

# Capítulo I

## INTRODUCCIÓN A STRUTS

Capítulo I INTRODUCCIÓN A STRUTS .....	1
1. El patrón Modelo – Vista – Controlador .....	2
2. Arquitectura de Struts .....	4
3. JavaBeans en Struts (ActionForms).....	6
4. Action Mappings .....	7
5. Trabajo con formularios .....	8
6. Validación automática en formularios .....	9

Struts es un framework de la capa de presentación que implementa el patrón de patrón MVC en Java.

Evidentemente, como todo framework intenta simplificar notablemente la implementación de una arquitectura según el patrón MVC. El mismo separa muy bien lo que es la gestión del flujo de la aplicación, del modelo de objetos de negocio y de la generación de interfaz.

## 1. El patrón Modelo – Vista – Controlador

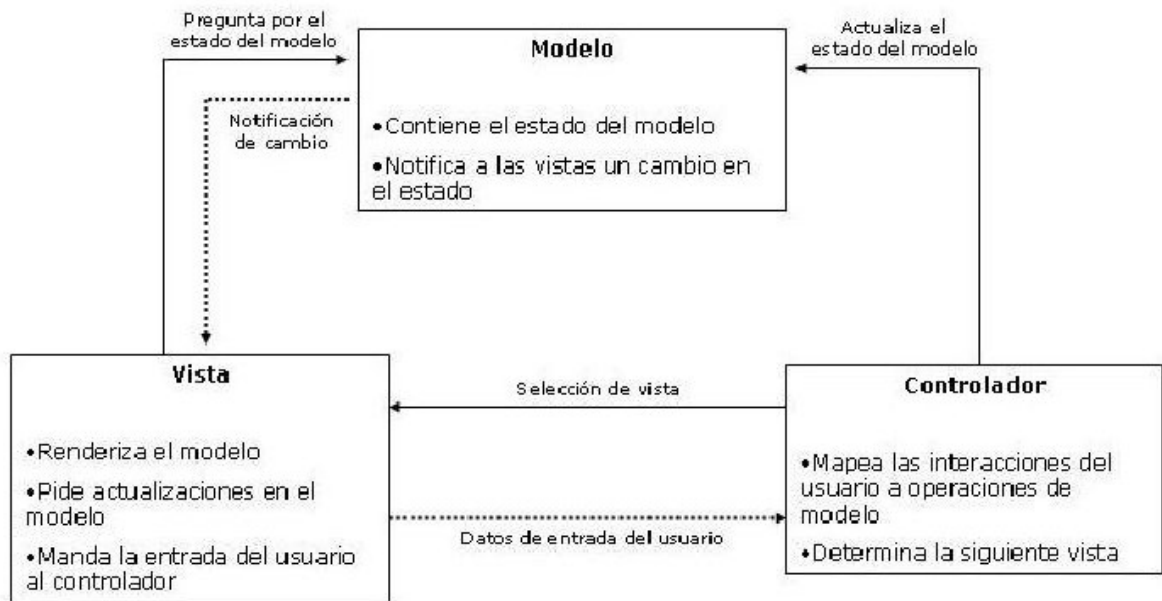
MVC (Modelo-Vista -Controlador) es un patrón de diseño aportado originariamente por el lenguaje SmallTalk a la Ingeniería del Software. Consiste principalmente en dividir las aplicaciones en tres partes:

- Controlador
- Modelo
- Vistas.

El controlador es el encargado de redirigir o asignar una aplicación a cada petición; el controlador debe poseer de algún modo, un "mapa" de correspondencias entre peticiones y respuestas que se les asignan. El modelo sería la lógica de negocio a fin de cuentas.

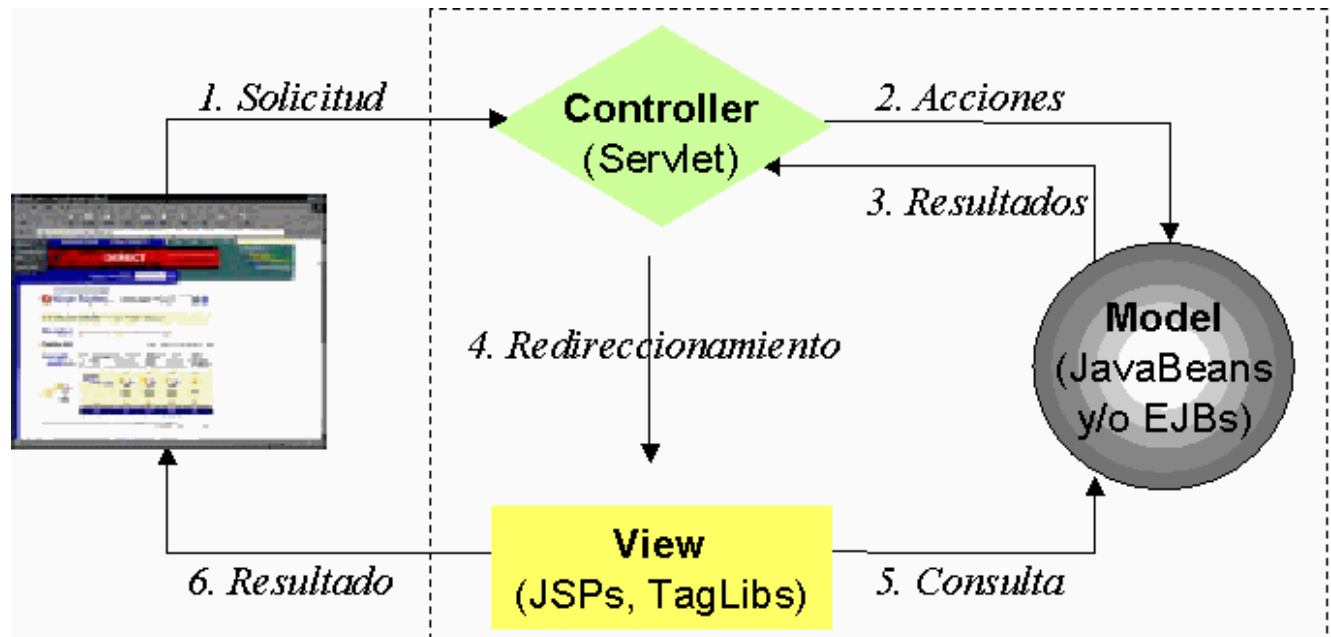
Una vez realizadas las operaciones necesarias el flujo vuelve al controlador y este devuelve los resultados a una vista asignada.

El siguiente gráfico nos muestra la interacción entre el Modelo la Vista y el Controlador.



