


STRUTS 2

Struts

- **Struts 2** es un framework, dentro de las capas en las que se divide una aplicación en la arquitectura **JEE**, el cual implementa el controlador del patrón de diseño **MVC (Modelo Vista Controlador)**.
- Además proporciona algunos componentes para la capa de vista.
- Por si fuera poco, proporciona una integración perfecta con otros frameworks para implementar la capa del modelo (como **Hibernate** y **Spring**).



Para hacer más fácil presentar datos dinámicos, el framework incluye una biblioteca de etiquetas web. Las etiquetas interactúan con las validaciones y las características de internacionalización del framework, para asegurar que las entradas son válidas, y las salidas están localizadas

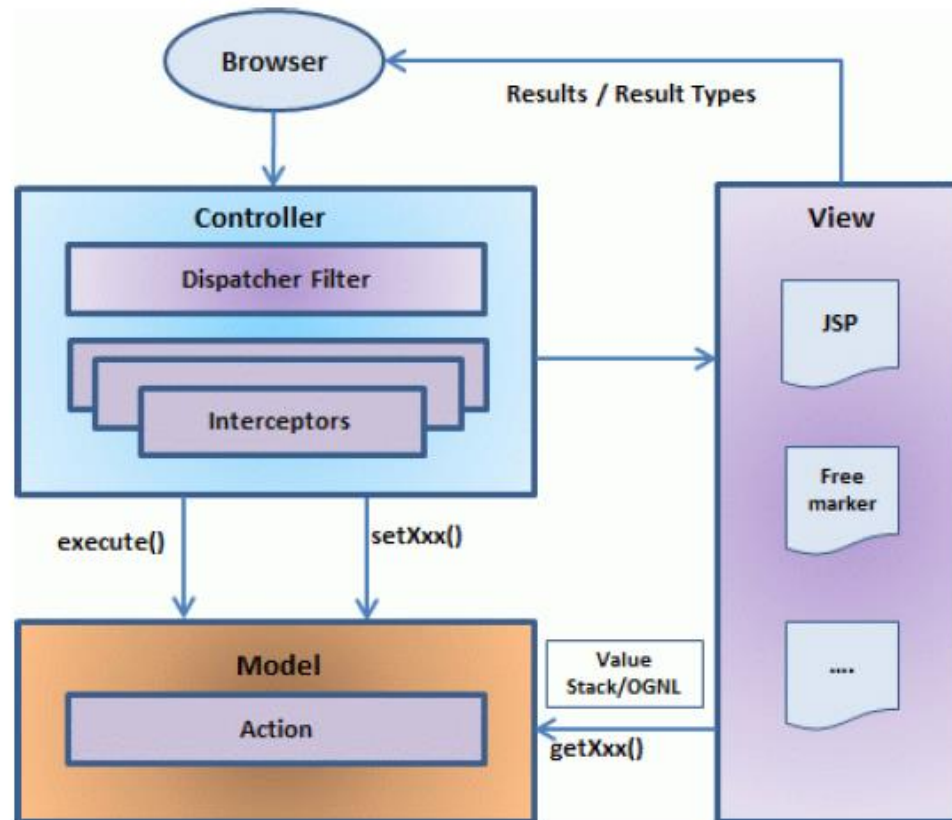


Struts

ESTRUCTURA DE UNA APLICACIÓN WEB STRUTS Y EL PATRÓN MVC

Struts

- Arquitectura Struts 2: implementación del patrón MVC



Struts

- Los pasos que sigue una petición son:
 1. El navegador web hace una petición.
 2. El filtro de Struts (FilterDispatcher) decide que Action debe atender la petición.
 3. Se ejecutan los interceptores de entrada.
 4. Se ejecuta el Action (normalmente el método execute).
 5. El Action indica qué Result debe generar la salida.
 6. Se ejecutan los interceptores de salida.
 7. El resultado es enviado al navegador.



Struts

PUESTA EN MARCHA

Struts



Ejemplo 1: HelloWorld

Struts

- HolaMundoStruts
 - Crear una aplicación Web Dinamica con Eclipse. (generar el web.xml)
 - Descargar librerías de Struts2 : Essential Dependencies Only(<http://struts.apache.org/download.cgi>)
 - Añadir las siguientes librerías al proyecto:
 - Struts2-core.jar, xwork.jar, ognl.jar, javassist, freemarker.jar, commons-lang, commons-logging.jar, commons-fileupload.jar, commons-io.jar

Struts

- HelloWorld
 - Crear el Action:

```
package example;
import com.opensymphony.xwork2.ActionSupport;
public class HolaMundoAction extends ActionSupport {
    private String saludo="¡Hola a todos! ";
    public String execute() throws Exception {
        return SUCCESS;
    }
    public String getSaludo() {
        return saludo;
    }
    public void setSaludo(String saludo) {
        this.saludo = saludo;
    }
}
```

Struts

- HelloWorld

- Crear el fichero struts.xml en la carpeta src y añadir el action en el paquete example:

```
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
<package name="example" namespace="/example" extends="struts-default">
    <action name="HolaMundoAction" class="example.HolaMundoAction">
        <result name="success" >/HolaMundoVista.jsp</result>
    </action>
</package>
</struts>
```

Struts

- HelloWorld

- Crear la vista (HolaMundoVista.jsp) :

