Struts

Sesión 3: Validación automática. Pruebas

Indice

- **El plugin *validator*. Instalación**
- Configuración de los validadores
- Pruebas con StrutsTestCase

Plugin validator

- Permite validar automáticamente sin necesidad de programar el validate() (un poco al estilo de JSF)
  - Configurable: qué validar y cómo se especifica en un archivo XML
  - Extensible: podemos programar nuestros propios validadores
  - Puede validar también en el cliente (con JavaScript generado automáticamente)

Instalación de validator

- Importar el commons-validator.jar (incluído con la distribución de Struts)
- Indicar en el struts-config.xml que vamos a usar *validator* y cómo se llaman los 2 fich. de config.

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property property="pathnames"
          value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
</plug-in>
```

- Para validator-rules.xml se usa el que viene con Struts salvo que definamos validadores. La configuración se hace en validation.xml

Indice

- El plugin *validator*. Instalación
- **Configuración de los validadores**
- Pruebas con StrutsTestCase

Ejemplo de validation.xml

```
<!DOCTYPE form-validation PUBLIC "-//Apache Software Foundation//DTD Commons
Validator Rules Configuration 1.0//EN"
"http://jakarta.apache.org/commons/dtds/validator_1_0.dtd">
<form-validation>
 <formset>
    <form name="registro"> (ActionForm que se valida)
            <field property="login" depends="required,minlength"> (propiedad y validadores)
                    <arg0 key="registro.nombre"/>   (argumento para el mensaje de error)
                    <var>      (parámetro que se le pasa al validador)
                       <var-name>minlength</var-name>
                       <var-value>5</var-value>
                    </var>
            </field>
    ...
    </form>
  </formset>
...
</form-validation>
```

Algunos validadores predefinidos

- required: el dato no puede ser vacío (sin parámetros)
- date:fecha en formato válido
  - var-name: datePattern, datePatternStrict (String en el formato típico de DateFormat)
- mask: emparejar con una expresión regular
  - var-name: mask (la e.r.)
- intRange, floatRange, doubleRange: valor numérico en un rango
  - var-name: min, max
- maxLength: longitud máx (de cadena)
  - var-name: maxLength
- minLength: longitud mín. (de cadena)
  - var-name: minLength
- byte,short,integer,long,double,float (¿se puede convertir a…? – sin parámetros)
- email,creditcard (sin parámetros)

Mensajes de error

```
…
   <form name="registro"> (ActionForm que se valida)
          <field property="login" depends="required,minlength"> (propiedad y validadores)
                 <arg0 key="registro.nombre"/>   (argumento para el mensaje de error)
…
```

- El error se supone en el .properties bajo la clave errors.nombreDeValidador, en este caso:

```
errors.required = el campo {0} no tiene valor
errors.minlength = el campo {0} no tiene la longitud mínima
registro.nombre = login
```

Modificar el ActionForm

- Se debe heredar de ValidatorForm (o DynaValidatorForm si es dinámico)

```
public class LoginForm extends org.apache.struts.validator.action.ValidatorForm {
   …
}
```

- Ya no hace falta implementar validate()

Validación en el cliente

- Validator genera automáticamente el JavaScript

```
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
...
<html:form action="/login" onsubmit="return validateLoginForm(this)">
   ...
</html:form>
<html:javascript formName=“LoginForm"/>
```

Pruebas en Struts

- **StrutsTestCase**: capa sobre Junit y Cactus para probar acciones de Struts. Permite comprobar (entre otras cosas)
  - El forward que devuelve la acción
  - Si genera o no ActionErrors
  - Si falla la validación
- Dos enfoques de prueba
  - Mock: prueba **fuera** del contenedor web
  - Cactus: prueba **dentro** del contenedor

Indice

- El plugin *validator*. Instalación
- Configuración de los validadores
- **Pruebas con StrutsTestCase**

Ejemplo de prueba

```
Public class LoginActionTest extends CactusStrutsTestCase {
  public LoginActionTest(String testName) {
            super(testName);
  }
  protected void setUp() throws Exception {
            super.setUp();
            setRequestPathInfo("/login");
  }
  public void testLogin() {
            LoginForm unForm = new LoginForm();
            unForm.setLogin("j2ee");
            unForm.setPassword("j2ee");
            setActionForm(unForm);
            actionPerform();
            verifyForward(Tokens.SUCCESS);
            verifyNoActionErrors();
  }
```

Ejemplo de prueba (y II)

```
…
  public void testLoginKO() {
            // Configuramos el formulario
            LoginForm unForm = new LoginForm();
            unForm.setLogin("fdsagdsa");
            unForm.setPassword("gdsahaas");
            setActionForm(unForm);
            // Llamada al Action ;
            // Comprobamos que redirige al InputForward
            verifyInputForward();
            // Comprobamos que hay errores
            verifyActionErrors(new String[] {Tokens.ERROR_KEY_LOGIN_KO});
  }
```

Casillas de verificación

- Poner las opciones “a mano”

Una sola casilla:
```
<html:checkbox property="publi"/> Sí, deseo recibir publicidad sobre sus productos
```

Varias agrupadas:
```
<html:multibox property="aficiones">cine</html:multibox> Cine
<html:multibox property="aficiones">música</html:multibox> Música
```

- Generarlas automáticamente a partir de un getXXX()
  - Muy parecido a los botones de radio

```
<%@ taglib uri="http://struts.apache.org/tags-html-el" prefix="html-el" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
...
<c:forEach items="${FormRegistro.listaAficiones}" var=“a">
        <html-el:multibox property=“aficiones“> ${a} </html-el:multibox> ${a}
</c:forEach>
```

¿Preguntas...?