Struts: la vista

Índice

En este tema trataremos el interfaz entre la vista y las acciones en Struts: los denominados `ActionForms` y la implementación de la vista mediante las librerías de etiquetas incluidas en Struts.

1. ActionForms

1.1. Introducción

Aunque los datos introducidos en formularios pueden obtenerse dentro del código de las acciones directamente de la petición HTTP (como hemos visto en los ejemplos del tema anterior), Struts ofrece un mecanismo alternativo que proporciona distintas ventajas: los **ActionForm**s. Empleando ActionForms podemos conseguir:

- **Recolección automática de datos** a partir de los incluidos en la petición HTTP.
- **Validación de datos modular** (realizada separadamente a la acción), y en caso de utilizar el plugin *Validator*, validación automática según lo especificado en un fichero de configuración.
- **Recuperación de los datos** para volver a rellenar formularios. De esta forma se evita el típico problema de que el usuario tenga que volver a rellenar un formulario entero porque uno de los datos es incorrecto.

Podemos considerar un ActionForm como si fuera un JavaBean que captura los datos de un formulario. Los datos se pueden extraer, validar, cambiar y volver a colocar en otro formulario. No obstante, no tiene por qué haber una correspondencia uno a uno entre un ActionForm y un formulario HTML, de manera que se puede utilizar el mismo ActionForm para englobar varios formularios separados en distintas páginas (caso típico de un asistente). También se puede reutilizar el mismo ActionForm en distintos formularios que compartan datos (por ejemplo, para dar de alta o modificar los datos de un usuario registrado).

En Struts se pueden definir básicamente dos tipos distintos de ActionForms

- Clases descendientes de la clase base **ActionForm**. Deben incorporar métodos Java para obtener/cambiar cada uno de los datos (al estilo JavaBeans), cuya definición puede resultar tediosa. Desde Struts 1.1 las propiedades se pueden almacenar en un Map, con lo que solo es necesario definir un único método para acceso a todas las propiedades, pasando como parámetro el nombre de la deseada.
- Instancias de la clase **DynaActionForm** o descendientes de ésta. Permiten definir los campos en el fichero `struts-config.xml`, de manera que se pueden añadir o modificar campos minimizando la necesidad de recompilar código. Mediante la clase **DynaValidatorForm** se puede incluso validar datos de manera automática, según las reglas especificadas en un fichero de configuración aparte.

1.2. El ciclo de vida de un ActionForm

La generación y procesamiento de un ActionForm pasa por varias etapas:

1. El controlador recibe la petición del usuario, y chequea si la acción asociada utiliza un ActionForm. De ser así, crea el objeto
2. Se llama al método `reset()` del objeto, que el desarrollador puede sobreescribir para "limpiar" sus campos. Esto tiene sentido si el ActionForm persiste más allá de la petición actual (lo cual se puede especificar al definirlo).
3. El ActionForm se almacena en el ámbito especificado en su definición (petición, sesión o aplicación)
4. Los datos del ActionForm se rellenan con los que contiene la petición HTTP. Cada parámetro HTTP se asocia con el dato del mismo nombre del ActionForm. Un punto importante es que en HTTP los parámetros son cadenas, con lo que en principio las propiedades del ActionForm deben ser Strings, aunque los valores booleanos se convierten a boolean. No obstante, más allá de esta conversión básica no hay conversión automática de tipos.
5. Se validan los datos, llamando al método `validate()`, que el desarrollador debe sobreescribir para implementar la validación deseada.
6. Si se han producido errores de validación, se efectúa una redirección a la página especificada para este caso. Si no, se llama al `execute()` de la acción y finalmente se muestra la vista asociada a esta.
7. Si en la vista se utilizan las *taglibs* de Struts para mostrar datos, aparecerán los datos del ActionForm.

1.3. Declarar un ActionForm (en el fichero de configuración)

Los ActionForm se definen dentro del fichero `struts-config.xml`, dentro de la sección `<form-beans>`. Cada ActionForm viene definido por un elemento `<form-bean>`. Por ejemplo, para definir un ActionForm con el nombre `FormLogin` y asociado a la clase Java `acciones.forms.FormLogin` haríamos:

```
<form-beans>
   <form-bean name="FormLogin" type="acciones.forms.FormLogin">
</form-beans>
```

Para que los datos de un `ActionForm` sean accesibles a una acción, hay que definir una serie de atributos dentro del elemento `action` de `struts-config.xml`. Por ejemplo, para asociar un `ActionForm` de la clase `FormLogin` a la acción `login` de los ejemplos anteriores, se podría hacer:

```
<action path="/login" type="acciones.AccionLogin"
```

```
        name="FormLogin" scope="session"
        validate="true" input="/index.jsp">
   <forward name="OK" path="/personal.jsp"/>
   <forward name="errorUsuario" path="/error.html"/>
</action>
```

El atributo `name` indica el nombre simbólico para el `ActionForm`. El `scope` tiene el mismo significado que cuando tratamos con JavaBeans. En caso de que se desee validar los datos, hay que especificar el atributo `validate=true` y utilizar el atributo `input` para designar la página a la que se volverá si se han producido errores de validación.

1.4. Declarar la clase java del ActionForm

Esta tarea es muy similar a definir un JavaBean. Hay que especificar métodos get/set para cada campo, y colocar la lógica de validación en el método `validate()`. Por ejemplo:

```
package acciones.formularios;
import org.apache.struts.action.*;
import javax.servlet.http.HttpServletRequest;

public class FormLogin extends ActionForm {
   private String login;
   private String password;

   public void setLogin(String login) {
      this.login = login;
   }
   public String getLogin() {
      return login;
   }

   public ActionErrors validate(ActionMapping mapping,
                                HttpServletRequest request) {
      if ((getlogin()==null)||(getLogin.equals("")))
         errores.add(ActionErrors.GLOBAL_ERROR
                     , new ActionError("error.requerido.usuario"));
      return errores;
   }
}
```

El método `validate()` debe devolver una colección de errores (o null), representada por el objeto **ActionErrors**.