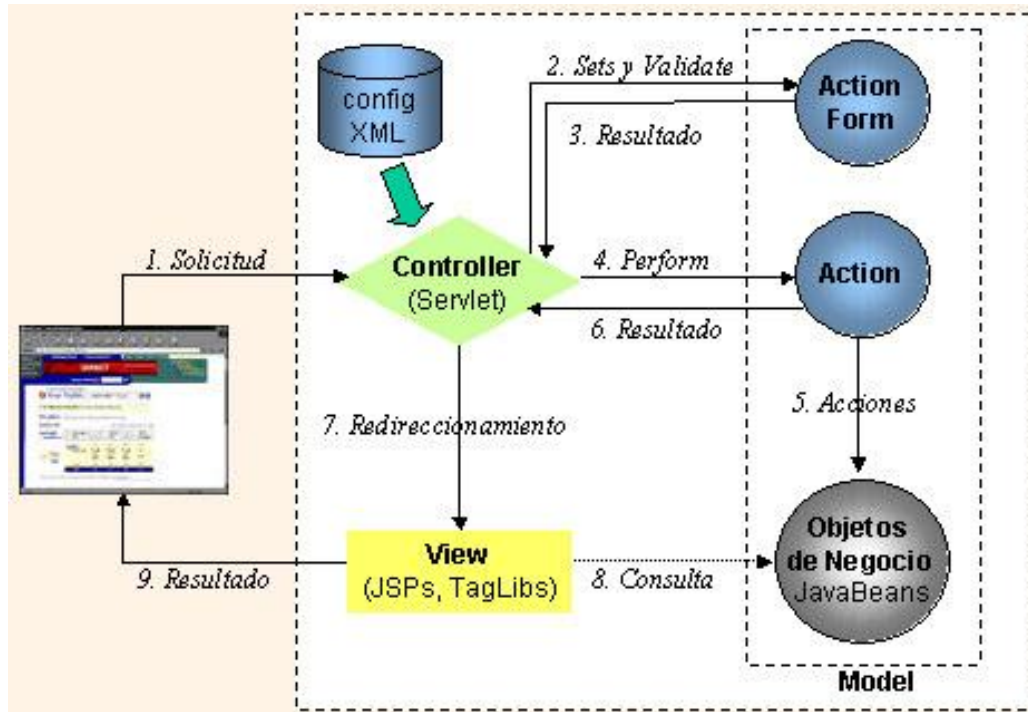
## 2. Arquitectura de Struts

El siguiente gráfico muestra la funcionalidad de Struts.



El navegador genera una solicitud que es atendida por el Controlador (un Servlet especializado). El mismo se encarga de analizar la solicitud, seguir la configuración que se le ha programado en su XML y llamar al Action correspondiente pasándole los parámetros enviados. El Action instanciará y/o utilizará los objetos de negocio para concretar la tarea. Según el resultado que retorne el Action, el Controlador derivará la generación de interfaz a una o más JSPs, las cuales podrán consultar los objetos del Modelo para mostrar información de los mismos.

Este gráfico nos provee una visión más detallada del funcionamiento de Struts



### 3. JavaBeans en Struts (ActionForms)

Una de las tareas que durante el desarrollo de una aplicación consume mucho trabajo (aunque en realidad no lo merezcan) es la interacción con formularios, ya sea para editar u obtener nueva información. Las comprobaciones, la gestión de errores, el volver a presentarle el mismo form al usuario con los valores que puso y los mensajes de error y un largo etcétera están soportadas por Struts a fines de hacernos la vida un poco más fácil.

La idea es la siguiente: todo el trabajo de comprobaciones y generación de mensajes de error se implementa en los **ActionForm** y todo el trabajo de generación de interfaz en la/s JSP.

Los JavaBeans también se pueden usar para manejar formularios de entrada. Un problema clave en el diseño de aplicaciones Web es retener y validar lo que el usuario ha introducido entre solicitudes. Con **Struts**, podemos definir un conjunto de clases bean formulario, subclasificando `org.apache.struts.action.ActionForm`, y almacenar fácilmente los datos de un formulario de entrada en estos beans formularios. El bean se graba en una de las colecciones estándar o de contexto compartidas, por eso puede ser usado por otros objetos, especialmente un objeto `Action`.

## 4. Action Mappings

Es una clase Java, en la que el `ActionServlet` delega el manejo del requerimiento y de la respuesta.

Los objetos **Action** son instancias de subclases de `org.apache.struts.action.Action`. Funcionan como miniservlets, aunque no son servlets; sus responsabilidades fundamentales son: acceder a la capa de negocios, preparar objetos para la capa de presentación y manejar errores.

El **Action** es parte de *Controlador*, por lo tanto se recomienda que el comportamiento relacionado a la lógica del negocio, como por ejemplo acceder a la bd, sea realizado por objetos de negocio (clases separadas) y no adentro del mismo Action; el Action es el lugar ideal para codificar tareas específicas de la web, por ejemplo manejo de la sesión de usr., del requerimiento, del contexto de la aplicación.

El **ActionServlet** crea una **única instancia de cada subclase de Action** por aplicación y las usa para servir a todos los requerimientos que recibe. Es un objeto **multithread**, no es un objeto **thread-safe**.

Usa la colección de objetos **ActionMappings** para determinar el **Action** que manejará cada requerimiento entrante. Finalmente, invocará al método **execute()** del Action al que le envía un conjunto de objetos útiles. Cuando el método **execute()** termina, devuelve un objeto **ActionForward**, que es usado por el ActionServlet para determinar a donde pasará el control para completar el requerimiento. Generalmente el **ActionForward** define que la próxima componente a mostrar es de presentación como una JSP aunque puede referirse a otro Action (encadenamiento de Action) o a otro recurso de la aplicación, como HTML. Si el **ActionForward** es **null**, el ActionServlet asume que el Action generó la respuesta y completó el requerimiento (en cuyo caso no hace nada).

## 5. Trabajo con formularios

Una vez u otra, la mayoría de los desarrolladores web han construido formularios usando las capacidades estándar del HTML, como la etiqueta `<input>`. Los usuarios esperan que las aplicaciones interactivas tengan ciertos comportamientos, y uno de estos está relacionado con el manejo de errores -- si el usuario comete un error, la aplicación debería permitirle corregir sólo lo que necesita ser modificado -- sin tener que re-introducir cualquier parte del resto de la información de la página o formulario actual.

Conseguir esto es tedioso y aburrido cuando codificamos usando HTML y páginas JSP. Por ejemplo, un elemento de entrada para un campo `username` podría parecerse a esto (en JSP):

```
<input type="text" name="username"
      value="<%= loginBean.getUsername() %>" />
```

lo que es difícil de teclear correctamente, confunde a los desarrolladores HTML que no tienen conocimientos sobre conceptos de programación, y puede causar problemas con editores HTML. En su lugar **Struts** proporciona una facilidad comprensiva para construir formularios, basada en la facilidad de las Librerías de Etiquetas Personalizadas de JSP 1.1. El caso de arriba sería renderizado de esta forma usando **Struts**:

```
<html:text property="username" />
```

sin la necesidad de referirnos explícitamente al formulario JavaBean del que se recupera el valor inicial. Esto lo maneja automáticamente el marco de trabajo.

Algunas veces los formularios HTML se usan para cargar otros ficheros. La mayoría de los navegadores soportan esto a través de un elemento `<input type="file">`, que genera un botón navegador de ficheros, pero es cosa del desarrollador manejar los ficheros entrantes. **Struts** maneja estos formularios "multipart" de la misma forma que los formularios normales. En la siguiente sección, usaremos **Struts** para crear un simple formulario de login, y también un simple formulario multiparte.

## 6. Validación automática en formularios

Struts ofrece una facilidad adicional para validar los campos de entrada que ha recibido. Para utilizar esta característica, sobrescribimos el siguiente método en nuestra clase ActionForm:

```
public ActionErrors validate(ActionMapping mapping, HttpServletRequest request);%
```

El método validate() es llamado por el servlet controlador después de que se hayan rellado las propiedades del bean, pero antes de se llame al método perform() correspondiente de la clase Action. El método validate() tiene las siguientes opciones:

- Realiza las validaciones apropiadas y no encuentra problemas
- Devuelve null o ejemplares de ActionErrors de longitud cero, y el servlet controlador procederá a llamar al método perform() de la clase Action apropiada.
- Realiza las validaciones apropiadas y encuentra problemas.
- Devuelve un ejemplar de ActionErrors conteniendo ActionError's, que son clases que contienen las claves del mensaje de error que se deberían mostrar. El servlet controlador almacena este array como un atributo de la solicitud disponible para usarse por la etiqueta <html:errors>, y devolverá el control al formulario de entrada.

