Struts

Sesión 4: Introducción a Struts 2

Indice

- **Motivación**
- De Struts 1.x a Struts 2
  - Acciones
  - Actionforms
  - Taglibs
  - Internacionalización
  - Validación
- Conceptos nuevos en Struts 2

¿Por qué era necesario Struts 2?

- Actualmente, en JavaEE se asumen como básicos muchos principios que Struts 1.x no respeta
  - **"Convention over configuration"**: reducir configuración al mínimo, asumiendo siempre que existen opciones por defecto
    Como habéis visto, la sintaxis de struts-config.xml y validation.xml no es precisamente concisa. No hay opciones por defecto, todo hay que especificarlo
  - **APIs "no invasivos"**: eliminar en nuestro código, en la medida de lo posible, las dependencias del API. Evitar por ejemplo que nuestras clases deban heredar de clases propias del API
    En Struts 1.x, tanto las acciones como los *actionforms* deben heredar de clases del *framework*. Esto reduce la portabilidad de nuestro código y complica la realización de pruebas unitarias

Instalación y configuración

- Copiar los JARs necesarios a WEB-INF/lib
- Configurar un filtro de servlets en el web.xml

```
<filter>
  <filter-name>struts2</filter-name>
  <filter-class>
    org.apache.struts2.dispatcher.FilterDispatcher
  </filter-class>
</filter>
<filter-mapping>
  <filter-name>struts2</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Fichero de configuración

- **struts.xml**, en el CLASSPATH
  - Formato mucho más conciso que el de Struts 1.x
  - Se puede sustituir casi totalmente por anotaciones

```
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD
Struts Configuration 2.0//EN" "http://struts.apache.org/dtds/struts-
2.0.dtd">
<struts>
  <package name="default" extends="struts-default">
            <action ...>
              <result>...</result>
            </action> ...
 </package>
</struts>
```

Indice

- Motivación
- **De Struts 1.x a Struts 2**
  - Acciones
  - Actionforms
  - Taglibs
  - Internacionalización
  - Validación
- Conceptos nuevos en Struts 2

Acciones en Struts 1.x

- Tienen fuertes dependencias del API del framework, dificultando portabilidad y pruebas unitarias

```
import javax.servlet.http.*;
import org.apache.struts.action.*;
public class AccionLogin extends Action {
  public ActionForward execute(ActionMapping mapping, ActionForm form,
        HttpServletRequest req, HttpServletResponse resp) throws Exception {
    boolean usuarioOK;
    //obtener login y password y autentificar al usuario
    //si es correcto, poner usuarioOK a 'true' ...
    //dirigirnos a la vista adecuada según el resultado
    if (usuarioOK)
        return mapping.findForward("OK");
    else
        return mapping.findForward("errorUsuario"); } }
}
```

Acciones en Struts 2

- Única condición: el método que ejecuta la acción debe tener la signatura **String execute()**
  - **Hay cero dependencias de clases del framework**

```
package acciones;
public class AccionLogin {
   public String execute() {
      boolean usuarioOK;
      //obtener login y password y autentificar al usuario
      //si es correcto, poner usuarioOK a 'true' ...
      //dirigirnos a la vista adecuada según el resultado
      if (usuarioOK)
           return "OK";
      else
           return "errorUsuario";
    }
}
```

Configuración de la acción en el XML

- Por defecto, la URL para acceder a la acción es *nombre*.action

```
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration 2.0//EN" "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <package name="default" extends="struts-default">
   <action name="login" class="acciones.AccionLogin">
      <result name="OK">/usuario.jsp</result>
      <result name="errorUsuario">/error.jsp</result>
   </action>
  </package>
</struts>
```

ActionForms

- Son el punto más débil de Struts 1.x
  - Dependencias del API
  - Todos los campos acaban siendo de tipo String. Por ello no se pueden usar los actionforms como objetos de negocio, llevando a
    - Multiplicación del número de clases necesarias
    - Necesidad de convertir manualmente los valores
    - Necesidad de copiar los valores al objeto de negocio
- En Struts 2 no existen los actionforms. Son javabeans convencionales, sin más requisitos

"ActionForm" en Struts 2

- Cero dependencias del API de Struts

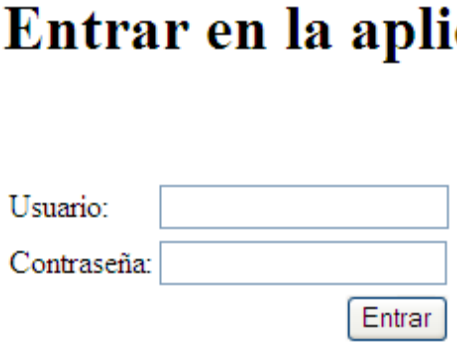
```
public class Usuario {
   private String login;
   private String password;