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
  <head>
    <title>Hola mundo, V.Struts</title>
  </head>
  <body>
    <h2><s:property value="saludo"/></h2>
  </body>
</html>
```

Struts

- HolaMundoStruts

- Añadimos el filtro de struts a nuestro web.xml:

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

Esa es la librería donde Struts2 guarda su filtro desde la versión 2.5, para versiones anteriores la librería e incluso el nombre del filtro cambia.

Struts

Si aparece algún error de alguna clase que no se encuentra (ClassNotFoundException) verificar que Eclipse empaqueta las librerías de Struts2 al archivo .war que se despliega en el servidor.

Para eso. Boton derecho sobre el proyecto-> properties-> Deployment Assembly.

Aquí hay que añadir Java Build Path Entries, y seleccionar todas las librerías de Struts2.

Struts

- HelloWorld
 - Ejecutar la aplicación:

<http://localhost:8080/HolaMundoStruts/example/HolaMundoAction.action>

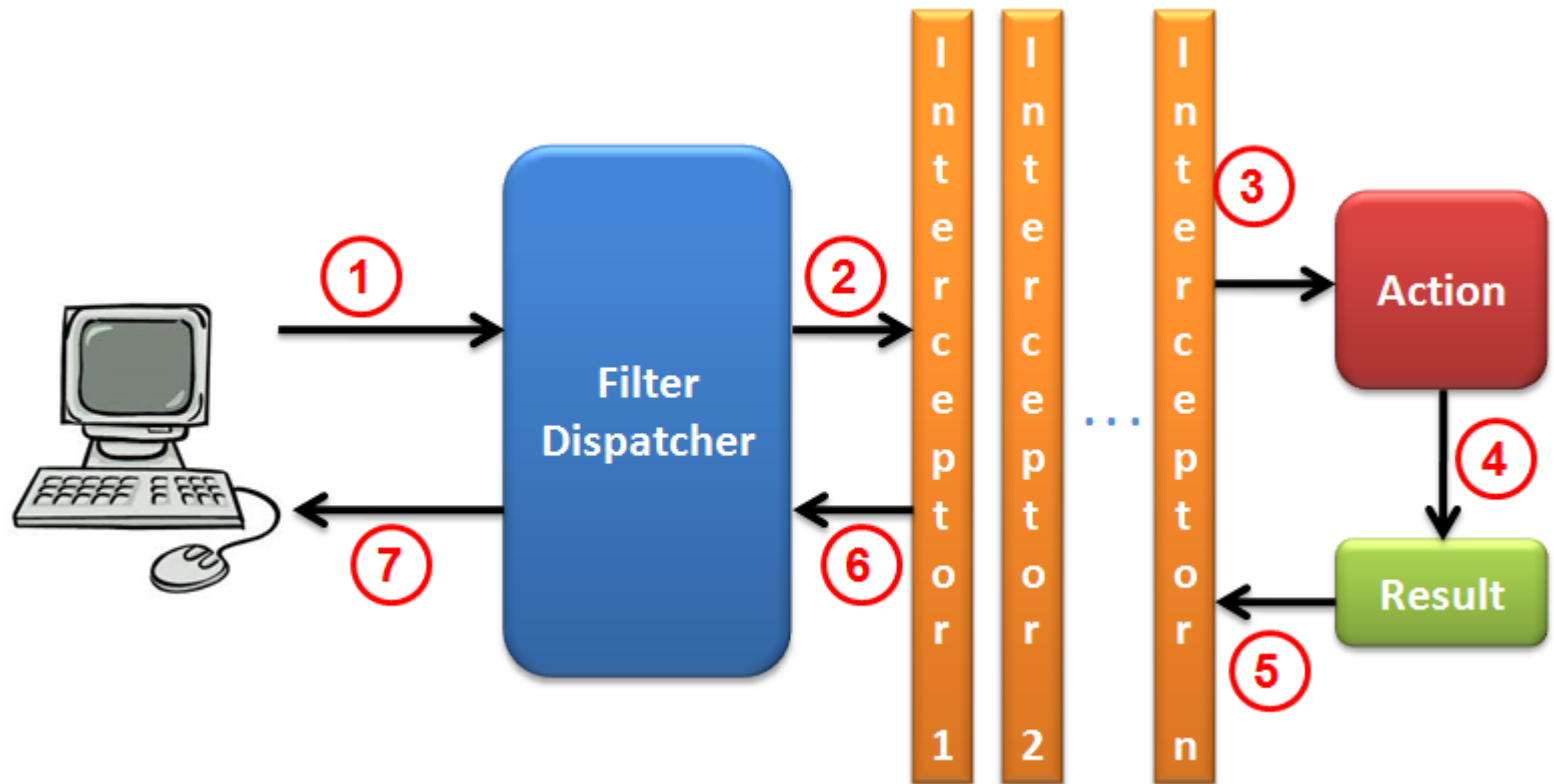
0


<http://localhost:8080/HolaMundoStruts/example/HolaMundoAction>

Struts



**¿Cómo funciona todo esto?
El corazón de Struts2.**



- 
- El corazón de Struts 2 es un filtro, conocido como el "FilterDispatcher". Este es el punto de entrada del framework. A partir de él se lanza la ejecución de todas las peticiones que involucran al framework.

Las principales responsabilidades del "FilterDispatcher" son:

- Ejecutar los Actions, que son los manejadores de las peticiones.
- Comenzar la ejecución de la cadena de interceptores
- Limpiar el "ActionContext", para evitar fugas de memoria.

Struts 2 procesa las peticiones usando tres elementos principales:

- Interceptores
- Acciones
- Resultados

Struts

Los pasos que sigue una petición son:

- 1** - El navegador web hace una petición para un recurso de la aplicación (index.action, reporte.pdf, etc.). El filtro de Struts (FilterDispatcher) revisa la petición y determina el Action apropiado para servirla.
- 2** - Se aplican los interceptores, los cuales realizan algunas funciones como validaciones, flujos de trabajo, manejo de la subida de archivos, etc.
- 3** - Se ejecuta el método adecuado del Action (por default el método "execute"), este método usualmente almacena y/o regresa alguna información referente al proceso.
- 4** - El Action indica cuál result debe ser aplicado. El result generará la salida apropiada dependiendo del resultado del proceso.
- 5** - Se aplican al resultado los mismos interceptores que se aplicaron a la petición, pero en orden inverso.
- 6** - El resultado vuelve a pasar por el FilterDispatcher aunque este ya no hace ningún proceso sobre el resultado.
- 7** - El resultado es enviado al usuario y este lo visualiza.

Struts

ACTION

Struts

•Action:

- Las acciones (Actions) son las clases que implementan la respuesta a una petición.
- Cada URL es mapeada a un Action.
- Un objeto Action actua como JavaBean.
- Un Action debe tener un método execute que no reciba argumentos y que devuelva String o Result.
- Si la respuesta es un String el Result se obtiene a partir de la configuración del Action.
- Puede implementar la interface `com.opensymphony.xwork2.Action`