Como se mencionó anteriormente, esta característica es totalmente opcional. La implementación por defecto de validate() devuelve null, y el servlet controlador asumirá que no se requiere que se haga ninguna validación por parte de la clase Action. Una aproximación común es realizar validaciones iniciales usando el método validate(), y luego manejar la validación de la "lógica de negocio" desde el objeto Action.

## Capítulo II

# ACTION MAPPINGS

Capítulo II ACTION MAPPINGS .....	10
1. Patrón Comando para aplicaciones web .....	11
2. ActionServlet .....	13
3. Action .....	18
3.1. Definiendo la clase Action .....	20
3.2. Los parámetros del método execute() .....	21
4. ActionMappings .....	23
5. Configuración de Struts .....	25
5.1. El archivo struts-config.xml .....	25
5.1.1. La sección <form-bean>.....	25
5.1.2. La sección <action-mapping>.....	26
5.1.3. La sección <global-forwards>.....	26
5.1.4. Internacionalización.....	27

## 1. Patrón Comando para aplicaciones web

Dentro de los patrones de diseño, el patrón Comando se define de la siguiente forma:

*Encapsula una petición como un objeto, permitiendo así parametrizar a los clientes con distintas peticiones, encolar o llevar un registro de las peticiones y poder deshacer las operaciones*

Al implementar los comandos como objetos, el patrón Comando permite a los frameworks de la aplicación invocar a los métodos de comandos específicos de la aplicación

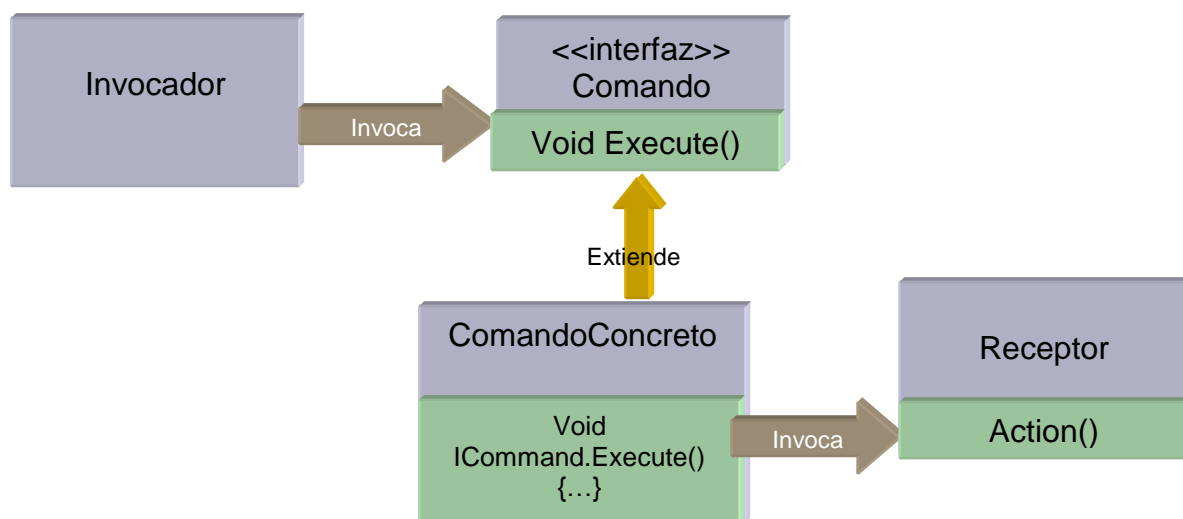


Figura 2-1 El diagrama del patrón Comando

En la figura 2-1, la clase Invocador representa un objeto en una aplicación, tal como un botón o un elemento de un menú. El invocador mantiene una asociación con un comando a través de una referencia a un objeto que implementa la interfaz Comando, la cual define un método `execute()`. Las clases comando concretas implementan esa interfaz, generalmente empleando un objeto conocido como receptor

En tiempo de ejecución, el Invocador llama al método `execute()`, y el comando invoca el método del receptor.

Este patrón de diseño también se le conoce como Action Servlet y es empleado en varios frameworks, entre ellos Struts.

## 2. ActionServlet

El componente Controlador del framework de Struts es la espina dorsal de todas las aplicaciones web de Struts y se puede considerar una fábrica de objetos Action para ejecutar la lógica requerida por la aplicación. Es implementado empleando un servlet llamado `org.apache.struts.action.ActionServlet`. Este servlet recibe todas las solicitudes de los clientes y delega el control de cada solicitud a una clase definida por el usuario del tipo `org.apache.struts.action.Action`. El ActionServlet delega el control basado en el URI de la petición de entrada. Una vez que ha sido procesada la acción, ésta regresa una llave al ActionServlet, que es empleada entonces por el ActionServlet para determinar la vista que presentará los resultados del procesamiento de la acción.

Las responsabilidades del ActionServlet se pueden resumir de la siguiente forma:

- Procesar las peticiones del usuario
- Determinar que es lo que desea hacer el usuario en base a la petición que realizó
- Obtener datos del modelo (en caso de que se requiera) para darle una vista adecuada
- Seleccionar la vista correcta para responder al usuario

El término ActionServlet y controlador es usado de forma indistinta en Struts.

A continuación se muestra un diagrama de clases en donde se puede observar la interacción del ActionServlet con las acciones asociadas.