   public String getLogin() { return login; }
   public void setLogin(String login) { this.login = login; }
   public String getPassword() { return password; }
   public void setPassword(String password) {
      this.password = password;
   }
}
```

Referencia al "actionform" desde la acción

- **"Inyección de dependencias":** Struts llama automáticamente a setUsuario, pasándole el objeto con los campos rellenados con datos

```
import modelo.Usuario;
public class LoginAccion {
   private Usuario usuario;
   public Usuario getUsuario() { return usuario; }
   public void setUsuario(Usuario usuario) {
      this.usuario = usuario;
   }
   public String execute() {
      String login = usuario.getLogin();
      ...
   }
}
```

Tags para formularios



Así es como rellenamos los
s del bean

```
<%@taglib prefix="s2" uri="/struts-tags" %>
<html>
<head> <title>Ejemplo de taglib HTML en Struts 2</title> </head>
<body>
  Entrar en la aplicación:<br/>
  <s2:form action="login">
    <s2:textfield label="usuario" name="usuario.login"/>
    <s2:password label="contraseña" name="usuario.password"/>
    <s2:submit value="login"/>
  </s2:form>
</body>
</html>
```

Diferencias con las tags de Struts 1.x

- Las etiquetas han cambiado
- Los campos generan también un “rótulo” con texto precediéndolos
- Se puede asociar un campo a una propiedad de un bean sin más que poner nombre_bean.nombre_propiedad. Estamos usando un lenguaje de expresiones similar a EL llamado OGNL
- No es necesario poner etiquetas para mostrar errores de forma explícita. Adiós a <html:messages>

Internacionalización

- Al igual que en Struts 1.x se usan extensivamente ficheros .properties
  - En la raíz del CLASSPATH, fichero struts.properties,definimos el nombre del fichero de mensajes (ya no es en el XML)

```
struts.custom.i18n.resources=mensajes
```

  - Se pueden definir properties separados para cada acción, package, y otros

Internacionalización (II)

- Fichero mensajes.properties

```
loginTitle=Entrar en la aplicación
login=Usuario
password=Contraseña
enter=Entrar
```

- HTML

```
<%@taglib prefix="s2" uri="/struts-tags" %>
…
<s2:label key="loginTitle"/>
<s2:form action="login">
    <s2:textfield key="login" name="usuario.login"/>
    <s2:password key="password" name="usuario.password"/>
    <s2:submit key="enter"/>
</s2:form>
```

Validación

- Ya no se usa commons validator. Se simplifica mucho la sintaxis
- Se puede definir la validación de cada acción en un XML aparte (práctica recomendada)
  - Por convención, el fichero debe tener el mismo nombre que la clase, con el sufijo –validation.xml, localizado físicamente en el mismo directorio que el código de la acción

```
<!DOCTYPE validators PUBLIC "-//OpenSymphony Group//XWork Validator
1.0.2//EN" "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
  <field name="usuario.login">
      <field-validator type="requiredstring">
         <message>Login no puede estar vacío</message>
      </field-validator>
  </field>
</validators>
```

Validación (II)

- La acción debe implementar el interfaz ValidationAware para “activar” la validación
  - Es más sencillo heredar de ActionSupport, que ya tiene implementaciones por defecto para los métodos del interfaz
- En el struts.xml un error en validación se asocia con el resultado “input”

```
<action name="login"
        class="es.ua.jtech.struts2.prueba.acciones.LoginAccion">
    <result name="ok">/usuario.jsp</result>
    <result name="input">/index.jsp</result>
</action>
```

Conceptos nuevos en Struts 2

- Viendo las transparencias anteriores puede dar la impresión de que no hay ideas nuevas en Struts 2, solo se simplifica el trabajo. Nada más lejos de la realidad
  - **Interceptores**: elementos que interceptan la petición antes de que se ejecute la acción “destino”. Son los responsables por ejemplo de la inyección de dependencias y la validación. Normalmente ya están implementados, simplemente hay que configurar los que necesitemos
  - **OGNL**: un completo lenguaje de expresiones, similar a EL pero mucho más potente