outputText value="February" /> <h:inputText value="#{sales.feb}" /> ... </h:panelGrid> </h:form></pre>	

Componentes HTML <h:> (VIII)

Etiqueta	Descripción
<h:selectBooleanCheckbox>	Se renderiza como un elemento <input> con un atributo type definido como checkbox.
Ejemplo: <pre><h:form> <h:selectBooleanCheckbox value="#{user.vip}" /> </h:form></pre>	

Etiqueta	Descripción
<h:selectManyCheckbox>	Se renderiza como un elemento HTML <table> con un elemento input por cada uno de sus hijos, componentes de tipo <f:selectItem> y <f:selectItems>.
Ejemplo: <pre><h:form> <h:selectManyCheckbox value="#{user.projects}"> <f:selectItems value="#{allProjects}" /> </h:selectManyCheckbox> </h:form></pre>	

Etiqueta	Descripción
<h:selectManyListbox>	Se renderiza como un elemento <select>. Las opciones se representan por los componentes hijos de tipo <f:selectItem> y <f:selectItems>.
Ejemplo: <pre><h:form> <h:selectManyListbox value="#{user.projects}"> <f:selectItems value="#{allProjects}" /> </h:selectManyListbox> </h:form></pre>	



Componentes HTML <h:> (IX)

Etiqueta	Descripción
<h:selectManyMenu>	Se renderiza como un elemento <select>. Las opciones se representan por los componentes hijos de tipo <f:selectItem> y <f:selectItems>.
Ejemplo:	
<pre><h:form> <h:selectManyMenu value="#{user.projects}"> <f:selectItems value="#{allProjects}" /> </h:selectManyMenu> </h:form></pre>	

Etiqueta	Descripción
<h:selectOneListbox>	Se renderiza como un elemento <select>. Las opciones se representan por los componentes hijos de tipo <f:selectItem> y <f:selectItems>.
Ejemplo:	
<pre><h:form> <h:selectOneListbox value="#{user.country}"> <f:selectItems value="#{allCountries}" /> </h:selectOneListbox> </h:form></pre>	



Componentes HTML <h:> (X)

Etiqueta	Descripción
<h:selectManyMenu>	Se renderiza como un elemento <select>. Las opciones se representan por los componentes hijos de tipo <f:selectItem> y <f:selectItems>.
Ejemplo:	
<pre><h:form> <h:selectManyMenu value="#{user.projects}"> <f:selectItems value="#{allProjects}" /> </h:selectManyMenu> </h:form></pre>	

Etiqueta	Descripción
<h:selectOneListbox>	Se renderiza como un elemento <select>. Las opciones se representan por los componentes hijos de tipo <f:selectItem> y <f:selectItems>.
Ejemplo:	
<pre><h:form> <h:selectOneListbox value="#{user.country}"> <f:selectItems value="#{allCountries}" /> </h:selectOneListbox> </h:form></pre>	

Componentes HTML <h:> (XI)

Etiqueta	Descripción
<h:selectOneMenu>	Se renderiza como un elemento <select>. Las opciones se representan por los componentes hijos de tipo <f:selectItem> y <f:selectItems>.
Ejemplo:	
<pre><h:form> <h:selectOneMenu value="#{user.country}"> <f:selectItems value="#{allCountries}" /> </h:selectOneMenu> </h:form></pre>	

Etiqueta	Descripción
<h:selectOneRadio>	Se renderiza como un elemento <input> con un atributo type definido como radio. Las opciones se representan por los componentes hijos de tipo <f:selectItem> y <f:selectItems>.
Ejemplo:	
<pre><h:form> <h:selectOneRadio value="#{user.country}"> <f:selectItems value="#{allCountries}" /> </h:selectOneRadio> </h:form></pre>	



Etiquetas Core <f:> (I)

Etiqueta	Descripción
<f:actionListener>	Crea una instancia de la clase definida en el atributo <code>type</code> y la añade al componente padre de tipo <code>action</code> para manejar el evento relacionado con el disparo de la acción.
Ejemplo:	
<pre><h:form> <h:commandButton value="Save"> <f:actionListener type="com.mycompany.SaveListener" /> </h:commandButton> </h:form></pre>	