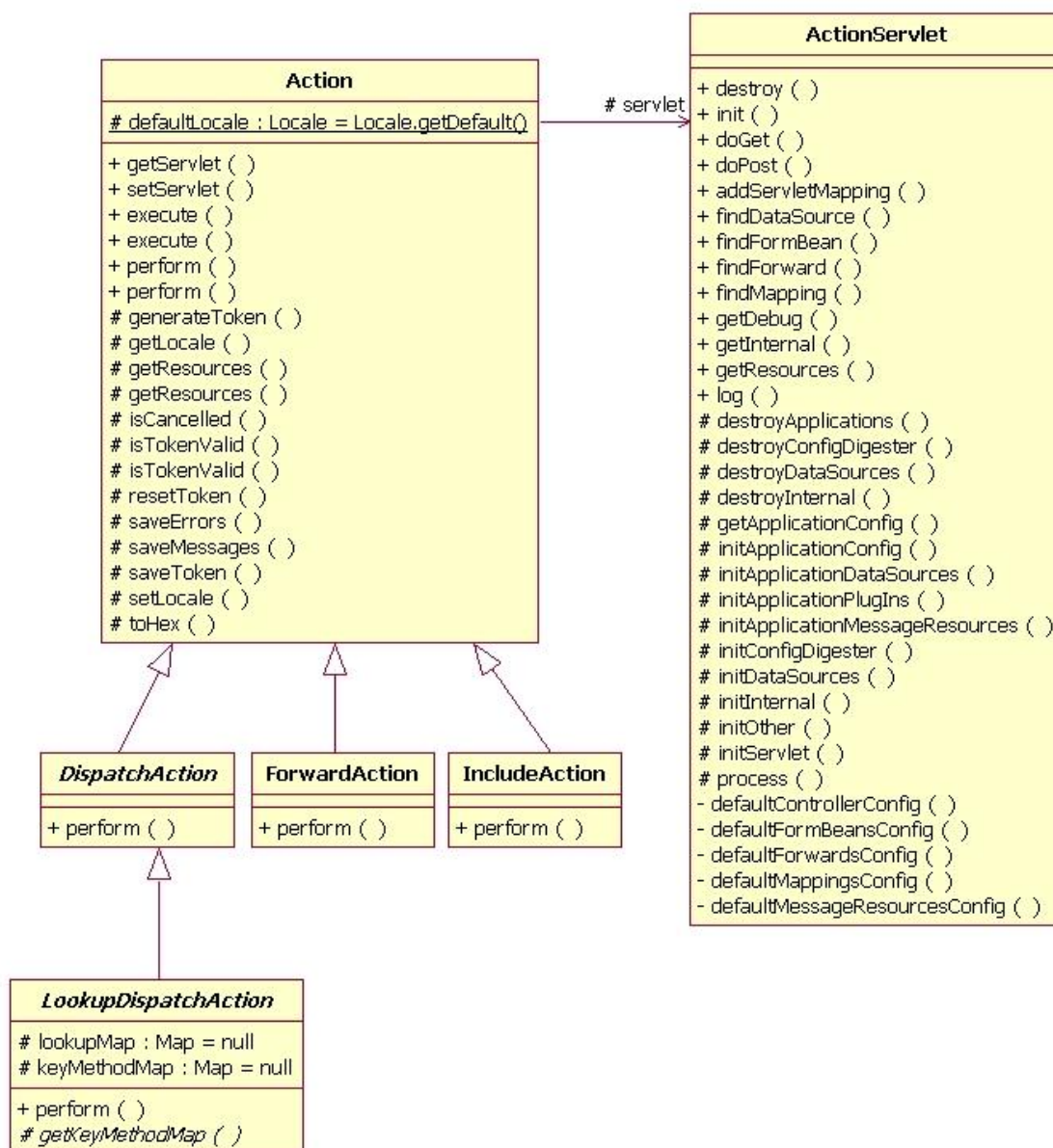


Figura 2-2. Diagrama parcial de clases de Struts

Adicionalmente a controlar la aplicación, la instancia de `ActionServlet` es responsable de la inicialización y limpieza de recursos. Cuando el controlador se inicializa, carga la configuración de la aplicación correspondiente al parámetro de inicialización (`init-param`) “`config`”. Recorre la enumeración de todos los elementos `init-param` buscando aquellos cuyo nombre empiece con `config/`. Para cada uno de estos elementos, Struts carga el

archivo de configuración especificado por el valor de ese init-param, y le asigna un valor de prefijo consistiendo en el texto que precede “config/” en el nombre. Por ejemplo, el prefijo del módulo especificado por el init-param llamado config/sigaDemo sería “sigaDemo”. Para acceder al módulo sigaDemo, se tendría que usar una URL similar a:

http://localhost:8080/demoStruts/sigaDemo/unaAccion.do

## Aplicación de Ejemplo

A continuación se hará un ejemplo simple que haga uso del controlador de Struts

1. Registre un nuevo proyecto web en Eclipse llamado **DemoStruts**. Agregue la biblioteca de struts.jar y agregue a la carpeta WebRoot/WEB-INF los TLDs struts-bean.tld y struts-htnml.tld
2. Modifique el archivo web.xml para que coincida con el mostrado a continuación. Note que se ha agregado el servlet ActionServlet, así como el servlet-mapping

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.4"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <init-param>
      <param-name>debug</param-name>
      <param-value>3</param-value>
```

```
</init-param>
<init-param>
  <param-name>detail</param-name>
  <param-value>3</param-value>
</init-param>
<load-on-startup>0</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
</web-app>
```

3. **Agregue el archivo struts-config.xml a la carpeta WEB-INF con la siguiente información:**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration 1.2//EN" "http://jakarta.apache.org/struts/dtds/struts-
config_1_2.dtd">

<struts-config>
  <data-sources />
  <form-beans />
  <global-exceptions />
  <global-forwards />
  <action-mappings >
    <action path="/Saludo"
      forward="/hola.jsp"/>
  </action-mappings>
</struts-config>
```

4. Agregue a WebRoot el archivo hola.jsp, con la siguiente información

```
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean"%>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html"%>
<html:html locale="true">
<head>
  <title>Título
  </title>
  <html:base />
</head>
<body bgcolor="white">
  <h3>
    Encabezado
  </h3>
  <p>
    Mensaje
  </p>
</body>
</html:html>
```

5. Ejecute la página desde un navegador Web usando la URL **<http://localhost:8080/DemoStruts/Saludo.do>** y verifique el resultado

### 3. Action

Cuando un usuario hace una petición de algo, la solicitud es manejada por el componente ActionServlet de Struts. Cuando el ActionServlet recibe la petición, intercepta la URL y, en base a los archivos de configuración de Struts, le cede el manejo de la petición a la clase Action correspondiente. La clase Action es una parte del controlador y es responsable de la comunicación con la capa de modelo.

Las acciones de Struts generalmente extienden de la clase org.apache.struts.action.Action. Struts proporciona también subclases Action que se pueden extender para acciones con requerimientos muy específicos. La clase Action actúa como un “wrapper” sobre la lógica de negocios y proporciona una interfaz a la capa de modelo de la aplicación, como ya se mencionó. Se encarga también de transferir los datos de la capa de la vista a la capa de proceso de negocios específica para finalmente regresar los datos procesados de la capa de negocios a la capa de la vista.

Una acción trabaja como un adaptador entre el contenido de una solicitud http de entrada y la lógica de negocios correspondiente. De esta forma, el controlador de struts selecciona una acción correcta y crea una instancia de ella, si es necesario, para finalmente mandar llamar al método execute.

Para emplear la acción, es necesario crear una subclase de Action y sobrescribir el método execute().

#### **Aplicación de Ejemplo**

A continuación se hará un ejemplo simple que haga uso de una acción

1. Modifique el archivo `struts-config.xml` a la carpeta `WEB-INF` tal como se muestra a continuación:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration 1.2//EN" "http://jakarta.apache.org/struts/dtds/struts-
config_1_2.dtd">

<struts-config>
  <data-sources />
  <form-beans />
  <global-exceptions />
  <global-forwards />
  <action-mappings >
    <action path="/Saludo"
      forward="/hola.jsp"/>
    <action path="/PruebaAccion"
      type="mx.com.siga.struts.PruebaAction">
      <forward name="muestraSaludo" path="/hola.jsp"/>
    </action>
  </action-mappings>
</struts-config>
```

2. Agregue un nuevo paquete llamado `mx.com.siga.struts` en la carpeta de `src` y creé una nueva clase llamada `PruebaAction.java`. La única función de esta clase será direccionar al archivo `hola.jsp`. Esta clase devuelve un `ActionForward` llamado "muestraSaludo", el cual se definió en el archivo `struts-config.xml`.