```
public interface Action {  
    public static final String SUCCESS = "success";  
    public static final String NONE = "none";  
    public static final String ERROR = "error";  
    public static final String INPUT = "input";  
    public static final String LOGIN = "login";  
    public String execute() throws Exception;  
}
```

Struts

La Interfaz de Acción ofrece 5 constantes que se pueden devolver, son:

SUCCESS indica que la ejecución de la acción ha tenido éxito y un resultado de éxito se debe mostrar al usuario.

ERROR indica que la ejecución de la acción falló y un resultado de error se debe mostrar al usuario.

LOGIN indica que el usuario no está logueado y como resultado al usuario se le debe mostrar el inicio de sesión

INPUT indica que la validación ha fallado y un resultado del formulario se debe mostrar al usuario de nuevo.

NONE indica que la ejecución de la acción ha tenido éxito aunque el resultado no se debe mostrar al usuario.

Struts

- **Action:**

- Puede heredar de la clase `com.opensymphony.xwork2.ActionSupport`

```
public class ActionSupport implements Action, Validateable,
ValidationAware,
TextProvider, LocaleProvider, Serializable {
    ...
    public String execute() throws Exception {
        return SUCCESS;
    }
}
```

Struts

- Action: La programación implica varias fases:

- Mapeo de la acción con una clase.

```
<action name="intentaLogin" class="actions.LoginAction">
    <result name="input">/index.jsp</result>
    <result name="error">/index.jsp</result>
    <result>/loginok.jsp</result>
</action>
```

- Mapeo del resultado con una vista.

```
<action name="intentaLogin" class="actions.LoginAction">
    <result name="input">/index.jsp</result>
    <result name="error">/index.jsp</result>
    <result>/loginok.jsp</result>
</action>
```

- Escribir la lógica de control en la clase Action.

```
public String execute() throws Exception {
    return Action.SUCCESS
    o
    return Action.ERROR
}
```


Mapeo de la acción con una clase

- **name:** debe ser definido en todas las acciones, por este nombre se identifica el action
- **class:** Clase que implementa el action
- **method** :Es Opcional. Si no se especifica ningun metho, el metodo **execute()** se considerará como el metodo mapeado.

• Así este código:

```
<action name="product" class="victor.example.HolaMundo">
```

sería igual que este:

```
<action name="product" class="victor.example.HolaMundo" method="execute">
```

- Si quieres invocar un método particular del action, se debe indicar en el atributo method.

Ejemplo: Action Mapping

```
<action name="Logon" class="tutorial.Logon">  
  <result>Menu</result>  
  <result name="input"> /tutorial/Logon.jsp</result>  
</action>
```

Action Mapping: Atributo name

- Los enlaces a acciones suelen ser generados dentro de una aplicación con las etiquetas de Struts.
- La etiqueta puede indicar el nombre de la acción y Struts2 se encargará de componer la extensión por defecto y cualquier parte que necesite.

```
<s:form action="bienvenido">
```

```
<s:textfield label = "Introduzca su nombre" name="nombre" />
```

```
<s:submit>
```

```
</s:form>
```

Action por defecto

- Podemos especificar acciones por defecto que manejen las peticiones que no coincidan con ninguna acción.

```
<package name="Hello" extends="action-default">  
  <default-action-ref name="UnderConstruction">  
  
    <action name="UnderConstruction">  
      <result>/UnderConstruction.jsp</result>  
    </action>  
  </default-action-ref>  
</package>
```

Ejemplo en JSP

```
<html>
<head><title>Add Blog Entry</title></head>
<body>
<form action="save.action" method="post">
Title: <input type="text" name="title" /><br/>
Entry: <textarea rows="3" cols="25"
name="entry"></textarea>
<br/>
<input type="submit" value="Add"/>
</form>
</body>
</html>
```

Ejemplo con Etiquetas de Struts

```
<%@ taglib prefix="s" uri="/WEB-INF/struts-tags.tld" %>
<html>
<head><title>Add Blog Entry</title></head>
<body>
<s:form action="save" method="post" >
<s:textfield label="Title" name="title" />
<s:textarea label="Entry" name="entry" rows="3"
cols="25" />
<s:submit value="Add"/>
</s:form>
</body>
</html>
```

Completando el HolaMundo

- HelloWorld

- Crear la página principal (index.jsp):

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8">
        <title>Aplicación Hola Mundo Struts!!</title>
    </head>
    <body>
        <h1>Hello World. Version Struts</h1>
        <s:form action =example/HolaMundoAction" >
        <s:textfield name="nombre" label='Introduce tu Nombre' />
        <s:submit value="Enviar" /> </s:form>
        o
        <form action="example/HolaMundoAction">
            <label for="name">Introduce tu nombre</label><br/>
            <input type="text" name="nombre"/>
            <input type="submit" value="Saluda"/>
        </form>
    </body>
</html>
```

Completando el HolaMundo

- Modifico el action añadiendo:

```
private String nombre="";
    public String execute() throws Exception {
        saludo+=nombre;
        return SUCCESS;
    }
    public String getNombre() {
return nombre;
}
    public void setNombre(String nombre) {
this.nombre = nombre;
}
```


Interfaz *Action*

- Es el punto de entrada en la ejecución de una acción. Cualquier clase puede implementar la interfaz e implementar el método `execute()`.
- Es opcional, si una acción no implementa `Action` el framework por reflexión será capaz de buscar un método `execute()`.

```
public interface Action{  
    public String execute() throws Exception;  
}
```

Métodos de acción (Action Methods)

- Permiten indicar varios puntos de entrada a la misma Action.
 - ▣ Ejemplo, en una acción que realice un CRUD (create, retrieve, update y delete) cada operación puede ser un punto de entrada distinto a la misma acción

```
<action name="delete" class="example.CrudAction" method="delete">
```

- Si no existe método `execute()` en la acción y ningún otro método coincide, el framework lanza una excepción.

