Introducción a JavaServer Faces

Índice

JavaServer Faces (JSF) es un *framework* para el desarrollo de interfaces web en Java que ofrece al desarrollador:

- **Un conjunto estándar de componentes gráficos** (*widgets*) para la web, al estilo de los componentes de Swing. En JavaServer Faces los componentes se definen mediante *taglibs* propias.
- **Vinculación entre dichos componentes y *beans* Java** denominados *backing beans*. Los componentes muestran la información que hay en los *backing beans* y la información introducida en los componentes modifica su contenido.
- **Soporte para validación y conversión automática de los datos** introducidos en los componentes web.
- **Un modelo de navegación** en el que el desarrollador puede decidir qué página JSP se mostrará a continuación en función de cuál es la página actual y el resultado de la última acción realizada.
- **Soporte para la internacionalización** de las aplicaciones.

El framework JSF implementa un MVC (Modelo, Vista, Controlador), en donde la *Vista* se compone de páginas JSP con etiquetas específicas JSF, el *Modelo* está representado por los denominados *backing beans*, beans Java definidos en el fichero de configuración y asociados a las páginas JSF y el *Controlador* se implementa mediante métodos en los *backing beans* que son llamados desde acciones disparadas por eventos de usuario (pulsar un botón o un enlace).

En ciertas funcionalidades JSF se solapa con Struts. Por ejemplo, ambos *frameworks* implementan una arquitectura MVC (aunque no exactamente de la misma forma, siendo además la de Struts más sofisticada) y una serie de etiquetas propias que implementan componentes de la interfaz (aunque los componentes JSF llevan mucha ventaja a las *taglibs* de Struts). Al ofrecer Struts y JSF funcionalidades distintas en algunos aspectos es de esperar que ambos convivan durante un tiempo. De hecho, el líder de la especificación JSF es Craig McClanahan, el desarrollador original y principal responsable de Struts. Fuera del mundo J2EE, el competidor principal de JSF es la arquitectura de componentes web de ASP.NET. De hecho, muchos ven en la versión actual de JSF la respuesta de Sun a esta tecnología de Microsoft.

1. Configurar una aplicación para JSF

JavaServer Faces se ha implementado como un conjunto de librerías Java, servlets, taglibs y ficheros de configuración XML que hay que colocar en un contenedor web Java, como por ejemplo Tomcat. Sun ha desarrollado dos especificaciones: JSF 1.1 y JSF 1.2. Nosotros vamos a trabajar en estas sesiones con la versión 1.1, en concreto la implementación de referencia denominada *JavaServer Faces v1.1.01 Reference Implementation*. Para que una

aplicación JSF funcione correctamente es necesario instalar también las JSTL. En concreto, vamos a trabajar con los ficheros JAR de la distribución de Apache Jakarta *jakarta-taglibs-standard-1.1.2*: `jstl.jar` y `standard.jar`.

1.1. Las librerías y taglibs

La implementación de referencia está compuesta de dos librerías: `jsf-api.jar` y `jsf-impl.jar` que contienen las clases básicas del *framework* y el servlet que se encarga del procesamiento de las páginas JSF.

Además JSF se apoya en varias librerías del proyecto Apache, concretamente `commons-beanutils.jar`, `commons-collections.jar`, `commons-digester.jar` y `commons-logging.jar`. Dichas librerías se incluyen dentro de la distribución de JSF, aunque también pueden conseguirse en la web del proyecto Apache (`http://jakarta.apache.org`).

Por último, se necesitan también los ficheros JAR que dan soporte a las JSTL: `jstl.jar` y `standard.jar`.

- `jsf-api.jar`
- `jsf-impl.jar`
- `commons-beanutils.jar`
- `commons-collections.jar`
- `commons-digester.jar`
- `commons-logging.jar`
- `jstl.jar`
- `standard.jar`

Todos estos ficheros deberán colocarse en el directorio `WEB-INF/lib` de la aplicación web.

Puedes encontrar estos ficheros en cualquiera de los ejemplos que hay en los ejercicios de esta sesión.

