



Apache Jakarta Struts



Contenido

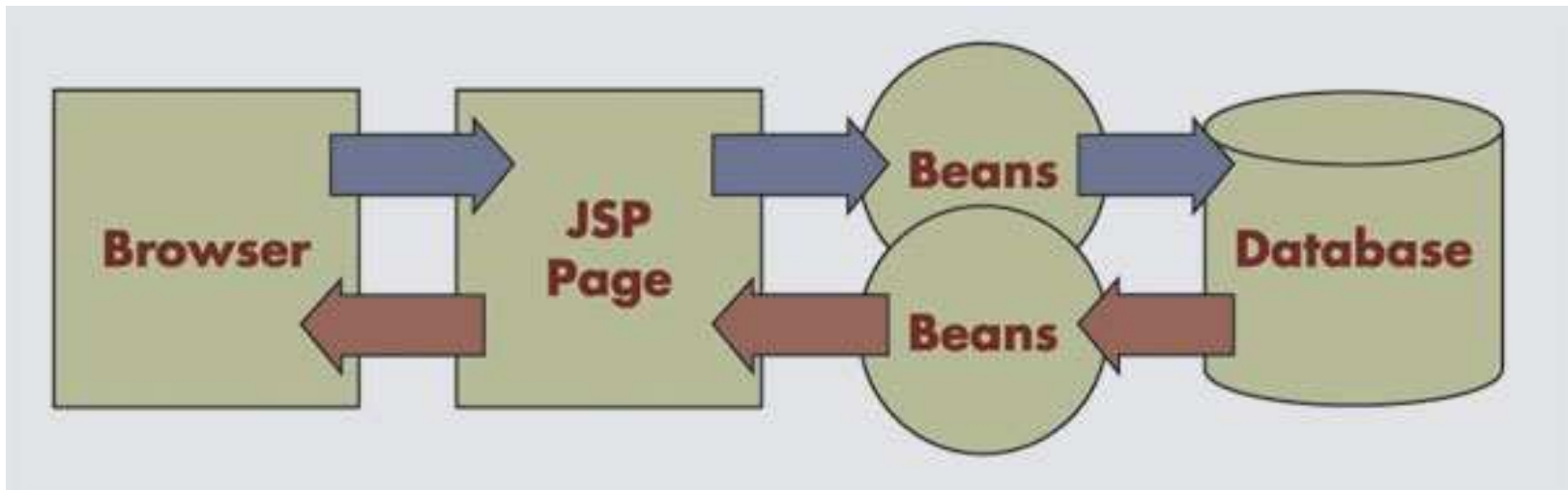
- Patrón de diseño MVC
- El framework Apache Struts
 - Instalación
 - Flujo de control en Struts
 - Procesamiento de peticiones
 - Control de errores
 - Manejo de Formularios
 - Librerías de etiquetas de Struts
 - Internacionalización
- Ejemplo de aplicación Struts



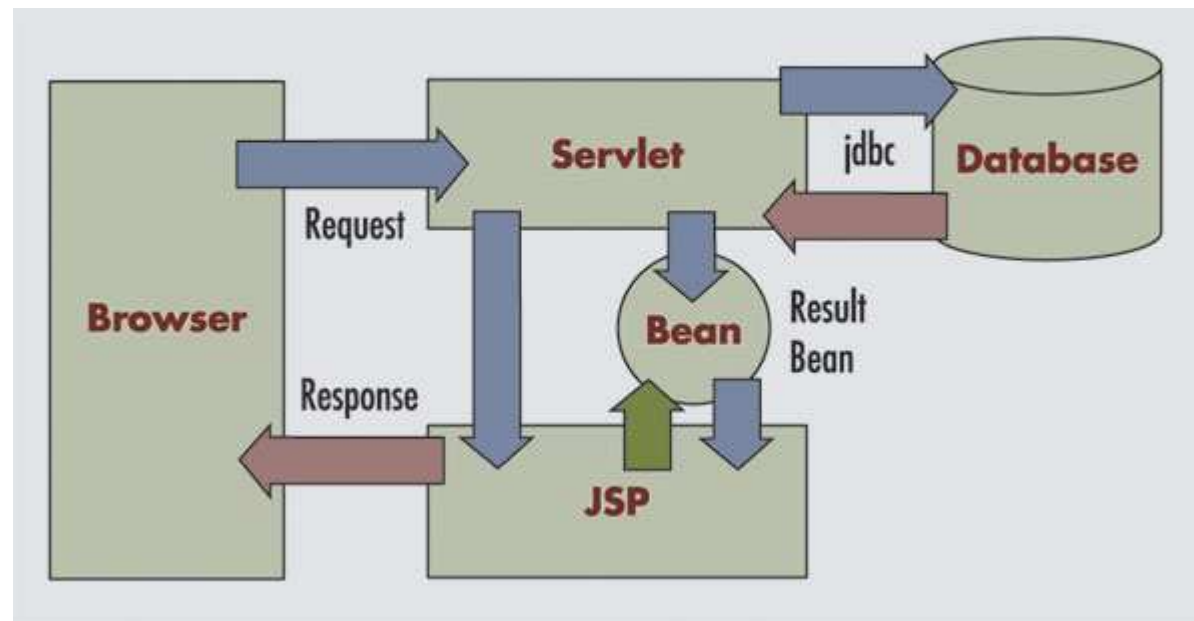
Modelos de desarrollo de aplicaciones Web en Java