Etiqueta	Descripción
<f:attribute>	Define un atributo genérico para el componente padre.
Ejemplo:	
<pre><h:form> <h:inputText id="from" value="#{filter.from}" /> <h:inputText value="#{filter.to}"> <f:validator validatorId="com.mycompany.laterThanValidator" /> <f:attribute name="compareToComp" value="from" /> </h:inputText> </h:form></pre>	



Etiquetas Core <f:> (II)

Etiqueta	Descripción
<f:convertDateTime>	Crea una instancia de un conversor de tipo <code>DateTime</code> con los parámetros proporcionados y lo registra en el componente padre.
Ejemplo:	
<pre><h:form> <h:inputText value="#{user.birthDate}"> <f:convertDateTime dateStyle="short" /> </h:inputText> </h:form></pre>	

Etiqueta	Descripción
<f:convertNumber>	Crea una instancia de un conversor de tipo <code>Number</code> y lo registra en el componente padre.
Ejemplo:	
<pre><h:form> <h:inputText value="#{user.salary}"> <f:convertNumber integerOnly="true" /> </h:inputText> </h:form></pre>	

Etiqueta	Descripción
<f:converter>	Crea una instancia de un conversor definido por el usuario y registrado con un identificador determinado.
Ejemplo:	
<pre><h:form> <h:inputText value="#{user.ssn}"> <f:converter converterId="ssnConverter" /> </h:inputText> </h:form></pre>	



Etiquetas Core <f:> (III)

Etiqueta	Descripción
<f:facet>	Añade un componente con una faceta determinada al componente padre. Para que más de un componente pueda compartir la misma faceta, se deben agrupar con un elemento <h:panelGroup>.

Ejemplo:

```
<h:dataTable value="#{reportController.reports}" var="report">
  <f:facet name="header">
    <h:outputText value="Reports" />
  </f:facet>
  ...
</h:dataTable>
```

Etiqueta	Descripción
<f:loadBundle>	Carga un fichero de recursos y lo hace disponible a través de una variable. El path del fichero debe estar disponible en el classpath de la aplicación web (esto es, en el directorio WEB-INF/classes).

Ejemplo:

```
<f:loadBundle basename="messages" var="msgs" />
<h:outputText value="#{msgs.title}" />
```

Etiqueta	Descripción
<f:param>	Define un parámetro de una expresión de texto.

Ejemplo:

```
<f:loadBundle basename="messages" var="msgs" />
<h:outputFormat value="#{msgs.sunRiseAndSetText}">
  <f:param value="#{city.sunRiseTime}" />
  <f:param value="#{city.sunSetTime}" />
</h:outputFormat>
```



Etiquetas Core <f:> (IV)

Etiqueta	Descripción
<f:selectItem>	Crea una instancia de un ítem y se lo asigna al componente padre.
Ejemplo:	
<pre><h:form> <h:selectManyCheckbox value="#{user.projects}"> <f:selectItem itemValue="JSF" itemValue="1" /> <f:selectItem itemValue="JSP" itemValue="2" /> <f:selectItem itemValue="Servlets" itemValue="3" /> </h:selectManyCheckbox> </h:form></pre>	

Etiqueta	Descripción
<f:selectItems>	Crea múltiples instancias de ítems y los asigna al componente padre.
Ejemplo:	
<pre><h:form> <h:selectManyCheckbox value="#{user.projects}"> <f:selectItems value="#{allProjects}" /> </h:selectManyCheckbox> </h:form></pre>	

Etiqueta	Descripción
<f:subView>	Crea un grupo de componentes dentro de una vista.
Ejemplo:	
<pre><f:view> <f:subview id="header"> <jsp:include page="header.jsp" /> </f:subview> ... <f:subview id="footer"> <jsp:include page="footer.jsp" /> </f:subview> </f:view></pre>	



Etiquetas Core <f:> (V)

Etiqueta	Descripción
<f:validateDoubleRange>	Crea una instancia de validador de rango doble y lo asigna al componente padre.
Ejemplo:	
<pre><h:inputText value="#{product.price}"> <f:convertNumber type="currency" /> <f:validateDoubleRange minimum="0.0" /> </h:inputText></pre>	