Método comodín en acciones

- ☐ Acciones que comparten un patrón común.
- ☐ Agrupación de los mapeos similares mediante un comodín, en lugar de tener diferentes mapeos.

```
<action name="*Crud" class="example.CrudAction" method="{1}">
```

- ☐ Una referencia a editCrud llamará al método edit de la clase.
- Una referencia a deleteCrud llamará al método delete.
- ☐ {1} indica que coge el valor del 1º asterisco(*)

Paso de mensajes

- Action:

- Paso de mensajes a la presentación:

- Invocando al método `addActionMessage` se agregan a una lista.

- En la vista (jsp) se muestra la lista de mensajes con la etiqueta `<s:actionmessage/>`

- Paso de mensajes de error a la presentación:

- Invocando al método `addActionError` se agregan a una lista.

- En la vista (jsp) se muestra la lista de mensajes con la etiqueta `<s:actionerror/>`

Struts

RESULTS

Struts

- Results:

- Los Results envían la respuesta de la ejecución del Action al usuario.
- Tiene dos componentes: el tipo de result y el result.
- El tipo de result indica como debe ser tratado el resultado (ejecutar un JSP, hacer una redirección, enviar un fichero).Por defecto su valor es dispatcher.
- Un Action puede tener más de un result asociado, lo que permite generar diversas vistas (result “SUCESS” vs result “ERROR”).

Struts

- Results:

- El mapeo entre el return del Action y el Result se indica en el fichero struts.xml

```
<action name="intentaLogin" class="actions.LoginAction">  
    <result name="input">/index.jsp</result>  
    <result name="error">/index.jsp</result>  
    <result>/loginok.jsp</result>  
</action>
```

- Los tokens predefinidos para el name son:

- String SUCCESS = "success";
- String NONE = "none";
- String ERROR = "error";
- String INPUT = "input";
- String LOGIN = "login";

- Se pueden agregar los valores de *name* que se quiera siempre que sean devueltos por los objetos Action.

Etiqueta Result

Algunos ejemplos:

Sin valores sin defecto

```
<result name="sucess" type="dispatcher">  
<param name="location">/listado.jsp</param>  
</result>
```

Algunos valores por defecto

```
<result>  
<param name="location">/listado.jsp</param>  
</result>
```

Valor por defecto en <param>

```
<result>/listado.jsp</result>
```


Etiqueta Result

Ejemplo de <action> con varios <result>

```
<action name="Hello">  
  <result>/jsp/resultado.jsp</result> <!--  
  name="success" -->  
  <result name="error">/hello/Error.jsp</result>  
  <result name="input">/hello/INput.jsp</result>  
</action>
```

Tipos de resultado (Result Types)

- *Indican el tipo de acción que se realiza al finalizar el Action. Pueden ser*
 - *Dispatcher*
 - *Redirect Action*
 - *Chain*
 - *Redirect*
 - *Stream*
 - *FreeMarker*
 - *Velocity*
 - *PlainText*
 - *Tiles*
 - *HttpHeader*

- Los más importantes son los primeros:
- **dispatcher**" - Este es el result más usado y el default. Envía como resultado una nueva vista, usualmente una **jsp**.
- **"redirect"** - Le indica al navegador que debe redirigirse a una nueva página, que puede estar en nuestra misma aplicación o en algún sitio externo, y por lo tanto este creará una nueva petición para ese recurso.
- **"redirectAction"** - Redirige la petición a otro **Action** de nuestra aplicación. En este caso se crea una nueva petición hacia el nuevo **Action**.
- **"chain"** - Al terminarse la ejecución del **Action** se invoca otro **Action**. En este caso se usa la misma petición para el segundo **Action**, el cual se ejecuta de forma completa, con todo su stack de interceptores y sus **results** (podemos crear cadenas de cuantos **Actions** queramos).
- **"stream"** - Permite enviar un archivo binario de vuelta al usuario.

Global Results



- *Podemos indicar resultados que se aplicarán a múltiples actions.*
 - *Por ejemplo, a nivel de seguridad si un usuario no tiene privilegios de acceso se le redirige a la página de login.*
- *Primero se buscan results definidos dentro de Actions, si no coincide ninguno se comprueban los global results.*

Global Results

- *Un ejemplo*

```
<global-results>  
<result name="error">/Error.jsp</result>  
<result name="invalid.token">/Error.jsp</result>  
<result name="login" type="redirect-  
action">Logon!input</result>  
</global-results>
```