- Los servlets son buenos ejecutando lógica de negocio, pero no son tan buenos presentando información
- JSPs son muy buenos presentando pero pésimos introduciendo lógica programática en ellos
- La combinación Servlet/JSPs es lo más común hoy en día en el desarrollo de aplicaciones web
- Dos arquitecturas:
 - Model-1: JSPs para presentación y control y JavaBeans para la lógica
 - Model-2: Model-View-Controller = JavaBeans-JSPs-Servlets
 - MVC es tan común que se han desarrollado varias infraestructuras en torno a este patrón de diseño:
 - Apache Struts
 - Java Server Faces

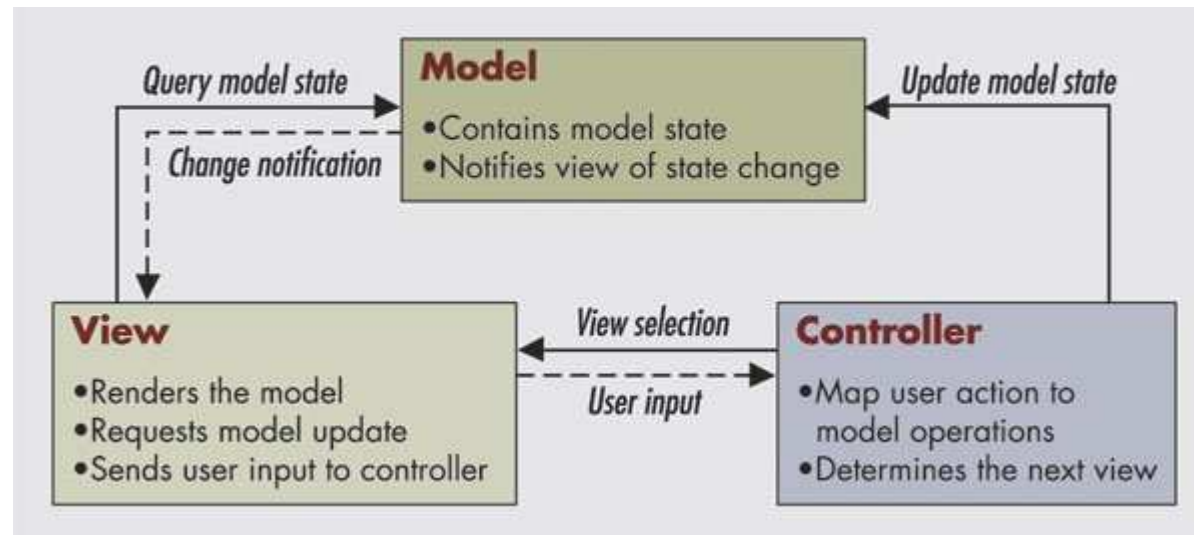
Arquitectura Model 1



Arquitectura Model 2



Modelo MVC 0





Modelo MVC I

- El Controlador (Controller)
 - Servlet central recibe peticiones, procesa URL recibida y delega procesamiento a JavaBeans
 - Servlet guarda resultado de procesamiento realizado por JavaBeans en el contexto de la petición, la sesión o la aplicación
 - Servlet transfiere control a un JSP que lleva a cabo la presentación de resultados



Modelo MVC II

- El Modelo (Model)
 - JavaBeans (o EJBs para aplicaciones más escalables) juegan el rol de modelo:
 - Algunos beans ejecutan lógica
 - Otros guardan datos
 - Normalmente:
 1. Servlet controlador invoca un método en bean lógico y éste devuelve un bean de datos
 2. Autor de JSP tiene acceso a bean de datos



Modelo MVC III

- La Vista (View)
 - Rol ejecutado por JSPs
 - Servlet Controlador transfiere control al JSP después de haber guardado en un contexto el resultado en forma de un bean de datos
 - JSP usa `jsp:useBean` y `jsp:getProperty` para recuperar datos y formatear respuesta en HTML o XML



Modelo MVC IV

- En resumen:
 - Los beans o EJBs ejecutan la lógica de negocio y guardan los resultados
 - Los JSPs proveen la información formateada
 - Los servlets coordinan/controlan la ejecución de los beans y los JSPs



Frameworks

- Las frameworks pueden ser vistas como implementaciones de patrones de diseño que facilitan la reutilización de diseño y código
- Dado que MVC ha sido utilizado en muchas aplicaciones web, el desarrollo de frameworks que da soporte a áreas comunes en todas las aplicaciones MVC es necesario
 - Apache Struts es una de estas frameworks
- Aplicaciones basadas en Struts consistirán de:
 - Código Java
 - Deployment descriptors que configuran la framework para el uso de nuestra aplicación



Apache Jakarta Struts/Struts 2.0

- Implementación del modelo 2/patrón de diseño MVC que facilita la creación de aplicaciones web en Java.
- Creada por Craig McClanahan y donada a la Apache Software Foundation en el 2000 (pertenece a Apache Jakarta).
- Ahora, **Struts 2.0** impulsado por Patrick Lightbody, cuyo objetivo era atraer líderes de diferentes Frameworks Web en uno único. Unión de WebWork y Struts
- WebWork (2 proyectos):
 - Xwork → generic command framework
 - WebWork → dependiente del contexto.