Desde la versión 1.1 de Struts, se pueden utilizar ActionForms que almacenen internamente las propiedades en un objeto Map. Para el acceso a/modificación de cualquier propiedad solo será necesario implementar dos métodos

```
public void setValue(String clave, Object valor)
public Object getValue(String clave)
```

En formularios con muchos campos, resultará tedioso definir métodos *get/set* para cada uno de ellos. En su lugar, podemos utilizar `DynaActionForms`, en los que los campos se definen en el fichero de configuración de Struts. Por ejemplo:

```
 <form-bean name="FormLogin"
type="org.apache.struts.action.DynaActionForm">
   <form-property name="login" type="java.lang.String"/>
   <form-property name="password" type="java.lang.String"/>
 </form-bean>
```

Como se ve, en este caso la clase del ActionForm no la definimos nosotros sino que es propia de Struts. Para acceder a los campos del `DynaActionForm` esta clase define un método genérico **get** al que se pasa como parámetro el nombre del campo. Algo similar ocurre para cambiar el valor de un campo (método **set**).

Si deseamos validar un `DynaActionForm`, tendremos que utilizar una clase propia que herede de ésta, y sobreescribir su método `validate()`, o mejor aún, podemos validar datos automáticamente utilizando el módulo de Struts denominado **validator**. No obstante el uso de validator queda fuera del ámbito de este tema, por lo que se aconseja la consulta de la documentación de Struts o de algún libro de la bibliografía sobre el tema.

2. Las librerías de etiquetas de Struts

Una vez se ha ejecutado la lógica de negocio y el controlador ha pasado el control a la vista que corresponda, ésta debe mostrar los resultados de la acción. Si dichos resultados se han colocado en *beans*, será sencillo mostrarlos en la página utilizando las etiquetas estándar de `<jsp:useBean/>` o las librerías de etiquetas propias de Struts. Estas etiquetas están especialmente diseñadas para trabajar con ActionForms, aunque pueden funcionar con cualquier *bean*. De este modo, se pueden mostrar los resultados de la acción en forma de texto o bien en forma de campo de formulario, para que el usuario pueda realizar modificaciones y el "ciclo de vida" del ActionForm comience de nuevo (los datos del ActionForm se rellenan con la petición HTTP, se validan, los procesa la acción...).

Struts viene con varias taglibs distintas, las principales son:

- beans: acceso a JavaBeans y sus propiedades, y definición de nuevos beans
- html: para crear formularios y controles de formulario vinculados a ActionForms.
- logic: control de flujo, condicionales y repetición

La mayor parte de etiquetas propias de Struts han quedado obsoletas con la aparición de JSTL, por lo que nos centraremos en las diseñadas para formularios, que no tienen equivalente en JSTL.

2.1. Las etiquetas para formularios

Un grupo fundamental de etiquetas es el dedicado a controles de formulario, ya que a través de ellos se muestra el contenido de los ActionForms y puede modificarse. Aunque estas etiquetas están diseñadas expresamente para ActionForms pueden asociarse a cualquier bean. Así, estas etiquetas son similares a los componentes GUI que veremos que tiene JavaServer Faces, aunque no llegan a un nivel tan alto de sofisticación. La siguiente tabla muestra algunas de estas etiquetas

Etiqueta Struts	Equivalente HTML	Uso
`<html:form>`	`<form>`	Se pueden especificar atributos `name`, `type` y `scope` para indicar el nombre, clase y ámbito del bean del que se tomarán los datos. Si no se especifica nada, dichas propiedades se toman del `action-mappings` en `struts-config.xml`
`<html:text/>`	`<input type="text">`	el atributo `property` especifica la propiedad del `actionForm` a vincular. Con `name` se puede especificar de qué bean procede el dato
`<html:radio/>`	`<input type="radio">`	`property` idem al anterior. El atributo `value` se compara con el valor de la propiedad del bean para que el botón aparezca marcado o no `<html:radio property="estado" value="casado"/>`
`<html:checkbox/>`	`<input type="radio">`	`property` idem al anterior. El atributo `value` se compara con el valor de la propiedad del bean para que la casilla aparezca marcada o no
`<html:password/>`	`<input type="password"/>`	idem con `property`. El atributo `redisplay` se utiliza para que el valor del campo no se vuelva a

		rellenar. `<html:password property="password" redisplay="false"/>`
`<html:select/>` y `<html:options/>`	`<select>` y grupo de `<option>`	idem con `property` para `<html:select/>`. En el caso de `<html:options>` puede ser una `Collection`. `<html:select property="estadoCivil"> <html:options property="listaEstados"/> </html:select>`

Para poder usar alguna de estas etiquetas en una página JSP es necesario importar la *taglib* de HTML de Struts. Al principio de la página habrá que poner:

```
   <%@taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html"
%>
```

2.2. Mostrar los errores

La forma más sencilla de mostrar los errores generados durante el procesamiento de una acción es utilizar la etiqueta `<html:errors/>`. Esta etiqueta muestra los errores en el formato por defecto (letra con color rojo).

Si se desea mostrar únicamente los errores asociados a una propiedad de un bean, entonces hay que añadir el atributo `property`

```
<html:form>
   <html:text property="login"/> <html:errors property="login"/>
   ...
</html:form>
```

Para que esto funcione, la acción que ha generado el error debe haberlo añadido a la lista de errores asociándolo a dicha propiedad:

```
errors.add("login", new ActionError("error.requerido", "nombre de
usuario"));
```