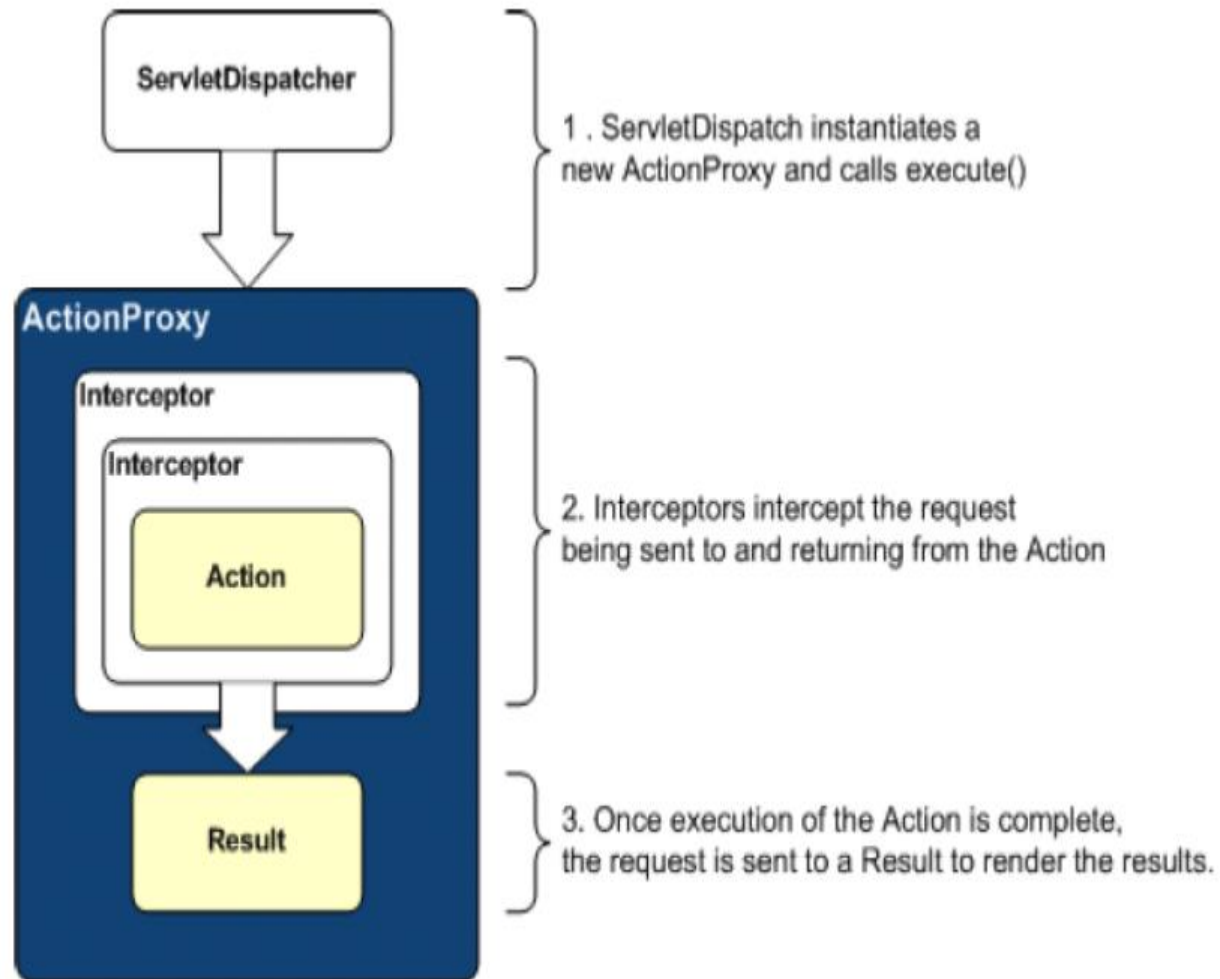
Struts

INTERCEPTORES

¿ Por que Interceptores?

- *Algunas acciones comparten “preocupaciones” (concerns) comunes.*
 - *Validación de Input*
 - *Procesamiento de subida de ficheros.*
 - *Protección contra doble submit de formularios.*
 - *Carga de listas antes de mostrar una página.*
- *Struts2 nos permite de una forma sencilla compartir soluciones para todas estas preocupaciones mediante Interceptores.*

Interceptores en el ciclo de vida de un Action



Características de los Interceptores

- *Pueden ejecutar código antes y después de la ejecución de un Action*
- *Gran parte de la funcionalidad básica de struts 2 está implementada con ayuda de interceptores.*
 - *Evitar doble-submit*
 - *Conversión de tipos*
 - *Validaciones, etc.*
- *Cada uno de los interceptores son configurables.*

Interceptors y Actions

Los interceptores:

- *Pueden en algunos casos evitar que un Action se ejecute.*
 - *En un doble submit o fallo en validación.*
 - *Pueden cambiar el estado de un Action antes de que se ejecute.*
 - *Son definidos en una pila que indica su orden de ejecución.*
 - *En ocasiones este orden puede ser muy importante.*
 - *Pueden ser configurados por cada Action.*
-
- *Struts2 permite la creación de Interceptors personalizados que pueden utilizarse conjuntamente con los proporcionados por defecto*

Struts-Interceptores

- Cada interceptor proporciona una característica distinta al Action.
- Para sacar la mayor ventaja posible de los interceptores, un Action permite que se aplique más de un interceptor. Para lograr esto Struts 2 permite crear pilas o stacks de interceptores y aplicarlas a los Actions.
- Cada interceptor es aplicado en el orden en el que aparece en el stack. También podemos formar pilas de interceptores en base a otras pilas
- Una de estas pilas de interceptores es aplicada por default a todos los Actions de la aplicación (definida en struts-default)
- Existen varias pilas pre-configuradas que podemos usar:

Interceptores de Struts2

- *Struts2 cuenta con un conjunto de interceptores “listos para utilizar”. Los más importantes son:*
 - *Parameter Interceptor: Establece los parámetros de la request dentro del Action*
 - *Validation Interceptor: Realiza validaciones definidas en el archivo action-validation.xml*
 - **Mas...**

Más importantes

Interceptor	Nombre	Descripción
Alias	alias	Permite que los parámetros tengan distintos nombres entre peticiones.
Chaining	chaining	Permite que las propiedades del Action ejecutado previamente estén disponibles en el Action actual
Checkbox	checkbox	Ayuda en el manejo de checkboxes agregando un parámetro con el valor "false" para checkboxes que no están marcadas (o checadas)
Conversion Error	conversionError	Coloca información de los errores convirtiendo cadenas a los tipos de parámetros adecuados para los campos del Action.
Create Session	createSession	Crea de forma automática una sesión HTTP si es que aún no existe una.
Execute and Wait	execAndWait	Envía al usuario a una página de espera intermedia mientras el Action se ejecuta en background.
File Upload	fileUpload	Hace que la carga de archivos sea más fácil de realizar.
Logging	logger	Proporciona un logging (salida a bitácora) simple, mostrando el nombre del Action que se está ejecutando.
Prepare	prepare	Llama al método "prepare" en los acciones que implementan la interface "Preparable"
Servlet Configuration	servletConfig	Proporciona al Action acceso a información basada en Servlets.
Roles	roles	Permite que el Action se ejecutado solo si el usuario tiene uno de los roles configurados.
Timer	timer	Proporciona una información sencilla de cuánto tiempo tardo el Action en ejecutarse.
Workflow	workflow	Redirige al result "INPUT" sin ejecutar el Action cuando una validación falla.

Struts - Interceptores

Nombre del Stack	Interceptores Incluidos	Descripción
basicStack	exception, servletConfig, prepare, checkbox, multiselect, actionMappingParams, params, conversionError	Los interceptores que se espera se usen en todos los casos, hasta los más básicos.
validationWorkflow Stack	basicStack, validation, workflow	Agrega validación y flujo de trabajo a las características del stack básico.
fileUploadStack	fileUpload, basicStack	Agrega funcionalidad de carga de archivos a las características del stack básico.
paramsPrepareParamsStack	alias, i18n, checkbox, multiselect, params, servletConfig, prepare, chain, modelDriven, fileUpload, staticParams, actionMappingParams, params, conversionError, validation, workflow	Proporciona un stack completo para manejo de casi cualquier cosa que necesitemos en nuestras aplicaciones. El interceptor "params" se aplica dos veces, la primera vez proporciona los parámetros antes de que el método "prepare" sea llamado, y la segunda vez re-aplica los parámetros a los objetos que hayan podido ser recuperados durante la fase de preparación.