Características

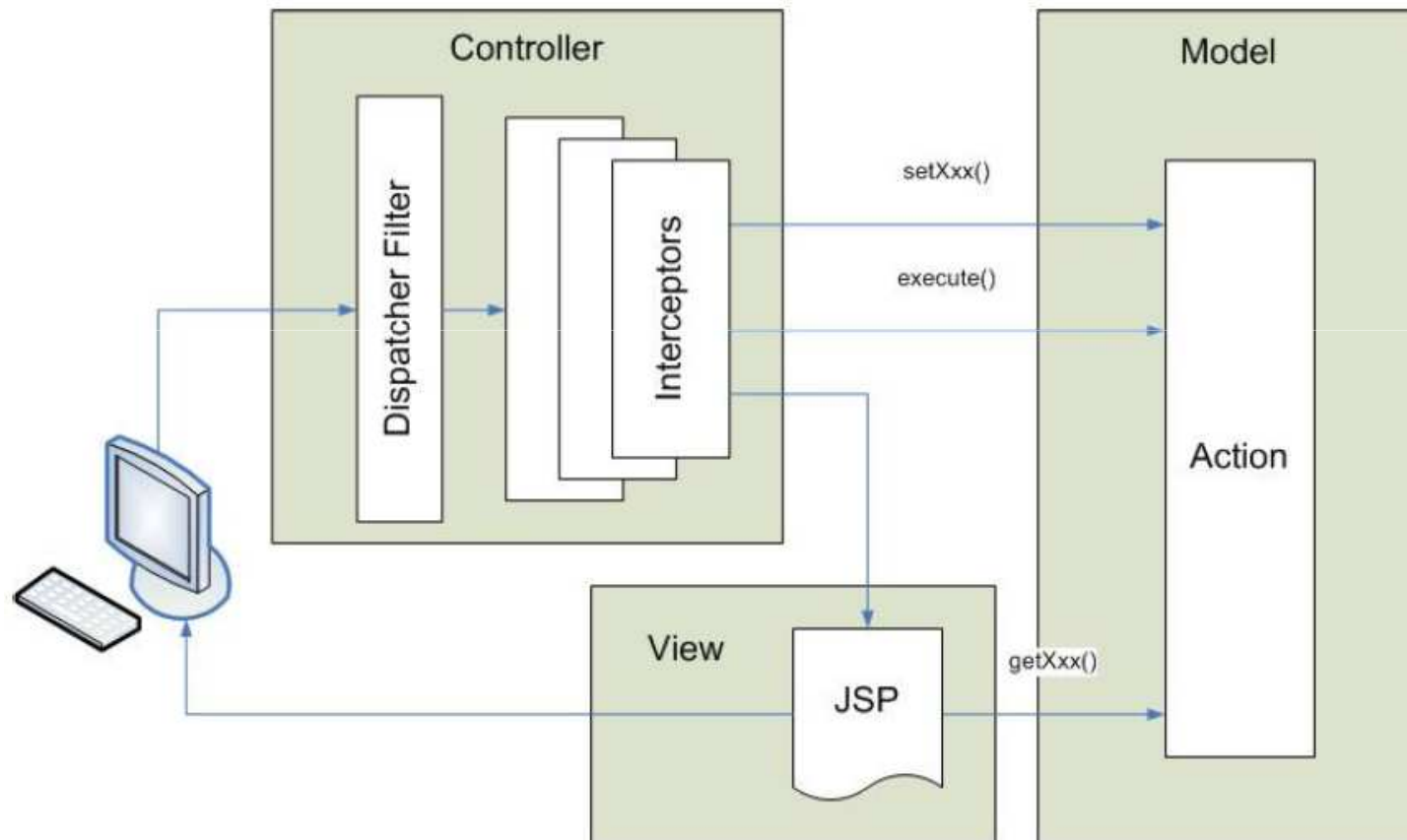
- Framework basado en Actions.
- Un gran grupo de desarrolladores y comunidad de usuarios.
- Anotaciones y opciones de configuración XML.
- POJO-basados en acciones que son fáciles de usar.
- Integración con Spring, SiteMesh y Tiles.
- Integración con el lenguaje de expresión OGNL.
- **Object-Graph Navigation Language**. Lenguaje de expresión para hacer get y set de propiedades sobre objetos java.
- Temas basados con los tags libraries y Ajax.
- Múltiples opciones de vista (JSF, Freemarker, Velocity and XSLT).
- Plu-ins para ampliar y modificar el framework.



Componentes de Struts (I)

- El MVC en Struts2 viene realizado en 5 componentes:
 - Actions.
 - Interceptores.
 - Pila de valores/OGNL.
 - Tipo de resultados.
 - Tecnologías de vista de resultados.