1.2. Los ficheros de configuración

Además de los ficheros JAR vistos en el apartado anterior, para configurar una aplicación Web JSF son necesarios los ficheros:

- `web.xml`
- `faces-config.xml`

Todo el procesamiento de las páginas basadas en JavaServer Faces lo realiza un servlet de la

clase `javax.faces.webapp.FacesServlet`. Por tanto, el primer paso para configurar nuestra aplicación JSF será redirigir las peticiones de las páginas JSF a este servlet. Esto podemos hacerlo en el fichero `web.xml` de la aplicación. La convención habitual es mapear todas las URLs de la forma `*.jsf` hacia este servlet. El siguiente código muestra un ejemplo de un fichero `web.xml` que contiene todo lo necesario para que JSF funcione correctamente:

```
<?xml version='1.0' encoding='UTF-8'?>

<!DOCTYPE web-app PUBLIC
  "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

    <display-name>Mini aplicación JSF</display-name>
    <description>
        Plantilla mínima de aplicación JSF
    </description>

    <context-param>
        <param-name>com.sun.faces.validateXml</param-name>
        <param-value>true</param-value>
        <description>
            Set this flag to true if you want the JavaServer Faces
            Reference Implementation to validate the XML in your
            faces-config.xml resources against the DTD.  Default
            value is false.
        </description>
    </context-param>

    <!-- Faces Servlet -->
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup> 1 </load-on-startup>
    </servlet>

    <!-- Faces Servlet Mapping -->
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.jsf</url-pattern>
    </servlet-mapping>

    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

La aplicación JSF se configura mediante un fichero adicional en formato XML dentro de la

carpeta `WEB-INF`, con el nombre `faces-config.xml`. Un ejemplo es el siguiente:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE faces-config PUBLIC
  "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.1//EN"
  "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">

<faces-config>
  <managed-bean>
    <managed-bean-name>userBean</managed-bean-name>
    <managed-bean-class>es.ua.jtech.jsf.login.UserBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>


 <navigation-rule>
    <from-view-id>/registro.jsp</from-view-id>
    <navigation-case>
       <from-outcome>registro-OK</from-outcome>
       <to-view-id>/welcome.jsf</to-view-id>
    </navigation-case>
 </navigation-rule>

  <navigation-rule>
     <from-view-id>/login.jsp</from-view-id>
     <navigation-case>
        <from-outcome>login-OK</from-outcome>
        <to-view-id>/welcome.jsf</to-view-id>
     </navigation-case>
     <navigation-case>
        <from-outcome>bad-login</from-outcome>
        <to-view-id>/badLogin.jsp</to-view-id>
     </navigation-case>
  </navigation-rule>

</faces-config>
```

Más adelante veremos qué se puede configurar dentro de este fichero.

1.3. Páginas JSF

Las páginas JSF son páginas JSP en las que se utiliza las taglibs proporcionadas por JSF. Al ser páginas JSP, deben tener la extensión `.jsp`. Cuando al servlet de JSF le llega una petición de una página (por ejemplo `login.jsf`) busca la página JSP correspondiente (`login.jsp`) y la procesa.

Un ejemplo de página JSF, es el siguiente fichero `login.jsp`

```
<html>
   <head> <title>Login</title> </head>
```

```
    <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
    <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
    <body>
    <h3>Página de login</h3>
   <f:view>
   <h:form>
      Nombre: <h:inputText value="#{userBean.name}"/> <br/>
      Contraseña: <h:inputSecret value="#{userBean.contraseña}"/>
      <h:commandButton value="Login" action="#{userBean.doLogin}"/>
      <br/>
  </h:form>
  </f:view>
  </body>
</html>
```

Como se ve, la página JSF utiliza una serie de componentes de interfaz de usuario en forma de etiquetas propias (`inputText`, `inputSecret`, `commandButton`). Es trabajo de JSF convertir estas etiquetas a etiquetas HTML.