Nombre del Stack	Interceptores Incluidos	Descripción
defaultStack	alias, servletConfig, i18n, prepare, chain, debugging, scopedModelDriven, modelDriven, fileUpload, checkbox, multiselect, staticParams, actionMappingParams, params, conversionError, validation, workflow	Es la pila que se aplica por default a todos los Actions de la aplicación.
executeAndWaitStack	execAndWait, defaultStack, execAndWait	Proporciona al stack básico las características de execute and wait, lo cual funciona para aplicaciones en las que deben subirse archivos que pueden tardar algún tiempo.

Interceptores personalizados

Podemos definir nuestros propios Interceptores de la siguiente forma:

- *Implementando interfaz `Interceptor`*
- *Heredar de la clase `AbstractInterceptor` que proporciona una implementación vacía de `init` y `destroy` que puede ser utilizada si estos métodos no van a ser implementados.*

```
public interface Interceptor extends Serializable {  
    void destroy();  
    void init();  
    String intercept(ActionInvocation invocation) throws  
        Exception;  
}
```


Configuración de Interceptors

```
<package name="default" extends="struts-default">  
  <interceptors>  
    <interceptor name="SimpleInterceptor "  
      class=".." />  
    <interceptor name="personalizado"  
      class=".." />  
  </interceptors>  
  <action name="login" class="tutorial.Login">  
    <interceptor-ref name="personalizado" />  
    <interceptor-ref name="SimpleInterceptor" />  
    <interceptor-ref name="defaultStack" />  
    <result name="input">login.jsp</result>  
    <result >/secure/home</result>  
  </action>  
</package>
```

Interceptores personalizados

Un ejemplo:

```
import com.opensymphony.xwork2.ActionInvocation;
import com.opensymphony.xwork2.interceptor.AbstractInterceptor;

public class SimpleInterceptor extends AbstractInterceptor {
    public String intercept(ActionInvocation invocation) throws Exception
    {
        MyAction action = (MyAction) invocation.getAction();
        action.setDate(new Date());
        return invocation.invoke();
    }
}
```

Múltiples interceptores

- También podremos definir y usar nuestra propia pila de intercep

```
<interceptor-stack name="basicStack">
  <interceptor-ref name="exception"/>
  <interceptor-ref name="servlet-config"/>
  <interceptor-ref name="prepare"/>
  <interceptor-ref name="checkbox"/>
  <interceptor-ref name="params"/>
  <interceptor-ref name="conversionError"/>
</interceptor-stack>
```

tores, por ejemplo:

```
<action name="hello"
class="ejemplo.MyAction">
  <interceptor-ref name="basicStack"/>
  <result>view.jsp</result>
</action>
```

Todos los interceptores

Interceptor	Configured Name	Description
Alias Interceptor	alias	Allows parameters to have different name aliases across requests; this is particularly useful when chaining actions with different names for the same information.
Chaining Interceptor	chaining	Allows the previously executed action's properties to be available to the current action; usually this interceptor is used with the result type "chain."
Checkbox Interceptor	checkbox	Assists in managing check boxes by adding a parameter value of false for check boxes that are not checked (usually there would be no information in the HTTP request).
Conversion Error Interceptor	conversionError	Places error information from converting strings to parameter types into the action's field errors.
Create Session Interceptor	createSession	Automatically creates an HTTP session (if one does not already exist).
Debugging Interceptor	debugging	Provides several different debugging screens to the developer.
Execute and Wait Interceptor	execAndWait	Sends the user to an intermediary waiting page while the action executes in the background.

Todos los interceptores

Interceptor	Configured Name	Description
Exception Interceptor	exception	Maps exceptions that are thrown from an action to a result, allowing automatic exception handling via redirection.
File Upload Interceptor	fileUpload	Facilitates easy file uploading.
Internationalization Interceptor	i18n	Keeps track of the selected locale during a user's session.
Logging Interceptor	logger	Provides simple logging by outputting the name of the action being executed.
Message Store Interceptor	store	Stores and retrieves the messages, field errors, and action errors in the session for actions implementing the ValidationAware interface.
Model Driven Interceptor	modelDriven	Places the model object onto the stack for actions implementing the ModelDriven interface.
Scoped Model Driven Interceptor	scopedModelDriven	Stores and retrieves the model object from a Interceptor configured scope for actions implementing the ScopedModelDriven interface.

Todos los interceptores

Interceptor	Configured Name	Description
Parameters Interceptor	params	Sets the request parameters on the action.
Parameter Filter Interceptor	n/a	Provides control over which parameters the action has access to (not configured by default).
Prepare Interceptor	prepare	Calls the prepare() method for actions implementing the Preparable interface.
Profiling Interceptor	profile	Allows simple profiling information to be logged for actions.
Scope Interceptor	scope	Stores and retrieves the action's state in the session or application scope.
Servlet Configuration Interceptor	servletConfig	Provides the action with access to various servlet-based information.
Static Parameters Interceptor	staticParams	Sets statically defined (param tags in the action's configuration) values on the action.
Roles Interceptor	roles	Allows the action to be executed only if the user is one of the configured roles.

Todos los interceptores

Interceptor	Configured Name	Description
Timer Interceptor	timer	Provides simple profiling information in the form of how long the action takes to execute.
Token Interceptor	token	Checks the action for a valid token to prevent duplicate formsubmission.
Token Session Interceptor	tokenSession	Same as token, but for invalid tokens, the submitted data is stored in the session.
Validation Interceptor	validation	Provides validation support for actions.
Workflow Interceptor	workflow	Redirects to an INPUT view without executing the action when validation fails.