Componentes de Struts (II)

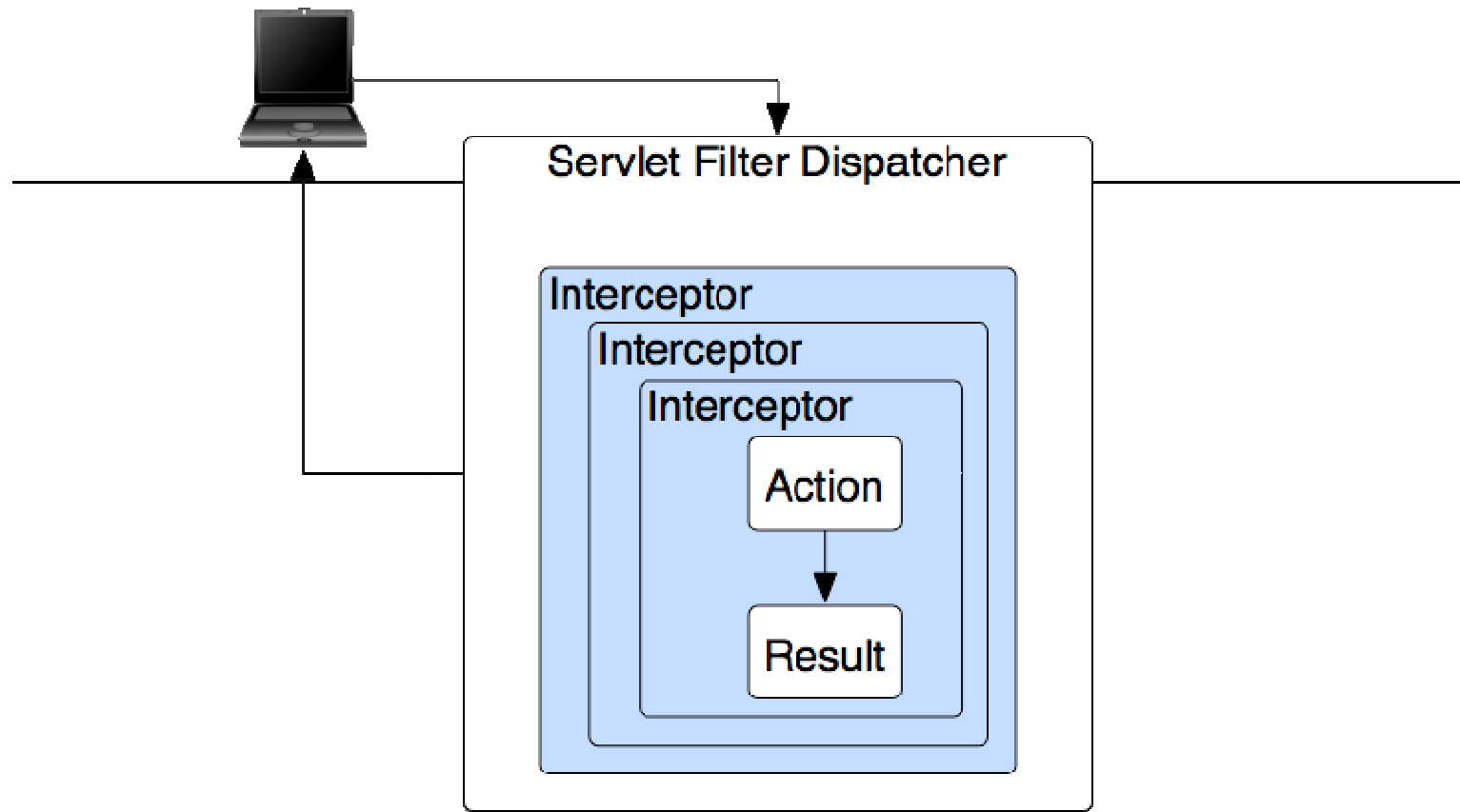




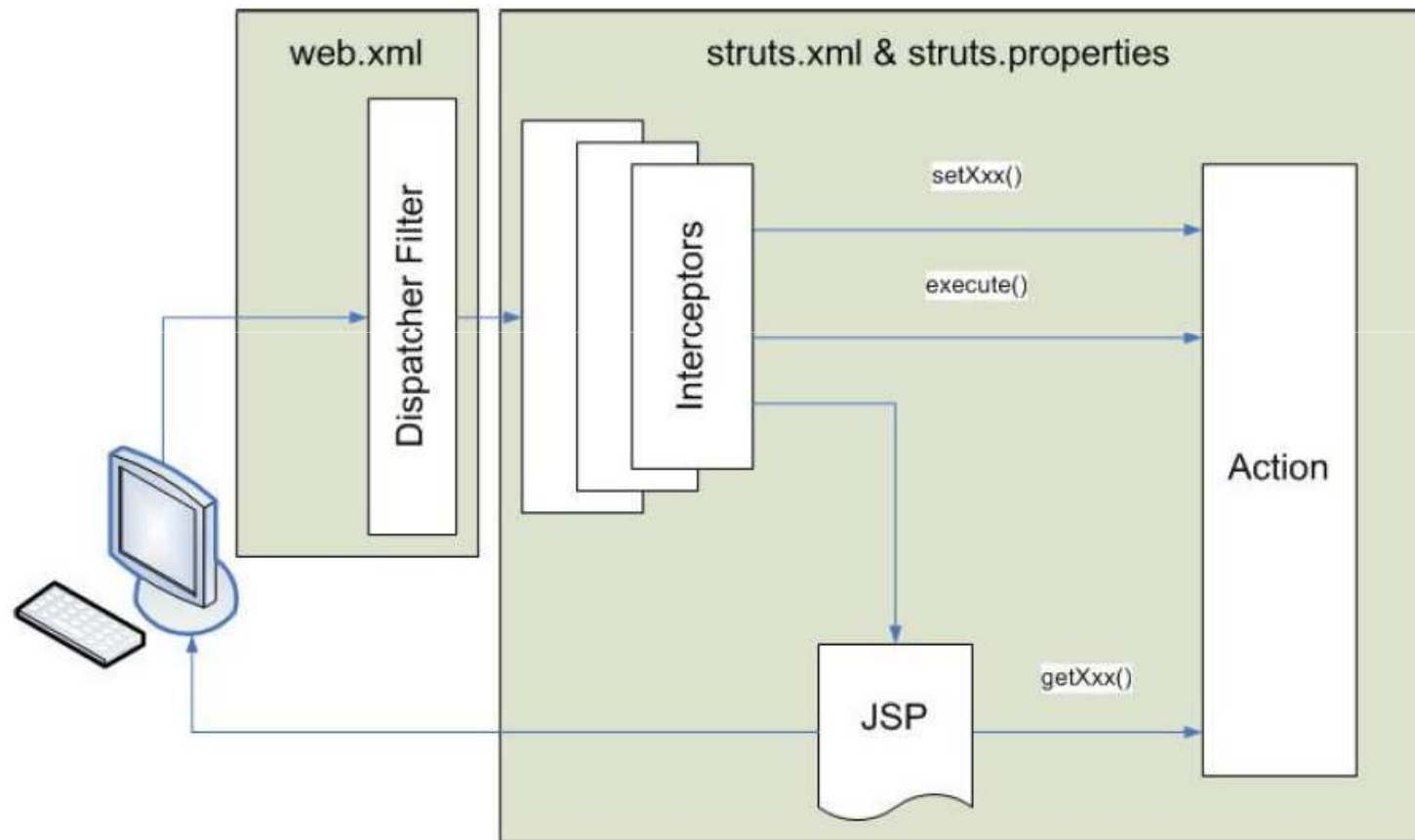
Componentes de Struts (III)


- El **controlador** está implementado por un dispatch server filter como interceptor.
- El **modelo** es implementado con acciones.
- La **vista** con una combinación de tipo de resultados y resultados.
- La pila de valores y OGNL se compone de hilos comunes, enlaces y integración entre todos los componentes.

Struts 2 Architecture



Configuración de ámbito de fichero para los elementos del framework





Configuración de la aplicación FilterDispatcher (web.xml)

```
<filter>
  <filter-name>action2</filter-name>
  <filter-class>
    org.apache.struts2.dispatcher.FilterDispatcher
  </filter-class>
</filter>
<filter-mapping>
  <filter-name>action2</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```



Fichero struts.properties

- Este fichero provee un mecanismo para cambiar el comportamiento por defecto del framework.
- Normalmente, no se necesita modificar este fichero.
- Todas las propiedades contenidas en este fichero se pueden configurar en el fichero "web.xml" con el tag "init-param".
- O en el fichero "struts.xml" con el tag "constant".
- Para más información sobre como modificar las propiedades:
<http://struts.apache.org/2.x/docs/strutsproperties.html>
- Contiene un fichero de propiedades denominado "default.properties" que esta contenido en el Jar de distribución.
- Para modificar una propiedades, simplemente crear un fichero llamado "struts.properties" en el directorio src del proyecto.



Entorno de desarrollo (struts.properties)