Los componentes de la vista están asociados a datos de un bean Java (*backing bean*) a través del atributo `value`. Por ejemplo, el valor del componente `inputText` está asociado a la propiedad `login` del *bean* Java llamado `userBean`. Estos *backing bean* se definen en el fichero `faces-config.xml`. En JSF existe un lenguaje de expresiones (EL) parecido al de JSP, pero las expresiones vienen precedidas de un "#" en lugar de un "$". No se ha utilizado directamente el EL de JSP para independizar JSF en la medida de lo posible de la tecnología empleada para la vista. Como se ve, también se pueden asociar componentes al "disparo" de métodos java, como en el caso del atributo `action` del botón `commandButton`.

2. El ciclo de proceso de una página JSF

Supongamos que el navegador accede por primera vez a la página anterior. El servlet que procesa las páginas JSF irá pasando por las siguientes fases de procesamiento:

- **Construcción del árbol de componentes**: se construye un árbol de objetos de la clase `UIComponent`, que representa la estructura de componentes web de la página JSF.
- **Enlace con el modelo (*backing bean*)**: cada componente toma su valor de la propiedad del backing bean asociada. Si el bean no existe, se crea, basándose en información contenida en el fichero `faces-config.xml`.
- **Generación de la vista**: cada componente JSF es el responsable de "mostrarse" en la vista (en este caso, como etiquetas HTML) con los valores obtenidos en el paso anterior.

Cuando el usuario pulsa sobre un componente que dispara alguna acción (en nuestro caso el `commandButton`) se hace otra llamada al *servlet* que procesa las páginas JSF. En la web, los componentes que disparan acciones serán botones de tipo *submit* o bien enlaces. en este

caso, las fases de proceso serán las siguientes:

- **Aplicación de los valores de la petición**: se asigna a cada componente del árbol de objetos el valor asociado en la petición HTTP (en nuestro caso, los valores que el usuario haya introducido en los campos del formulario). Si el componente tiene asociada una propiedad del modelo con un tipo distinto a String, JSF intentará hacer la conversión (ya que los parámetros HTTP son todos Strings).
- **Validación:** se comprueba que los valores asignados sean válidos según las condiciones especificadas por el usuario (en el ejemplo anterior, con el atributo `required` se especifica que el componente `inputText` debe contener algún valor). Si en esta fase se produce algún error, se termina el procesamiento.
- **Actualización de los valores del modelo**: si los componentes están vinculados a algún *bean* de Java, los valores asignados a los componentes pasan a asignarse a dicho *bean*. En el ejemplo anterior, se asignarían a las propiedades `login` y `password` del objeto `usuario`.
- **Invocación de la aplicación:** Recordemos que todo este ciclo se ha disparado a consecuencia de un componente (en el ejemplo un `commandButton`) que llevaba asociada una acción (en el ejemplo el método `userBean.doLogin()`). En esta fase se dispara la acción propiamente dicha. El desarrollador debe colocar el dicho método la "lógica de negocio" de su aplicación. Este método devolverá una cadena que, junto con la configuración en `faces-config.xml` decidirá la siguiente página a mostrar.
- **Generación de la respuesta**: finalmente, JSF muestra la página deseada. Si se han producido errores, se mostrará de nuevo la misma página que teníamos.

3. Componentes para la interfaz de usuario

Una de las partes fundamentales de JSF es su biblioteca de componentes para la creación de interfaces web. Como se ha visto en el ejemplo anterior, dichas componentes están disponibles en forma de etiquetas propias. Para usarlas es necesario importar dos *taglibs* de JSF, la `core` y la `html`.

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
```

Todas las etiquetas JSF deben venir dentro de la etiqueta `view` del *taglib core*. Esta etiqueta actúa como la raíz del árbol de componentes JSF. En una página solo puede haber un único `view`.

```
<f:view>
 ...
</f:view>
```

Cada componente JSF es de un tipo y tiene una forma de representarse en la vista. El nombre

de la mayor parte de componentes sigue la convención de unir ambos elementos. Por ejemplo, entre los tipos de componentes están los de entrada (`input`). Un componente de entrada puede mostrarse como texto visible (`text`) o bien como texto oculto (`secret`). Así, por ejemplo un `<h:inputText>` sería un campo de entrada con texto visible (que en HTML se traduciría en un `<input type="text">`) mientras que un `<h:inputSecret>` sería un campo de entrada con texto oculto (en HTML un `<input type="password">`).

La lista completa de los taglibs JSF se puede consultar en la especificación de Sun.

3.1. Formularios

Engloban varios datos que deben procesarse conjuntamente. Para definirlos se utiliza la etiqueta `form` del *taglib html.*