Todos los interceptores

Stack Name	Included Intercepted	Description
basicStack	exception, servletConfig, prepare, checkbox, params, conversionError	The interceptors that are expected to be used in a minimal situation.
validationWorkflowStack	basicStack, validation, workflow	Add stacks features.s validation and workflow to the basic
fileUploadStack	fileUpload, basicStack	Adds file uploading to the basic stacks features.
modelDrivenStack	modelDriven, basicStack	Adds model functionality to the basic stacks features.
chainStack	chain, basicStack	Adds chaining to the basic stacks features.
i18nStack	i18n, basicStack	Adds locale persistence to the basic stacks features.
paramPrepareParamsStack	exception, alias, params, servletConfig, prepare, i18n, chain, modelDriven, fileUpload, checkbox, staticParams, params, conversionError, validation, workflow	Provides a complete stack including a pre-action method call. The params interceptor is applied twice: the first time to provide the parameters before the prepare() method is called, and a second time to reapply the parameters to objects that may have been retrieved during the prepare phase.

Todos los interceptores

Stack Name	Included Intercepted	Description
defaultStack	exception, alias, servletConfig, prepare, i18n, chain, debugging, profiling, scopedModelDriven, modelDriven, fileUpload, checkbox, staticParams, params, conversionError, validation, workflow	Provides a complete stack, including debugging and profiling.
executeAndWaitStack	execAndWait, defaultStack, execAndWait	Provides an execute and wait stack, which is useful for features such as file uploading where a waiting page is presented to the user.

EXCEPCIONES

Struts

- Excepciones. Operaciones con Struts.
 - Capturando excepciones de manera global:
 - Definición de la excepción a partir del fichero struts.xml (dentro del package adecuado y antes de la declaración de los action):

```
<global-results>  
    <result name="errorDesconocido">/error.jsp</result>  
    <result name="errorBBDD">/errorBBDD.jsp</result>  
</global-results>
```

```
<global-exception-mappings>  
    <exception-mapping exception="java.lang.Exception"  
result="errorDesconocido" />  
    <exception-mapping exception="java.sql.SQLException"  
result="errorBBDD" />  
</global-exception-mappings>
```

Struts

- Excepciones. Operaciones con Struts.
 - Mostrando la información de las excepciones:

Nombre de la exception:

```
<s:property value="exception" />
```

Detalle de la exception:

```
<s:property value="exceptionStack" />
```

Struts

Struts.xml

Struts.xml

- `<package>`.
 - Agrupación de acciones, resultados, tipos de resultados, interceptores y pilas de interceptores en unidades lógicas.
 - Conceptualmente, los paquetes son similares a las clases. Pueden ser extendidos, tienen partes individuales que pueden ser sobrescritas por “sub” paquetes
 - Atributos:
 - **name**: identifica el paquete. Obligatorio.
 - **namespace**: identifica la URL al paquete. Opcional.
 - **extends**: identifica el nombre del paquete del que hereda la información de configuración. Opcional.
 - **abstract**: agrupador de paquetes; el paquete no es accesible vía nombre del paquete. Opcional.

Struts.xml

- <include>.
 - Permite incluir otro fichero de configuración, estructurando la aplicación.
 - Atributos:
 - file: nombre del fichero de configuración.
 - Ejemplo:

```
<include file="example.xml"/>
```

Struts.xml

- <Tag bean>.
 - Permite definir beans globales.
 - Atributos:
 - class: clase que identifica el Bean. Obligatorio.
 - type: interfaz que implementa el Bean.
 - name: nombre del Bean.
 - scope: ámbito del Bean (default, singleton, request, session o thread).

Struts.xml

- <constant>
 - Sirve para configurar valores de struts.
 - Por ejemplo:

```
<constant name="struts.devMode" value="true" />
```

- Le indicará a Struts que nos encontramos en la etapa de desarrollo, con esto generará más mensajes de salida para que sepamos si estamos haciendo algo mal, y nos mostrará los errores de una forma más clara.

Struts.xml

- `<constant>`
 - Otra constante útil que podemos definir es `"struts.configuration.xml.reload"`. Esta constante permite que cuando modifiquemos los archivos de configuración de Struts 2 no tengamos que volver a hacer un deploy de la aplicación para que los cambios tomen efecto:

```
• <constant name="struts.configuration.xml.reload" value="true"/>
```

- Tendría el mismo efecto que:

- Añadir `struts.devMode = true` en el archivo `struts.properties`

- Añadir este fragmento de XML en el `Web.xml`:

```
<filter> <filter-name>struts</filter-name>  
  <filter-  
class>org.apache.struts2.dispatcher.filter.StrutsPrepareAndExecuteFilter<  
/filter-class>  
  <init-param>  
    <param-name>struts.devMode</param-name>  
    <param-value>true</param-value>  
  </init-param>  
</filter>
```



ValueStack , ActionContext y OGNL

ValueStack

- Para acceder a los objetos necesarios para los jsp, struts provee una pila de objetos que denominado value stack.
- Los objetos que struts2 pública en esta pila son:
 1. **Objetos temporales:** son los objetos creados durante la ejecución de un jsp, por ejemplo, al iterar una colección mediante un tag se crea una referencia al objeto de la iteración actual.
 2. **El Modelo de Objeto** (The model object): Si el action define un modelo de objetos entonces este es publicado en la pila. (un bean)
 3. **El Action:** El Action que se esta ejecutando.

ValueStack

- Es importante conocer el orden, debido a que al querer acceder a uno de estos objetos utilizando OGNL, primero se buscara desde el grupo de menor índice al de mayor índice (como esta enumerado más arriba).

ValueStack

- Podemos obtener una referencia al valueStack desde nuestro action:
 - ▣ `ValueStack stack =
 ActionContext.getContext().getValueStack();`
- Y poner/quitar elementos de la cima:
 - ▣ `stack.push(new Persona("paco",50))`
 - ▣ `Persona p=(Persona) stack.pop();`

ValueStack

- Añadir nuevos valores de propiedades de los elementos en la pila
 - ▣ `stack.setValue("nombre", "Paco Martinez");`
- Recuperar propiedades
 - ▣ `String nombrePersonap= stack.findString("nombre");`
- Importante : ValueStack a la hora de encontrar un valor, lo busca en toda la pila (en el orden que ya hemos visto) pero no se limita a la profundidad 1, si no que va al siguiente nivel de profundidad. Intenta hacer un `getNombre()` o `setNombre(" Paco Martinez")` en todos los objetos de la pila hasta que encuentra uno que devuelva algo o tenga el set determinado.