- **struts.i18n.reload = true** – enables reloading of internationalization files
- **struts.devMode = true** – enables development mode that provides more comprehensive debugging
- **struts.configuration.xml.reload = true** – enables reloading of XML configuration files (for the action) when a change is made without reloading the entire web application in the servlet container
- **struts.url.http.port = 8080** – sets the port that the server is run on (so that generated URLs are created correctly)



Fichero struts.xml

- Contiene las acciones desarrolladas.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts
Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <package name="struts2" extends="struts-
default"      namespace="/struts2">
    ...
  </package>
</struts>
```



Fichero struts.xml (Include tag)

- Sirve para modular la aplicación para incluir otros ficheros de configuración.
- Contiene el atributo "file"

```
<struts>  
    <include file="billing-config.xml" />  
    <include file="admin-config.xml" />  
    <include file="reports-config.xml" />  
    ...  
</struts>
```
- El orden de incluir los ficheros es importante. El uso de un tag que es configurado en otro fichero, la configuración del include debe ocurrir antes de ser referenciado.
- Además, hay ficheros incluidos explícitamente, que son incluidos automáticamente. "struts-default.xml" y "struts-plugin.xml". Ambos contienen configuraciones por defecto para los tipos de resultado, interceptores, pila de interceptor, paquetes como también información para el entorno de ejecución de la aplicación Web.