```
  <h:form>
  ...
</h:form>
```

El nombre de este componente no sigue la notación estándar que une tipo y forma de visualización ya que solo hay una forma de ver formularios.

3.2. Comandos

Son componentes que disparan una acción cuando se activan, generalmente pulsándolos con el ratón. El ejemplo más típico de esta clase es el botón, que se define con la etiqueta `commandButton`. En HTML se traducirá generalmente en un `<input type="submit">`. Para disparar una acción mediante un enlace HTML utilizaríamos un `commandLink`. Ambos tienen un atributo `action` que asignaremos al método a ejecutar cuando se dispare la acción.

3.3. Componentes de entrada

Los componentes de entrada muestran valores y permiten al usuario modificarlos. JSF tiene un número considerable de estos componentes, que en la web se visualizan como los típicos controles de formulario HTML. La siguiente tabla muestra los tipos de componentes y los *renderer* que pueden asociarse a ellos (formas de visualizarlos).

Tipo	*Renderer*	Resultado en HTML

input	hidden	`<input type="hidden">`
	secret	`<input type="password">`
	text	`<input type="text">`
selectBoolean	checkbox	`<input type="checkbox">`
selectMany	checkbox	grupo de `<input type="checkbox">` en las celdas de una tabla
	listbox	`<select multiple>` con un atributo `size` igual al nº de opciones
	menu	`<select multiple size="1">`
selectOne	listbox	`<select>` con un atributo `size` igual al nº de opciones
	menu	`<select size="1">`
	radio	grupo de `<input type="radio">` en las celdas de una tabla

Así, por ejemplo un `<h:selectOneMenu>` generaría un cuadro de listado desplegable. Como se ve, muchos de estos componentes JSF son bastante sencillos y tienen una correspondencia 1:1 con etiquetas HTML. No obstante, algunos son de un nivel un poco más alto: por ejemplo `<h:selectManyCheckbox>` genera varias etiquetas HTML. Los `selectOne` y `selectMany` permiten especificar cada opción de forma individual con un elemento `<f:listItem>` o bien todas de una vez con `<f:listItems>`. En este último caso el *value* debe estar enlazado con un array, Collection o Map.

3.4. Componentes de salida

Estos componentes muestran valores no modificables. El caso más típico es la etiqueta `<h:outputText>`, que se utiliza para mostrar texto. La siguiente tabla muestra algunos componentes de salida

Tipo	Renderer	Resultado en HTML

output	link	`<a href="...">`, donde el enlace apunta donde especifica el atributo `value`
	text	texto

4. Creación de los backing beans

Los backing beans contienen los datos que "rellenan" los componentes JSF. No hay que confundirlos con los datos del modelo propiamente dicho de la aplicación web (la capa de datos definida por JDBC o por Hibernate). Los backing beans se deben comunicar con los objetos del modelo mediante la capa de negocio, pero esto es tarea del programador ya que el framework JSF no proporciona ninguna funcionalidad para ello.

Los backing beans se representan mediante clases Java convencionales, y no tienen por tanto ninguna dependencia del API de JSF. Lo único que hay que tener en cuenta es utilizar la convención JavaBean para el acceso/cambio de las propiedades (métodos `getXXX/setXXX`).

Para utilizar una clase en JSF habrá que hacer tres cosas

- Definir la clase Java, utilizando métodos `get/set` para el acceso a las propiedades
- Instanciar uno o varios objetos de dicha clase para que los usen las páginas JSF. Esto normalmente se hace en el fichero `faces-config.xml`
- Vincular las propiedades del objeto instanciado a los componentes del interfaz de usuario. Los métodos se pueden vincular a comandos.

4.1. Definir la clase del bean

El ejemplo anterior utiliza un objeto llamado `userBean` que es una instancia de la clase `UserBean` definida a continuación:

```
package es.ua.jtech.jsf.login;

public class UserBean {
        private String name;
        private String password;

        public String getPassword() {
                return password;
        }

        public void setPassword(String password) {
                this.password = password;
```