```
package mx.com.siga.struts;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

public class PruebaAction extends Action {

    public ActionForward execute(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response) throws Exception{
        return mapping.findForward("muestraSaludo");
    }
}
```

3. Ejecute la página desde un navegador Web usando la URL **[http://localhost:8080/DemoStruts/ PruebaAccion.do](http://localhost:8080/DemoStruts/PruebaAccion.do)** y verifique el resultado

### 3.1. Definiendo la clase Action

La siguiente es la firma de la clase Action:

```
public ActionForward execute(ActionMapping mapping,
                            ActionForm form,
                            javax.servlet.http.HttpServletRequest request,
                            javax.servlet.http.HttpServletResponse response)
    throws java.lang.Exception
```

La clase `Action` procesa la solicitud HTTP especificada y crea la respuesta HTTP correspondiente (lo la reenvía a otro componente Web que la creará). Este método regresa una instancia `ActionForward` describiendo donde y como el control debe ser reenviado, o `null` si la respuesta ya ha sido completada.

### 3.2. Los parámetros del método `execute()`

Los parámetros empleados en el método son los siguientes:

- **mapping.** El `ActionMapping` empleado para seleccionar esta instancia
- **form.** El bean opcional `ActionForm` para esta petición
- **request.** La petición http que se está procesando
- **response.** La respuesta que se está creando

En caso de existir alguna excepción, la clase `Action` disparará una excepción de tipo `java.lang.Exception`

Adicional a la firma mostrada del método `execute()`, existe otra firma alterna:

```
public ActionForward execute(ActionMapping mapping,
                             ActionForm form,
                             ServletRequest request,
                             ServletResponse response)
    throws java.lang.Exception
```

Dado que Struts será empleado la mayoría de las veces en aplicaciones Web, la forma que se empleará será la versión de “`HttpServletRequest`”. El método `execute()` que no emplea http se proporciona para aplicaciones que no dependen del protocolo http.

En el patrón de diseño MVC/Modelo 2, una clase `Action` típica implementará generalmente una lógica similar a la siguiente en su método `execute()`:

1. Valida el estado actual de la sesión del usuario. Si la clase Action encuentra que el usuario no se firmó, la solicitud se puede reenviar a la página de presentación que solicita los datos del usuario y la contraseña para que se registre el usuario.
2. Si la validación no se ha completado, se valida las propiedades del form bean conforme se requiera. Si se detecta un problema, se guardan las llaves de mensajes de error convenientes como un atributo de la solicitud y se reenvía el control de nuevo a la forma de entrada hasta que los errores sean corregidos.
3. Se ejecuta el procesamiento requerido para tratar con la solicitud (tal como guardar un registro en la base de datos). Esto puede hacerse con lógica de código embebido en la clase Action misma, pero debería ser ejecutado generalmente llamando un método adecuado de un bean de lógica de negocios.
4. Actualiza los objetos del lado del servidor que serán empleados para crear la siguiente página de la interfaz de usuario.
5. Regresa un objeto ActionForward conveniente que identifique la página de presentación que será empleada para generar esta respuesta. Basada en los beans recientemente actualizados.

En Apache Struts 1.0, las acciones ejecutaban un método llamado `perform()` en vez del método `execute()`. Estos métodos emplean los mismos parámetros y difieren solo en las excepciones que disparan. El método `perform()` antiguo dispara las excepciones `ServletException` y `IOException`. El nuevo método `execute()` solo dispara la excepción `Exception`.

El método `perform()` pudiera ser empleado en Apache Struts 1.1 a pesar de ser obsoleto y a partir de la versión 1.2 ha sido removido.

## 4. ActionMappings

Como se vio antes, el servlet del controlador necesita conocer varias cosas sobre como debe ser mapeado cada URI de solicitud a una clase Action correcta. Este conocimiento ha sido encapsulado en una clase Java llamada ActionMapping, cuyas propiedades más importantes son las siguientes:

- **type**. Nombre de la clase Java totalmente calificada que implementa la clase Action empleada para este mapeo.
- **name**. El nombre del form bean definido en el archivo de configuración que empleará esta acción.
- **path**. La ruta URI de la solicitud que es comparada para seleccionar este mapeo.
- **Unknown**. Se asigna un valor de true si esta acción debe ser configurada por defecto para esta aplicación, para manejar todas las solicitudes no manejadas por otra acción. Solo una acción puede ser definida para ser empleada por defecto dentro de una aplicación.
- **validate**. Se asigna un valor de true si el método validate de la acción asociada con este mapeo debe ser llamado.
- **forward**. La ruta URI de la solicitud a la cual se le transferirá el control cuando este mapeo sea invocado. Esta es una alternativa a declarar una propiedad type

En vez de tener que escribir una serie de clases de Java que se encarguen de instanciar objetos de tipo ActionMapping, Struts emplea el componente Commons Digester de Yakarta para parsear una descripción basada en un archivo de XML a los mapeos deseados y crear los objetos correctos inicializados a los valores por defecto apropiados.

El desarrollador debe crear un archivo con formato XML llamado `struts-config.xml` y colocarlo en el directorio `WEB-INF` de la aplicación. El formato de este documento se describe en el DTD en la dirección [http://jakarta.apache.org/struts/dtds/struts-config\\_1\\_1.dtd](http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd). El controlador usa una copia interna de este documento para conocer como mapear cada solicitud URI a una clase `Action` apropiada, por lo que no se requiere una conexión a Internet para realizar la validación.

En la siguiente sección se hablará más respecto al archivo de configuración.

## 5. Configuración de Struts

### 5.1. El archivo struts-config.xml

Como se ha observado en los ejemplos mostrados anteriormente, el primer elemento del archivo de configuración de Struts es `<struts-config>`. Adentro de este elemento, existen tres elementos importantes que son empleados para describir las acciones:

- `<form-beans>`
- `<global-forwards>`
- `<action-mappings>`

#### 5.1.1. La sección `<form-bean>`

Esta sección contiene las definiciones de form beans. Los form beans son descriptores que son empleados para crear instancias de tipo `ActionForm` en tiempo de ejecución. Este tema se verá más adelante.

Se empleará un elemento `<form-bean>` para cada form bean empleado. Tiene varios atributos importantes:

- **name**. Un identificador único para este bean, el cual será empleado como referencia para el correspondiente mapeo de la acción.
- **type**. Es el classname de Java totalmente calificado de la subclase `ActionForm` que empleará este form bean

### 5.1.2. La sección <action-mapping>

Esta sección contiene las definiciones de las acciones. Se emplea un elemento <action> para cada uno de los mapeos que se desea definir.

La mayoría de las acciones definirán al menos los siguientes atributos, vistos anteriormente:

- **path.** La ruta a la acción, relativa al contexto de la aplicación
- **type.** El classname de Java totalmente calificado de la clase Action
- **name.** El nombre del elemento <form-bean> a emplear con esta acción

### 5.1.3. La sección <global-forwards>

Esta sección contiene las definiciones de reenvío globales. Los reenvíos (Forwarders) son instancias de la clase ActionForward devueltas por un método execute de un ActionForm. Estos mapean nombres lógicos con recursos específicos (generalmente JSPs) , permitiendo modificar el recurso sin cambiar las referencias a el a lo largo de la aplicación. Se emplea un elemento <forward> para cada definición. Los atributos importantes de este elemento son:

- **name.** El nombre lógico de este reenvío. Es empleado en el método execute del ActionForm para reenviar al siguiente recurso correcto.
- **path.** La ruta relativa de contexto al recurso.
- **redirect.** True o False (por defecto). Indica si el ActionServer debe hacer un redireccionamiento (redirect) en vez de un reenvío (forward).

Una razón que es importante tener en cuenta al crear el archivo `struts-config.xml` es que los parámetros que contiene son dependientes del orden, por lo que es crucial colocar los subelementos en el orden correcto.

Adicional a la configuración que se mencionó del archivo `struts-config.xml`, las aplicaciones de Struts requieren de algunas consideraciones importantes para poder funcionar.

Dado que las aplicaciones de Struts son propiamente aplicaciones de Web, estas tienen que seguir las mismas reglas a las que cualquier aplicación Web debe apegarse. Una de las reglas es que cada aplicación Web debe tener su propio archivo `deployment descriptor web.xml`

En una aplicación de Struts, el archivo `web.xml` contiene la definición del servlet y el mapeo a `ActionServlet`. También contiene definiciones de archivos de recursos y al archivo `struts-config.xml`. De igual forma, puede contener las definiciones a las bibliotecas de tags de Struts.

Tal como cualquier otra aplicación Web, el archivo `web.xml` reside en el directorio `WEB-INF`, ya sea en forma empaquetada o desempaquetada.

#### **5.1.4. Internacionalización**

Hace algunos años, las aplicaciones contaban con un grupo local de usuarios, los cuales requerían uno (o en ocasiones dos) idiomas y una forma para representar las

cantidades numéricas como fechas, números y valores monetarios. Sin embargo, la explosión de aplicaciones desarrolladas en tecnologías Web, así como la disponibilidad de tales aplicaciones en Internet y otras redes ampliamente accesibles, han desaparecido los límites entre naciones. Esto ha repercutido en la necesidad de soportar características de internacionalización en las aplicaciones (i18n) y localización.

Struts cuenta con elementos enfocados a atender este tipo de necesidades. Tales elementos son:

- **Locale.** La clase de java fundamental que soporta la internacionalización es llamada Locale. Cada Locale representa una opción particular de país e idioma, así como un conjunto de diferentes formatos para cantidades tales como números y fechas.
- **ResourceBundle.** La clase `java.util.ResourceBundle` proporciona las herramientas fundamentales para soportar los mensajes en diferentes idiomas.

## Aplicación de Ejemplo

A continuación se hará un ejemplo simple que haga uso de la internacionalización

1. Agregue un nuevo archivo de propiedades en el paquete `mx.com.siga.struts` llamado `ApplicationResources.properties`. Agregue los siguientes mensajes:

```
# Recursos para el parámetro 'mx.com.siga.struts.ApplicationResources'  
# Proyecto DemoStruts  
# Mensajes en Español  
hola.titulo=Titulo  
hola.encabezado=encabezado  
hola.mensaje=mensaje
```

2. Modifique el archivo struts-config.xml a la carpeta WEB-INF tal como se muestra a continuación:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration 1.2//EN" "http://jakarta.apache.org/struts/dtds/struts-
config_1_2.dtd">

<struts-config>
  <data-sources />
  <form-beans />
  <global-exceptions />
  <global-forwards />
  <action-mappings >
    <action path="/Saludo"
      forward="/hola.jsp"/>
    <action path="/PruebaAccion"
      type="mx.com.siga.struts.PruebaAction">
      <forward name="muestraSaludo" path="/hola.jsp"/>
    </action>
  </action-mappings>
  <bmessage-resources parameter="mx.com.siga.struts.ApplicationResources" />
</struts-config>
```

3. Modifique el archivo hola.jsp como se muestra a continuación:

```
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean"%>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html"%>
<html:html locale="true">
```

```
<head>
  <title><bean:message key="hola.titulo" />
</title>
  <html:base />
</head>
<body bgcolor="white">
  <h3>
    <bean:message key="hola.encabezado" />
  </h3>
  <p>
    <bean:message key="hola.mensaje" />
  </p>
</body>
</html:html>
```

4. Ejecute la página desde un navegador Web usando la URL **[http://localhost:8080/DemoStruts/ PruebaAccion.do](http://localhost:8080/DemoStruts/PruebaAccion.do)** y verifique el resultado
5. Agregue un nuevo archivo de propiedades en el paquete `mx.com.siga.struts` llamado `ApplicationResources_en.properties`. Agregue los siguientes mensajes:

```
# Recursos para el parámetro 'mx.com.siga.struts.ApplicationResources'
# Proyecto DemoStruts
# Mensajes en Español
hola.titulo=Title
hola.encabezado=header
hola.mensaje=message
```

6. Modifique el archivo `struts-config.xml` a la carpeta `WEB-INF` tal como se muestra a continuación:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration 1.2//EN" "http://jakarta.apache.org/struts/dtds/struts-
config_1_2.dtd">

<struts-config>
  <data-sources />
  <form-beans />
  <global-exceptions />
  <global-forwards />
  <action-mappings >
    <action path="/Saludo"
      forward="/hola.jsp"/>
    <action path="/PruebaAccion"
      type="mx.com.siga.struts.PruebaAction">
      <forward name="muestraSaludo" path="/hola.jsp"/>
    </action>
  </action-mappings>
  <message-resources parameter="mx.com.siga.struts.ApplicationResources_en"
  />
</struts-config>
```

7. Ejecute de nuevo la página desde un navegador Web usando la URL **[http://localhost:8080/DemoStruts/ PruebaAccion.do](http://localhost:8080/DemoStruts/PruebaAccion.do)** y verifique el resultado

## Capítulo III

### FORMAS

Capítulo III FORMAS.....	32
1. ActionForms .....	33
1.1. Tags HTML de Struts .....	42
2. DynaActionForms .....	44
3. Validación .....	51

## 1. ActionForms

Un `ActionForm` es un bean de Java que extiende de la clase `org.apache.struts.action.actionForm`, el cual representa una forma de HTML con la que el usuario interactúa sobre una o más páginas. Un `ActionForm` mantiene el estado de la sesión para la aplicación Web y el objeto `ActionForm` es llenado automáticamente en el lado del servidor con datos proporcionados por una forma en el lado del cliente.

Es necesario que el `ActionForm` tenga propiedades para guardar estos datos de la forma con getters y setters. Los `ActionForms` pueden ser guardados tanto en la sesión (por defecto) como en el request. Si se maneja en la sesión es importante que el `ActionForm` implemente el método `reset` de la forma para inicializarla antes de cada uso.

### Aplicación de Ejemplo

A continuación se hará un ejemplo simple que haga uso de un `ActionForm`

1. Modifique el archivo `ApplicationResources.properties`. Agregue los siguientes mensajes:

```
# Recursos para el parámetro 'mx.com.siga.struts.ApplicationResources'  
# Proyecto DemoStruts  
# Mensajes en Español  
  
#### Configuración de Errores  
errors.header=<font color="#0000ff" face="Tahoma"><strong>Por favor, ingrese  
  los datos correctamente</font><br><ul>  
errors.footer=</ul>  
errors.prefix=<li ><font color="#ff0000">  
errors.suffix=</font></li>
```

```
# Mensajes varios
hola.titulo.captura=Introduzca los siguientes datos, por favor:
hola.titulo.exito=Datos correctos
hola.encabezado=encabezado
hola.mensaje=mensaje

# Mensajes de error
error.nombre.requerido=Se requiere el nombre
error.direccion.requerido=Se requiere la dirección
error.correo.requerido=Se requiere el correo
```

2. Modifique el archivo struts-config.xml a la carpeta WEB-INF tal como se muestra a continuación:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration 1.2//EN" "http://struts.apache.org/dtds/struts-
config_1_2.dtd">

<struts-config>
  <data-sources />
  <form-beans>
    <form-bean
      name="DireccionForm"
      type="mx.com.siga.struts.forms.RegistroForm"/>
  </form-beans>
  <global-exceptions />
  <global-forwards />
  <action-mappings >
    <action path="/Saludo"
      forward="/hola.jsp"/>
    <action path="/PruebaAccion"
```

```
        type="mx.com.siga.struts.PruebaAction">
        <forward name="muestraSaludo" path="/hola.jsp"/>
</action>
<action path="/Direccion"
        type="mx.com.siga.struts.RegistroAction"
        name="DireccionForm"
        scope="request"
        validate="true"
        input="/solicitarDireccion.jsp">
        <forward name="correcto" path="/exito.jsp"/>
</action>
</action-mappings>
<message-resources parameter="mx.com.siga.struts.ApplicationResources" />

</struts-config>
```