Fichero struts.xml (Package tag) I

- Este tag es empleado para agrupar configuraciones que comparten atributos compartidos como puede ser la pila de interceptores o namespace de URL.
- Principalmente, consiste en la configuración de las acciones, pero debe incluir cualquier tipo de información de configuración.
- Además, se emplea para organizar funciones separadas, las cuales deben ser separadas en diferentes ficheros de configuración.



Fichero struts.xml (Package tag) II

- Los atributos de este tag son:
 - *name* – a developer provided unique name for this package.
 - *extends* – the name of a package that this package will extend; all configuration information (including action configurations) from the extended package will be available in the new package, under the new namespace.
 - *namespace* – the namespace provides a mapping from the URL to the package. i.e. for two different packages, with namespace attributes defined as “package1” and “package2”, the URLs would be something like “/myWebApp/package1/my.action” and “/myWebApp/package2/my.action”
 - *abstract* – if this attribute value is “true” the package is truly configuration grouping, and actions configured will not be accessible via the package name



Fichero struts.xml (Package tag) III

- En muchos casos la configuración parte del fichero "struts-default.xml".
- Aunque, cuando se emplea el plug-ins esto cambia, se debería verificar la vinculación del nombre del paquete con la documentación del plug-ins.
- Hay dos elementos de configuración adicionales que pueden ser empleados con el tag <struts>, son:
 - <bean ... />
 - <constant ... /> tags.
- Estos tags ofrecen alternativas avanzadas de re-configuración avanzada del framework.



Actions

- Los Actions son un concepto fundamental en muchos framework de aplicación web.
- Son la unidad más básica de trabajo que se asocia con una request HTTP invocada por el usuario.
- En Struts 2 una acción puede ser empleado en dos caminos diferentes.
 - Single result
 - Multiple result



Single result

- Uso más básico, devuelve un único resultado, ejemplo:

```
class MyAction {  
    public void String execute() throws Exception {  
        return "success";  
    }  
}
```

- Primero, la clase no necesita extender de ninguna otra clase, ni implementar ningún interface. Simple POJO.
- Segundo, la clase tiene un método llamado "execute" (empleado por convención).
- La configuración más simple para un objeto Action es algo como:

```
<action name="my" class="com.fdar.infoq.MyAction" >  
    <result>view.jsp</result>  
</action>
```
- El atributo "name" provee la URL para ejecutar el action, en este caso una URL de "\my.action".
- La extensión ".action" es configurada en el fichero "struts.properties".
- El atributo "class" indica el paquete y nombre de la clase del action para ser ejecutado.



Multiple Results

- Se da cuando el action puede retornar diferentes resultados en función de la lógica de salida:

```
class MyAction {  
    public void String execute() throws Exception {  
        if( myLogicWorked() ) {  
            return "success";  
        } else {  
            return "error";  
        }  
    }  
}
```

- En el fichero de configuración

```
<action name="my" class="com.fdar.infoq.MyAction" >  
    <result>view.jsp</result>  
    <result name="error">error.jsp</result>  
</action>
```

- Siempre se toma el primer valor como "success", por defecto.



Result Types

- Los resultados que son generados y enviados al usuario desde un action para diferentes valores no necesitan ser de mismo tipo. Ejemplo:
 - El “success” debe generar una JSP
 - El “error” debe ser enviado un HTTP header back al navegador.
- El tipo de resultado es configurado con el atributo “type” sobre el nodo result.



Request and Form Data

- Sobre como un action debería trabajar y proveer datos para el objeto de persistencia de base de datos, el action necesita acceder a los valores desde la request como desde los datos del form.
- Struts2 se basa en el concepto de JavaBean. Para acceder a los datos con los getter and/or setter para ese campo. Cada parámetro de la request o dato de un formulario es un simple par de **nombre/valor**. Por ejemplo, Si una JSP hace una llamada a **"/home.action?framework=struts&version=2"** el action necesitaría proveer un setter `"setFramework(String frameworkName)"` como también un setter `"setVersion(int version)"`.



Acceso a Servicios de Empresa

- Un action antes de retornar un resultado, a veces, necesita procesar la ejecución.
- Se necesita acceder a diferentes objetos – business objects, data access objects o otros recursos.
- Struts2 emplea una técnica llamada *dependency injection*, o inversion of control.
- *Dependency injection*, puede ser implementada por constructor injection, **interface injection y setter injection**.
- Esto significa que para tener objetos disponibles para un action, se necesita tener solamente un setter.
- Hay además objetos que no maneja el framework de Spring, como es:
 - El objeto HttpServletRequest. Estos son manejados por la combinación de setter injection y interface injection.
 - Por cada objeto non-business hay un interface correspondiente (conocido como un interface “aware”) que el action es requerido de ser implementado.



Acceso a los datos desde el action

- Una técnica conocida es accediendo en `HttpServletRequest` o `elHttpSession`. Esto se puede realizar por la implementación de un interface "aware" interface y el setting del objeto para acceder bajo el name
- Si la intención es emplear *tag libraries* o *JSTL*, el acceso a los datos es mucho más fácil. Ambos métodos son directamente accesible desde al action desde el *Value Stack*. Trabajo adicional es proveer getters sobre el action que permite el acceso a los objetos que se necesitan para acceder.