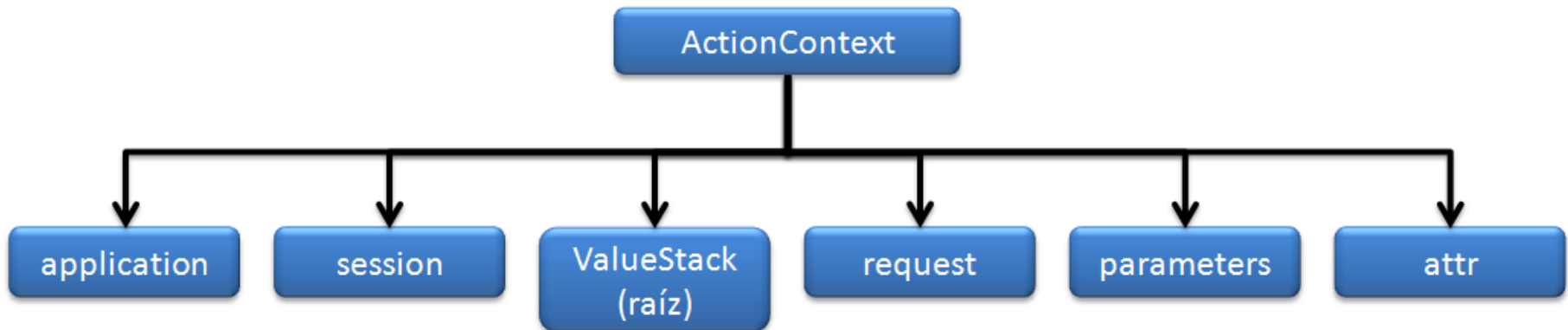
- También se pueden añadir elementos a la pila que sea recuperables directamente por el método `findValue`. Se hace:
 - `stack.set ("persona", new Persona("Pepe" ,54));`
 - `Persona pepe=(Persona) stack.findValue("Pepe")`

OGNL

- ❑ **OGNL** es el acrónimo de **Object Graph Navigation Language**, un lenguaje de expresiones muy poderoso que nos permite leer valores de objetos Java
- ❑ Se utiliza para hacer referencia y manipular los datos del **ActionContext**.
- ❑ **OGNL** ayuda en la transferencia de datos y conversión de tipos.

ActionContext map

- El mapa ActionController consiste en lo siguiente:
 - ▣ application- variables de ámbito de aplicación
 - ▣ Session – variables de ambito de session
 - ▣ ValueStack (raíz)
 - ▣ Request – variables en el ambito del request
 - ▣ Parameters – parametros del request
 - ▣ Atributes - los atributos almacenados en los ambitos de page,request, session y application



ActionContext en OGNL

- **ActionContext** es el contexto en el cual se ejecuta un **Action** (cada contexto es básicamente un contenedor de objetos que un **Action** necesita para su ejecución, como los objetos "**session**", "**parameters**", "**locale**", etc.), y el "**ValueStack**" como el objeto raíz.
- Podemos ver el contenido del **valueStack** y el **ActionContext** en un JSP para debugging con la etiqueta `<s:debug/>`
- El "**ValueStack**" es un conjunto de muchos objetos, pero para **OGNL** este aparenta ser solo uno el raíz..

OGNL

- En OGNL podemos acceder a los atributos (u otros objetos colgados) del objeto raíz (el valuestack), simplemente con el nombre
 - **<s:property value="nombre" />**
- Si queremos acceder a propiedades de otros elementos que no sean raíz, deberán ir precedidos por el nombre del elemento el #.
 - **<s:property value="#session.datoSesion" />**
 - **<s:property value="#parameters.dato" />**

OGNL

- También podemos ejecutar métodos de un objeto y obtener su salida simplemente indicándolo con paréntesis para diferenciarlo de una propiedad:
- **`<s:property value="getNombre()" />`**



ServletActionContext

ServletActionContext

- La clase `ServletActionContext` proporciona métodos para obtener los objetos `HttpServletRequest`, `HttpServletResponse`, `ServletContext` y `HttpSession`.
- Métodos más usados:
 - ▣ `public static HttpServletRequest getRequest ()` devuelve la instancia de `HttpServletRequest`.
 - ▣ `public static HttpServletResponse getResponse ()` devuelve la instancia de `HttpServletResponse`.
 - ▣ `public static ServletContext getServletContext ()` devuelve la instancia de `ServletContext`.

Session

- Podemos obtener la session:
 - ▣ `HttpSession session=ServletActionContext.getRequest().getSession();`



ModelDriven Actions

ModelDriven Actions

- Si una clase de acción implementa la interfaz `com.opensymphony.xwork2.ModelDriven`, entonces necesita devolver un objeto del método `getModel()`.
- Struts rellenará entonces los campos de este objeto con los parámetros del request, y este objeto se colocará encima de la pila una vez que se ejecute la acción.

ModelDriven Actions

□ Clase bean:

```
public class Persona {  
    String nombre;  
    int edad;  
    Getters y setters  
}
```

ModelDriven Action

□ ModelDriven Action

**public class PersonaAction extends ActionSupport implements
ModelDriven<Persona> {**

Persona model = new Persona();

public String execute(){

return *SUCCESS*;

}

getModel() y setModel(Persona p)

ModelDriven Action

□ Registro:

```
<body>  
<form action=actions/PersonaAction method="post">  
<input type="text" name="nombre"/>  
<input type="text" name="edad"/>  
<input type="submit"/>  
</form>  
</body>
```

ModelDriven Action

□ Struts.xml:

```
<action name="PersonaAction" class="actions.PersonaAction">  
  <result name="success">/resultado.jsp</result>  
</action>
```

□ resultado.jsp:

```
<body>  
  <h1>La persona es</h1>  
  <h3>${nombre} </h3>  
  <h3>${edad} </h3>  
</body>
```