3. Creé un nuevo archivo solicitarDireccion.jsp como se muestra a continuación. Observe que se emplean nuevos tags que pertenecen a la biblioteca de tags de Struts

```
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean"%>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>

<html:html locale="true">

<head>

<title><bean:message key="hola.titulo.captura"/></title>

<html:base/>

</head>
```

```
<body bgcolor="white">

<html:form action="/Direccion">

<html:errors/>
</strong></font><table>

    <tr>

        <td align="center" colspan="2">
            <strong><font size="4" face="Arial"><bean:message
key="hola.titulo.captura"/></font></strong>
        </td>
    </tr>
    <tr>
        <td align="right"><font face="Arial" size="2">
            Nombre
        </font></td>
        <td align="left">
            <font face="Arial" size="2"><html:text property="nombre"
size="30" maxlength="30"></html:text></font><font size="2">
            <br></font></td>
    </tr>
    <tr>
        <td align="right"><font face="Arial" size="2">
            Direcci&oacute;n
        </font></td>
        <td align="left">
            <font face="Arial" size="2"><html:text property="direccion"
size="30" maxlength="30"></html:text></font><font size="2">
            <br></font></td>
    </tr>

    <tr>
        <td align="right"><font face="Arial" size="2">
            Correo
        </font></td>
```

```
<td align="left">
    <font face="Arial" size="2"><html:text property="correo"
size="30" maxlength="30"></html:text></font><font size="2">
    <br></font></td>
</tr>
<tr>
    <td align="right">
        <html:submit>Guardar</html:submit>
    </td>
    <td align="left">
        <html:cancel>Cancelar</html:cancel>
    </td>
</tr>
</table>
</html:form>
</body>
</html:html>
```

#### 4. Créé un nuevo archivo exito.jsp como se muestra a continuación:

```
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean"%>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html"%>

<html:html locale="true">

<head>

<title><bean:message key="hola.titulo.exito"/></title>

<html:base/>

</head>

<body bgcolor="white">
```

```
<html:form action="/Direccion">

<html:errors/>

<table>

    <tr>

        <td align="center" colspan="2">
            <strong><font size="4" face="Arial"><bean:message
key="hola.titulo.exito"/></font></strong>
        </td>

    <tr>

        <td align="right"><font face="Arial" size="2">
            Nombre
        </font></td>
        <td align="left">
            <font face="Arial" size="2"><bean:write property="nombre"
name="DireccionForm"/>
        <br></font></td>
    </tr>

    <tr>

        <td align="right"><font face="Arial" size="2">
            Direcci&oacute;n
        </font></td>
        <td align="left">
            <font face="Arial" size="2"><bean:write property="nombre"
name="DireccionForm"/></font><font size="2">
        <br></font></td>
    </tr>

    <tr>

        <td align="right"><font face="Arial" size="2">
            Correo
        </font></td>
```

```
<td align="left">
    <font face="Arial" size="2"><bean:write property="correo"
name="DireccionForm"/> </font><font size="2">
    <br></font></td>
</tr>
</table>
</html:form>
</body>
</html:html>
```

5. Genere una nueva clase RegistroForm.java, dentro de un paquete nuevo mx.com.siga.struts.forms. Esta clase deberá extender la clase ActionForm. Observe como contiene varios atributos, así como los métodos reset y validate.

```
package mx.com.siga.struts.forms;

import javax.servlet.http.HttpServletRequest;

import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionMessage;
import org.apache.struts.action.ActionMessages;

public class RegistroForm extends ActionForm {

    private String nombre=null;
    private String direccion=null;
    private String correo=null;

    /**
     * @return el correo
     */
```

```
public String getCorreo() {
    return correo;
}
/**
 * @param correo el correo a asignar
 */
public void setCorreo(String correo) {
    this.correo = correo;
}
/**
 * @return la direccion
 */
public String getDireccion() {
    return direccion;
}
/**
 * @param direccion la direccion a asignar
 */
public void setDireccion(String direccion) {
    this.direccion = direccion;
}
/**
 * @return el nombre
 */
public String getNombre() {
    return nombre;
}
/**
 * @param nombre el nombre a asignar
 */
public void setNombre(String nombre) {
    this.nombre = nombre;
}

/**
```

```
* Resetea todas las propiedades a sus valores originales
*
* @param mapping El mapeo empleado para seleccionar esta instancia
* @param request La petición que se está procesando
*/
public void reset(ActionMapping mapping, HttpServletRequest request) {
    this.nombre=null;
    this.direccion=null;
    this.correo=null;
}

/**
 * Realiza las validaciones necesarias
 *
 * @param mapping El mapeo empleado para seleccionar esta instancia
 * @param request La petición que se está procesando
 * @return errors
 */
public ActionErrors validate(
    ActionMapping mapping, HttpServletRequest request ) {
    ActionMessages errors = new ActionMessages();

    if( getNombre() == null || getNombre().length() < 1 ) {
        errors.add("nombre",new ActionMessage("error.nombre.requerido"));
    }
    if( getDireccion() == null || getDireccion().length() < 1 ) {
        errors.add("dirección",new
            ActionMessage("error.direccion.requerido"));
    }
    if( getCorreo() == null || getCorreo().length() < 1 ) {
        errors.add("correo",new ActionMessage("error.correo.requerido"));
    }

    return (ActionErrors)errors;
}
```

```
}

```

6. Ejecute la página desde un navegador Web usando la URL **http://localhost:8080/DemoStruts/ Direccion.do** y verifique el resultado

### 1.1. Tags HTML de Struts

Struts proporciona una biblioteca de tags de HTML para una creación sencilla de interfaces de usuario. Para poder emplearla, basta con agregar la siguiente línea en el archivo JSP:

```
<%@ taglib uri="/tags/struts-html" prefix="html" %>

```

A continuación se muestran de forma general los tags más empleados de Struts:

Tags de HTML de Struts	
<code>&lt;html:message key="lallave"/&gt;</code>	Busca el mensaje correspondiente a la llave dada, dentro del archive de recursos y lo despliega.
<code>&lt;html:password property="prop" size="10"/&gt;</code>	Este tag sirve para manejar campos de tipo contraseña. La cadena es almacenada en la propiedad llamada proa en el form bean.
<code>&lt;html:text property="texto1" size="5"/&gt;</code>	Este tag crea un campo de texto. La cadena es obtenida para mostrarla y posteriormente almacenada en la propiedad llamada texto1 en el form bean.