Interceptors (I)

- Muchos de las capacidades en Struts2 son la implementaciones usando interceptores, ejemplos:
 - include exception handling.
 - file uploading.
 - lifecycle callbacks and validation.
- Los Interceptors son conceptualmente lo mismo que un servlet filters o la clase Proxy.
- Proveen un camino para suplir pre-processing y post-processing a través del action. Similar a servlet filters, los interceptors pueden ser secuencializados y ordenados. Tienen acceso al contexto de ejecución del action, a las variables del entorno y propiedades de ejecución.



Interceptors (II)

- Se trabaja con la asistencia de los siguientes interfaces:
 - SessionAware
 - ServletRequestAware (acceso al objeto `HttpServletRequest`)
 - RequestAware
 - ApplicationAware
 - ServletResponseAware
 - ParameterAware (acceso a los atributos de la request y formularios en forma de Map)
 - PrincipalAware (Acceso al Principal que representa la identidad del usuario, sus roles, etc.)
 - ServletContextAware (Acceso al objeto de contexto)



Interceptors (Configuración) I

- Para configurar un nuevo interceptor, se necesita definir el interceptor

```
<interceptors>
...
<interceptor name="miInterceptor "
class="com.miw.MiInterceptor"/>
</interceptors>
```

- Se necesita asegurar que el interceptor es aplicado al action correspondiente.

- Esto puede realizarse en dos caminos:

- El primero, asignar el interceptor a cada action individualmente:

```
<action name="my" class="com.fdar.infoq.MyAction" >
<result>view.jsp</result>
<interceptor-ref name=" miInterceptor "/>
</action>
```

Usando esta configuración no hay limitación del número de interceptores que se pueden aplicar a un action, Requerimiento, **los interceptores son listados en el orden que ellos son ejecutados.**

- El segundo camino es asignar un interceptor por defecto para el paquete actual:

```
<default-interceptor-ref name=" miInterceptor "/>
```

Esta declaración es realizada directamente bajo el tag <package ... /> tag y sólo un interceptor puede ser asignado por defecto.



Interceptors (Configuración) II

- Ejemplo de declaración de pila de interceptores en "struts-default.xml":

```
<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>
```
- Este nodo de configuración esta declaro bajo el nodo<package ... />.
- Cada tag <interceptor-ref ... /> referencia a otro interceptor o una pila de interceptor stack que ha sido configurada con anterioridad a la actual pila.

```
<action name="my" class="com.fdar.infoq.MyAction" >  
  <result>view.jsp</result>  
  <interceptor-ref name="basicStack"/>  
</action>
```
- Se configura igual que un interceptor por defecto – simplemente usa un interceptor

```
<default-interceptor-ref name="basicStack"/>
```



Implementación de Interceptores

- El interface que necesita implementar es simple, y proviene de Xwork framework. Tiene 3 métodos:

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



Value Stack

- The value stack es exactamente que lo que dice está – una pila de objetos.
- El value stack contiene los siguientes objetos en el orden:
 1. *Temporary Objects* – Durante la ejecución, hay objetos *temporales* que son instanciados y que se añaden a la pila. Por ejemplo, todos los elementos iterados en una colección desde una JSP son añadidos al Value Stack.
 2. *The Model Object* – Si se utiliza un objeto de modelo (terminología Struts2), éste se añade también a la pila.
 3. *The Action Object* – El Action ejecutado..
 4. *Named Objects* – #application, #session, #request, #attr y #parameters
- El acceso a value stack puede ser realizado de diferentes formas. El más común es vía tags desde JSP, Velocity and Freemarker.



OGNL

- OGNL stands for Object Graph Navigational Language, y provee un camino unificado para acceder a los objetos dentro de value stack.
- Ejemplo: “person.address” instancia ainstead of “getPerson().getAddress()” como expresión.
- OGNL soporta:
 - type conversion.
 - calling methods.
 - collection manipulation.
 - generation.
 - rojection across collections.
 - expression evaluation.
- Guía completa:
<http://www ognl.org/2.6.9/Documentation/html/LanguageGuide/index.html>.



Result Type

- Struts2 soporta muchos tipos de resultados. Estos pueden ser visuales, o interacciones con el entorno.
- Para configurar un action que ejecute un resultado específico, el atributo "type" es empleado. Si no se emplea, toma por defecto el type "dispatcher", que renderiza una JSP.
- Ejemplo:

```
<action name="my" class="com.fdar.infoq.MyAction">  
<result type="dispatcher">view.jsp</result>
```



Result Types (Configuración)

- Son configurados dentro del tag `<package ... />`.
- La configuración es similar a la configuración de un interceptor.
- Un atributo "name" provee un único identificador para el result type, y el atributo "class" provee la clase de implementation. El atributo "default" permite modificar el valor por defecto de result type.

```
<result-types>
  <result-type name="dispatcher" default="true"
    class="....dispatcher.ServletDispatcherResult"/>
  <result-type name="redirect"
    class="....dispatcher.ServletRedirectResult"/>
  ...
</result-types>
```



Implementación de Result Types

- Similar a interceptors, es posible crear uno propio y configurarlo en la aplicación web

```
public interface Result extends Serializable {  
    public void execute(ActionInvocation invocation)  
    throws Exception;  
}
```
- El objeto “ActionInvocation” provee acceso a runtime environment, `permitiendo al nuevo result type acceder a la información del form del action que ha sido ejecutado. El contexto incluye el objeto “HttpServletRequest”, el cual provee acceso a output stream de la actual request.