Etiqueta	Descripción
<f:validateLength>	Crea una instancia de validador de longitud de texto y lo asigna al componente padre.
Ejemplo:	
<pre><h:inputText value="#{user.zipCode}"> <f:validateLength minimum="5" maximum="5" /> </h:inputText></pre>	

Etiqueta	Descripción
<f:validateLongRange>	Crea una instancia de validador de rango long y lo asigna al componente padre.
Ejemplo:	
<pre><h:inputText value="#{employee.salary}"> <f:convertNumber type="currency" /> <f:validateLongRange minimum="50000" maximum="150000" /> </h:inputText></pre>	

Etiquetas Core <f:> (VI)

Etiqueta	Descripción
<code><f:valueChangeListener></code>	Crea una instancia de la clase definida por el atributo <code>type</code> y la asigna al componente padre para ser llamada cuando sucede un evento de cambio de valor.
Ejemplo:	
<pre><h:form> <h:selectBooleanCheckbox value="Details" immediate="true"> <f:valueChangeListener type="com.mycompany.DescrLevelListener" /> </h:selectBooleanCheckbox> </h:form></pre>	

Etiqueta	Descripción
<code><f:verbatim></code>	Permite renderizar texto.
Ejemplo:	
<pre><f:subview id="header"> <f:verbatim> <html> <head> <title>Welcome to my site!</title> </head> </f:verbatim> </f:subview></pre>	

Etiqueta	Descripción
<code><f:view></code>	Crea una vista JSF.
Ejemplo:	
<pre><f:view locale="#{user.locale}"> ... </f:view></pre>	



Etiquetas Core <f:> (VII)

Etiqueta	Descripción
<f:phaseListener>	Añade un phase listener a la vista padre. Esto nos permite realizar llamadas antes y después de la fase que nosotros queramos dentro del ciclo de vida de JSF
Ejemplo:	
<pre><h:commandButton action="#{jsfBean.submit}" value="Submit"> <f:phaseListener binding="#{jsfBean.phaseListenerImpl}" type="org.especialistajee.jsf.PhaseListenerImpl"/> </h:commandButton></pre>	

Etiqueta	Descripción
<f:event>	listener de eventos
Ejemplo:	
<pre><h:outputText id="beforeRenderTest1" > <f:event type="javax.faces.event.beforeRender" action="#{eventTagBean.beforeEncode}" /> </h:outputText></pre>	

Etiqueta	Descripción
<f:validateRequired>	El valor es obligatorio
Ejemplo:	
<pre><h:inputSecret id="password" value="#{user.password}"> <f:validateRequired /> </h:inputSecret></pre>	

Etiquetas Core <f:> (VIII)

Etiqueta	Descripción
<f:validateRegex>	Valida un valor contra una expresión regular
Ejemplo:	
<pre><h:inputSecret id="password" value="#{user.password}"> <f:validateRegex pattern="((?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[@#\$%]).{6,20})" /> </h:inputSecret></pre>	

Etiqueta	Descripción
<f:validateBean>	Hace uso del API de validación de beans (JSR 303) para realizar la validación.
Ejemplo:	
<pre><h:inputText value="#{sampleBean.userName}"> <f:validateBean disabled="true" /> </h:inputText></pre>	

Etiqueta	Descripción
<f:viewParam>	Define un parámetro en la vista que puede inicializarse a partir de cualquier parámetro de la petición.
Ejemplo:	
<pre><f:metadata> <f:viewParam name="id" value="#{bean.id}" /> </f:metadata></pre>	



Etiquetas Core <f:> (IX)

Etiqueta	Descripción
<code><f:metadata></code>	Agrupar viewParams.
Ejemplo:	
<pre><f:metadata> <f:viewParam name="id" value="#{bean.id}" /> </f:metadata></pre>	

Etiqueta	Descripción
<code><f:ajax></code>	Dota de comportamiento AJAX a los componentes.
Ejemplo:	
<pre><h:inputSecret id="passInput" value="#{loginController.password}"> <f:ajax event="keyup" render="passError"/> </h:inputSecret></pre>	



¿Preguntas?