```
        }

        public String getName() {
                return name;
        }

        public void setName(String name) {
                this.name = name;
        }

        public String doLogin() {
           if //login correcto
              return "login-OK"
              //login erróneo
              return "bad-login";
        }

}
```

4.2. Instanciar los backing beans

Los backing beans pueden definirse en la forma estándar en JSP, mediante la etiqueta `<jsp:useBean/>` o bien en el fichero de configuración `faces-config.xml` (lo más normal en JSF). En este fichero se declaran los *beans* a instanciar y se especifica su ámbito (petición, sesión o aplicación). En el siguiente ejemplo se instancia un bean llamado `usuario`, de la clase `UserBean` con ámbito de petición:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE faces-config PUBLIC
  "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.1//EN"
  "http://java.sun.com/dtd/web-facesconfig_1_1.dtd">

<faces-config>
  <managed-bean>
    <managed-bean-name>userBean</managed-bean-name>
    <managed-bean-class>es.ua.jtech.jsf.login.UserBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>

   ...

</faces-config>
```

Cuando una página JSF haga referencia a un objeto llamado `userBean` si no existe se creará automáticamente con el ámbito especificado.

4.3. Vincular las propiedades del bean a la interfaz de usuario

La vinculación entre la propiedad del *bean* y el componente de la interfaz se establece con el

atributo `value` de este último. Por ejemplo, para vincular la propiedad `password` del objeto `usuario` al componente `inputSecret` se hacía:

```
<h:inputSecret value="#{userBean.password}" />
```

Como se ve, la sintaxis utilizada es la misma del EL de JSP, pero con un símbolo # delante en lugar del $.

5. Acciones y navegación entre páginas

Otra de las características fundamentales del framework JSF es la facilidad de codificar los manejadores de eventos del usuario (pulsar en botones o enlaces) y la navegación entre páginas. Vamos a verlo.

5.1. Disparar una acción

Los componentes `commandButton` y `commandLink` pueden disparar acciones en la aplicación. En el atribution `action` de estas etiquetas se define el método al que hay que llamar cuando el usuario pulsa el botón o el enlace. En el ejemplo que venimos utilizando, el código asociado a la pulsación del botón "login" es el método `doLogin` del backing bean `userBean`

```
<h:commandButton value="login" action="#{userBean.doLogin}"/>
```

El método invocado a través del atributo `action` debe devolver un `String` que es un nombre simbólico para el resultado de la acción (por ejemplo, "login-OK", o "error"). En función de dicho nombre y de las reglas de navegación definidas en `faces-config.xml` se decide la siguiente página JSP a mostrar.

También es posible en una acción no llamar a código Java, sino simplemente devolver una cadena como resultado de la "acción" para mostrar otra página según las reglas de navegación. En este caso se pone la cadena como valor del atributo `action`.

```
<h:commandButton value="cancel" action="cancel"/>
```

5.2. Decidir cuál es la página destino

En el fichero `faces-config.xml` se puede definir a qué página hay que ir en función de hasta tres factores

- La página en la que estamos en el momento actual
- La acción ejecutada

- El resultado devuelto por dicha acción

Esto se define mediante un elemento `<navigation-rule>`. Por ejemplo, para indicar que, partiendo de `login.jsp` hay que realizar la petición JSF `personal.jsf` si todo va bien o mostrar `error.jsp` en caso de problemas, podríamos poner:

```
<navigation-rule>
   <from-view-id>/login.jsp</from-view-id>
   <navigation-case>
      <from-action>#{userBean.doLogin}</from-action>
      <from-outcome>OK</from-outcome>
      <to-view-id>/personal.jsf</to-view-id>
   </navigation-case>
   <navigation-case>
      <from-action>#{userBean.doLogin}</from-action>
      <from-outcome>error</from-outcome>
      <to-view-id>/error.jsp</to-view-id>
   </navigation-case>
</navigation-rule>
```

Como se ve en el ejemplo anterior, una regla de navegación especifica la página de la que se parte (`<from-view-id>`) y puede contemplar uno más casos (`<navigation-case>`) correspondientes a los resultados de una o varias acciones. Cada caso está compuesto por:

- `<from-action>` (opcional): El nombre de la acción. Si no se pone, la regla se aplica sea cual sea la acción ejecutada.
- `<from-outcome>` (opcional): el resultado devuelto por la acción. Si no se pone, la regla se aplica sea cual sea el resultado.
- `<to-view-id>` (obligatorio): página destino (terminada en la extensión .jsf si es una petición JSF o .jsp o .html).