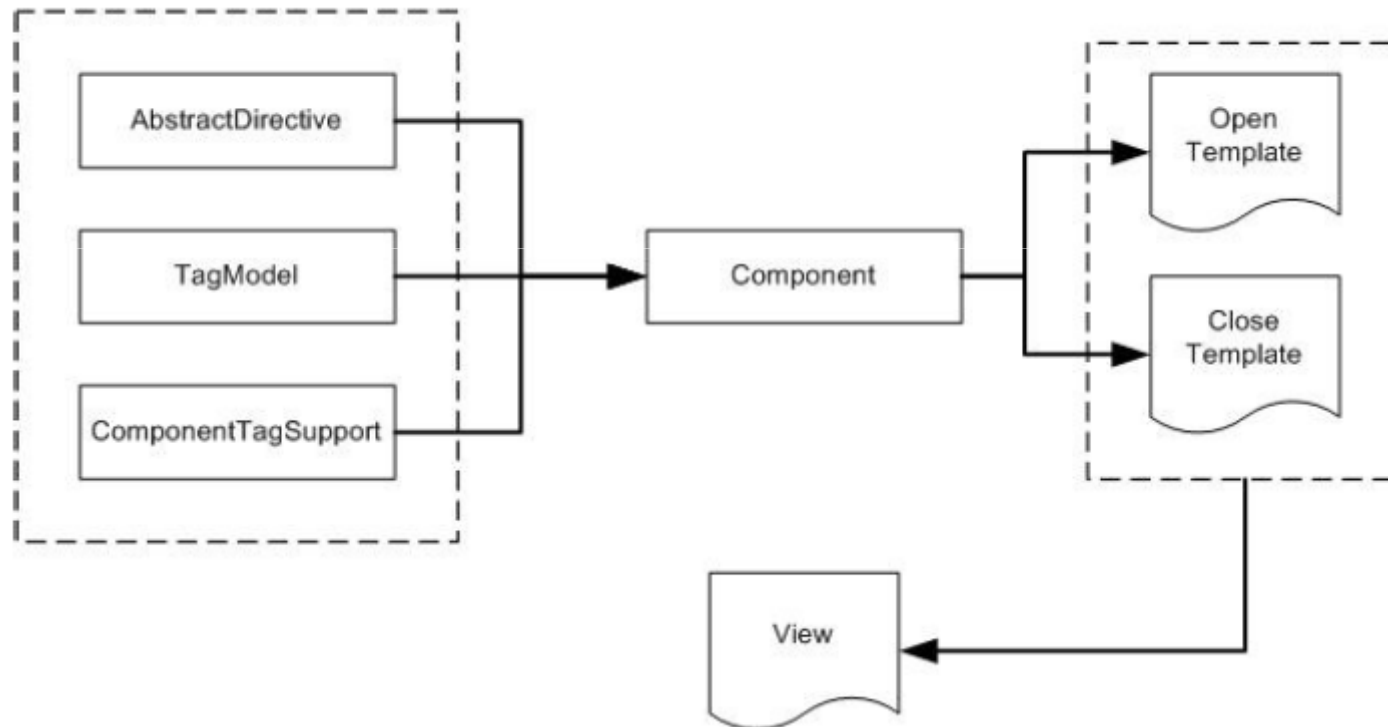
Result /View Technologies

- Aparte de resultado de JSP, hay otras tres tecnologías en Struts2:
 - Velocity Templates
 - Freemarker Templates
 - XSLT Transformations
- Como el resultado de Freemarker debería ser configurado en cambio de una JSP:

```
<action name="my" class="com.fdar.infoq.MyAction" >  
  <result type="freemarker">view.ftl</result>  
</action>
```
- El result XSLT varía. No solo cambia el nombre con stylesheet name, permite parámetros adicionales
 - El parámetro "stylesheetLocation" provides the name of the stylesheet to use in rendering the XML. If this parameter is not present, the untransformed XML will be returned to the user.
 - El parámetro "exposedValue" provides the property of the action, or an OGNL expression to be exposed as XML. If this parameter is not specified, the action itself will be exposed asXML.

```
<result type="xslt">  
  <param name="stylesheetLocation">render.xslt</param>  
  <param name="exposedValue">model.address</param>  
</result>
```

Tag Libraries (I)





Tag Libraries (II)

- El nucleo de su arquitectura es un conjunto de componentes.
- El objeto componente
 - representa a cada tag en su forma más básica.
 - Provee la lógica de gestión y renderizado necesaria.
- Cada tecnología de vista procee su propio wrapper para cada componente.
- El Wrapper adapta el renderizado de cada componente a la tecnología que se esté usando.