<code>&lt;html:submit&gt;Enviar&lt;/html:submit&gt;</code>	Este tag crea un botón de envío con el contenido proporcionado como el texto del botón.
<code>&lt;html:reset&gt;Reset&lt;/html:reset&gt;</code>	Este tag crea un botón de reset con el contenido proporcionado como el texto del botón.
<code>&lt;html:errors/&gt;</code>	Este tag imprime todos los errores existentes en la página.
<code>&lt;html:file property="fileSelectionBox"/&gt;</code>	Este tag crea el elemento para subir archivos a la forma. La propiedad debe ser del tipo <code>org.apache.struts.upload.FormFile</code> .
<code>&lt;html:checkbox property="myCheckBox"/&gt;</code>	Este tag crea una caja de selección en la forma.
<code>&lt;html:hidden property="hiddenfield"/&gt;</code>	Este tag crea un elemento oculto de html en la forma.
<code>&lt;html:radio value="abc" property="myCheckBox"/&gt;</code>	Este tag crea un botón de radio en la forma.
<code>&lt;html:select multiple="true" property="selectBox"&gt;</code>	Este tag crea una caja de lista en la forma. La propiedad <code>selectBox</code> debe ser un arreglo de tipos de datos soportados y el usuario puede seleccionar varias entradas. Use <code>&lt;html:options&gt;</code> para especificar las entradas.
<code>&lt;html:textarea property="myTextArea" value="Hello Struts" /&gt;</code>	Este tag crea un area de texto en la forma.
<code>&lt;html:form action="/Address" method="post"&gt;</code>	Este tag es empleado pra crear la forma de THML para mandar los datos al servidor.
<code>&lt;html:base/&gt;</code>	Este tag genera el tag base <code>&lt;BASE ...&gt;</code> indicándole a l navegador que la página actual está ubicada en alguna otra URL diferente a la que el navegador encontró. Cualquier referencia relative sera determinada por la URL proporcionada por <code>&lt;BASE HREF="..."&gt;</code> en vez de la URL actual. <code>&lt;BASE ...&gt;</code> se ubica en la sección <code>&lt;HEAD&gt;</code> .
<code>&lt;html:html&gt;</code>	Este tag despliega un elemento HTML <code>&lt;html&gt;</code> .

## 2. DynaActionForms

Ya se ha mostrado el uso de la clase `ActionForm` así como las bondades que ofrece. Sin embargo, mantener una clase `ActionForm` exclusiva para cada forma en la aplicación de Struts requiere mucho tiempo, en especial cuando cada una de ellas debe validarse.

Esta situación puede remediarse con el uso de clases `DynaActionForm`. `DynaActionForm` es una subclase especializada de `ActionForm` que permite la creación de form beans con conjuntos dinámicos de propiedades, sin que se requiera que el desarrollador cree una clase Java para cada forma. De esta forma, en vez de crear una nueva subclase de `ActionForm` y nuevos métodos getters y setters para cada una de las propiedades del bean, se listan sus propiedades, tipo y valores por defecto en el archivo de configuración de Struts.

### Aplicación de Ejemplo

A continuación se hará un ejemplo simple que haga uso de un `DynaActionForm`

1. Modifique el archivo `struts-config.xml` a la carpeta `WEB-INF` tal como se muestra a continuación. El tag `<form-bean>` que se agregará define las características del nuevo form, indicando las tres propiedades requeridas. De igual forma, se agregará un nuevo `action-mapping`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration 1.2//EN" "http://struts.apache.org/dtds/struts-
```

```
config_1_2.dtd">

<struts-config>
  <data-sources />
  <form-beans>
    <form-bean name="DireccionForm"
      type="mx.com.siga.struts.forms.RegistroForm" />
    <form-bean name="DynaDireccionForm"
      type="org.apache.struts.action.DynaActionForm">
      <form-property name="nombre" type="java.lang.String" />
      <form-property name="direccion" type="java.lang.String" />
      <form-property name="correo" type="java.lang.String" />
    </form-bean>
  </form-beans>
  <global-exceptions />
  <global-forwards />
  <action-mappings>
    <action path="/Saludo" forward="/hola.jsp" />
    <action path="/PruebaAccion"
      type="mx.com.siga.struts.PruebaAction">
      <forward name="muestraSaludo" path="/hola.jsp" />
    </action>
    <action path="/Direccion"
      type="mx.com.siga.struts.RegistroAction" name="DireccionForm"
      scope="request" validate="true" input="/solicitarDireccion.jsp">
      <forward name="correcto" path="/exito.jsp" />
    </action>
    <action path="/DynaDireccion"
      type="mx.com.siga.struts.DynaRegistroAction"
      name="DynaDireccionForm"
      scope="request" validate="true"
      input="/dynaSolicitarDireccion.jsp">
      <forward name="correcto" path="/exito.jsp" />
      <forward name="incorrecto"
        path="/dynaSolicitarDireccion.jsp" />
  </action-mappings>
</struts-config>
```

```
        </action>
    </action-mappings>
    <message-resources
        parameter="mx.com.siga.struts.ApplicationResources" />
</struts-config>
```

2. Agregue una nueva acción llamada DynaRegistroAction.java, la cual empleará la clase DynaActionForm

```
package mx.com.siga.struts;

import javax.servlet.http.*;
import org.apache.struts.action.*;

public class DynaRegistroAction extends Action {

    public ActionForward execute(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response) throws Exception{

        DynaActionForm addressForm = (DynaActionForm) form;

        //Se crea el objeto ActionMessages
        ActionMessages errors = new ActionMessages();
        //Se verifican entradas y se juntan los errores
        if(((String)addressForm.get("nombre")).equals("")) {
            errors.add("nombre", new
                ActionMessage("error.nombre.requerido"));
        }
    }
}
```

```
    }

    if(((String)addressForm.get("direccion")).equals("")) {
        errors.add("direccion",new
ActionMessage("error.direccion.requerido"));
    }

    if(((String)addressForm.get("correo")).equals("")) {
        errors.add("correo",new
ActionMessage("error.correo.requerido"));
    }

    //Se guardan los errores
saveErrors(request,errors);
//Se reenvía la página
if(errors.isEmpty()){
return mapping.findForward("correcto");
}else{
return mapping.findForward("incorrecto");
}
}
}
```

3. Créé un nuevo archivo dynaSolicitarDireccion.jsp como se muestra a continuación. Observe que se emplea ahora el Dyna Form que se creó anteriormente.

```
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean"%>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html"%>

<html:html locale="true">

<head>
```

```
<title><bean:message key="hola.titulo.captura" />
</title>

<html:base />

</head>

<body bgcolor="white">

  <html:form action="/DynaDireccion" method="post">

    <html:errors />
    </strong>
    </font>
    <table>

      <tr>

        <td align="center" colspan="2">
          <strong><font size="4" face="Arial"><bean:message
            key="hola.titulo.captura" />
          </font>
        </strong>
      </td>
      <td align="right">
        <font face="Arial" size="2"> Nombre </font>
      </td>
      <td align="left">
        <font face="Arial" size="2"><html:text
property="nombre"
          size="30" maxlength="30"></html:text>
        </font><font size="2"> <br>
        </font>
      </td>
    </tr>
  </table>
</form>
</body>
```

```
<tr>
  <td align="right">
    <font face="Arial" size="2"> Direcci&oacute;n </font>
  </td>
  <td align="left">
    <font face="Arial" size="2"><html:text
property="direccion"
                                size="30" maxlength="30"></html:text>
    </font><font size="2"> <br>
    </font>
  </td>
</tr>

<tr>
  <td align="right">
    <font face="Arial" size="2"> Correo </font>
  </td>
  <td align="left">
    <font face="Arial" size="2"><html:text
property="correo"
                                size="30" maxlength="30"></html:text>
    </font><font size="2"> <br>
    </font>
  </td>
</tr>
<tr>
  <td align="right">
    <html:submit>Guardar</html:submit>
  </td>
  <td align="left">
    <html:cancel>Cancelar</html:cancel>
  </td>
</tr>
</table>
</html:form>
</body>
```

```
</html:html>
```

4. Ejecute la página desde un navegador Web usando la URL **[http://localhost:8080/DemoStruts/ Direccion.do](http://localhost:8080/DemoStruts/Direccion.do)** y verifique el resultado

### **3. Validación**

Validación