6. Validación de datos y control de errores

Podemos considerar dos tipos de validación: *sintáctica* y *semántica*. La *sintáctica* sería verificar que los datos pueden convertirse al tipo apropiado (al de la propiedad del bean enlazado con el componente JSF). La *semántica* exigiría verificar que además cumplen una serie de condiciones impuestas por los requerimientos de la aplicación (por ejemplo, que un password debe tener 6 o más caracteres).

En JSF, el primer tipo de validación se lleva a cabo cuando se asignan los valores de la petición HTTP a los componentes del modelo. Para llevar a cabo el segundo tipo es necesario que el desarrollador asigne **validadores** a los componentes. JSF trae por defecto implementados algunos validadores, aunque el desarrollador puede implementar los suyos propios, ya que no son más que clases que implementan la interfaz `Validator`.

6.1. Validación sintáctica: conversores

Normalmente la conversión al tipo adecuado se hace de manera automática. Para ciertos casos podemos especificar cuál es el formato del valor de entrada para que JSF sea capaz de convertirlo. Casos típicos son las fechas o los números con formato especial.

El conversor de fechas se coloca con la etiqueta `<f:convertDateTime>` dicha etiqueta tiene un atributo `dateStyle` y otro `timeStyle` que pueden tomar los valores `default`, `long`, `medium`, `short` o `full`, correspondiéndose con las constantes del mismo nombre de la clase `DateFormat` del API de J2SE. El atributo `pattern` permite especificar el formato utilizando los símbolos admitidos por `SimpleDateFormat`. Algo similar ocurre con el conversor de números: `<f:convertNumber>` admite los mismos formatos que `NumberFormat`. Para una explicación más exhaustiva, consultar alguna referencia de las taglibs de JSF.

Así, por ejemplo, para especificar que un determinado campo es una fecha en formato dd/mm/aaaa y debe asociarse con la propiedad `nacimiento` del bean `usuario`, haríamos algo como:

```
<h:inputText value="#{userBean.fechaNacimiento}">
   <f:convertDateTime pattern="dd/mm/yyyy"/>
</h:inputText>
```

El campo "fechaNacimiento" en el backing bean debe ser de la clase `java.util.Date`.

6.2. Validación semántica: validadores

En la página los validadores se especifican mediante etiquetas o mediante atributos de los componentes. Por ejemplo, para exigir que un dato no pueda ser vacío se utiliza el atributo `required` con valor true. Por otro lado, para controlar la longitud de un campo se utiliza la etiqueta `validateLength` con los atributos `minimum` (nº mínimo de caracteres) y `maximum` (nº máximo).

Así, para especificar que tanto login como password son obligatorios y que este último debe tener un mínimo de 6 caracteres se haría:

```
...
<h:inputText value="#{userBean.name}" required="true"/> <br>
<h:inputSecret value="#{userBean.password}" required="true">
   <f:validateLength minimum="6"/>
</h:inputSecret>
...
```

La siguiente tabla muestra los validadores estándar definidos en JSF. Todos ellos admiten los

atributos minimum y maximum. El usuario puede definir sus propios validadores, aunque no veremos aquí cómo se hace.

Etiqueta	Validador
`<f:validateDoubleRange>`	rango para valores reales
`<f:validateLength>`	longitud para cadenas
`<f:validateLongRange>`	rango para valores enteros

6.3. Mostrar los errores de los datos introducidos

Como se ha comentado anteriormente, si se produce algún error en la fase de validación, se vuelve a la misma página. Para mostrar los errores asociados a un componente se puede utilizar la etiqueta `<h:message>`. Para ello es necesario asociar un identificador al componente (mediante el atributo `id`) y poner el mismo valor como atributo `for` de `<h:message>`. La etiqueta para mostrar el error puede colocarse en cualquier parte de la página, aunque lo habitual es ponerla junto al componente que ha generado el error.

```
<h:inputText value="#{usuario.login}" required="true" id="login"/>
<h:message for="login"/>
```

La etiqueta `<h:messages>` muestra juntos los errores de todos los componentes.

6.4. Definir mensajes de error propios

Los mensajes de error que muestra la etiqueta `<h:message>` son genéricos y por tanto pueden no ser muy adecuados para nuestra aplicación. Podemos definir nuestros propios mensajes en un fichero .properties en cualquier lugar del *classpath* de nuestra aplicación (por ejemplo en `WEB-INF/classes`). Cada mensaje se declara con una clave, un igual y un valor de texto. Las claves para los mensajes se definen en la especificación JSF y se corresponden con el nombre de la clase Java que implementa el validador seguido del atributo que "no se ha cumplido". Por ejemplo:

```
javax.faces.validator.DoubleRangeValidator.MINIMUM=Por favor, introduzca un
valor mayor o igual a {0}
```

El valor `{0}`lo sustituye JSF automáticamente por el valor introducido en el componente y que ha generado el error.

Para que JSF tome el fichero `.properties` hay que poner su nombre dentro de la etiqueta `<message-bundle>` en el fichero `faces-config.xml`. Por ejemplo, para decir que nuestros mensajes están en un fichero llamado `mensajes.properties` dentro del

classpath haríamos algo como

```
<faces-config>
   <application>
      <message-bundle>mensajes</message-bundle>
   </application>
</faces-config>
```

7. El estado actual de JSF

En la actualidad JSF es una tecnología todavía emergente. Aunque ya cuenta con una especificación inicial y existen algunas herramientas IDE de ayuda para el desarrollo de aplicaciones basadas en ella, todavía carece del grado de madurez e implantación de otros *frameworks* MVC para Web como Struts.

Sin embargo, se trata de una tecnología que hay que tener presente, debido a distintos aspectos:

- Hay que tener en cuenta que la especificación actual de JSF 1.1 (JRS 127) se aprueba en marzo de 2004, con lo que no tiene ni tan siquiera dos años de presencia estable. Frente a esto, Struts fue donado a la fundación Apache en mayo de 2000 y tiene, por tanto, una antiguedad considerablemente mayor.
- Proporciona una solución bastante más completa que Struts para el desarrollo de aplicaciones web siguiendo el modelo MVC: permite el desarrollo de componentes a medida, el fichero de configuración `faces-config.xml` es mucho más fácil de usar que `struts-config.xml`.
- La especificación está viva y se están corrigiendo distintos problemas que se han encontrado. En concreto, la JSR 252 ha liberado ya una propuesta de borrador final que dará lugar a JSF 1.2 y que, junto con la especificación de JSP 2.1 corrige algunos problemas de interacción entre ambas tecnologías.
- Existe una gran actividad en la comunidad de desarrolladores alrededor de JSF y los componentes reusables para aplicaciones web. Proyectos como *Apache MyFaces* (http://myfaces.apache.org/, muy impresionante la reciente implementación denominada Tobago) o *icefaces* (http://www.icesoft.com/products/icefaces.html) están elaborando nuevos y más flexibles componentes que hacen que las aplicaciones web se parezcan cada vez más a aplicaciones de clientes ricos.
- En la actualidad se están comenzando a desarrollar proyectos que relacionan JSF con otras tecnologías emergentes como AJAX (*Asynchronous JavaScript and XML*, ver http://en.wikipedia.org/wiki/AJAX) o XUL (*XML User Interface Language*, ver http://en.wikipedia.org/wiki/XUL).
- La especificación JSF está abierta al soporte de otras tecnologías de interacción gráfica

con componentes (dispositivos móviles, por ejemplo).

- Por último, pero no menos importante, JSF está apoyado por Sun y por la mayoría de empresas de importancia en el mundo de J2EE como Bea, IBM, JBoss o Oracle.

En conclusión, JSF es una tecnología emergente que hay que conocer por sus previsibles expectativas de crecimiento e implantación.