Spring

Sesión 2: Spring MVC

© 2006-2007 Depto. Ciencia de la Computación e IA

Indice

- Spring MVC vs. Struts
- Procesamiento de una petición
- Configuración básica
- Caso 1: petición sin entrada de datos
- Caso 2: petición con datos de entrada y validación

© 2006-2007 Depto. Ciencia de la Computación e IA

Spring MVC vs. Struts

- En general los dos frameworks ofrecen cosas similares
  - “Controllers” de Spring vs. Acciones de Struts
  - “Commands” de Spring vs. ActionForms de Struts
  - “Validators” de Spring vs. ValidatorForms de Struts
  - Taglibs de Spring vs. las de Struts
  - …
- Spring tiene una arquitectura mejor diseñada, más completa y configurable… lógico, ya que es mucho más moderno que Struts 1.X.

© 2006-2007 Depto. Ciencia de la Computación e IA

Procesamiento de una petición

Handler
Mapping

2

3

Request

1

Dispatcher
Servlet

Controller

ModelAndView

4

5

ViewResolver

6

View

© 2006-2007 Depto. Ciencia de la Computación e IA

Configuración básica (web.xml)

```
<servlet>
        <servlet-name>spring</servlet-name>
        <servlet-class> org.springframework.web.servlet.DispatcherServlet </servlet-
class>
        <load-on-startup>1</load-on-startup>
        <init-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>/WEB-INF/classes/mvc.xml</param-value>
        </init-param>
</servlet>
<servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>*.mvc</url-pattern>
</servlet-mapping>
```


© 2006-2007 Depto. Ciencia de la Computación e IA

Enfoque que vamos a seguir

- Spring MVC ofrece quizá demasiadas alternativas
- Mejor que ver las cosas sistemáticamente (imposible en 2 horas), vamos a ver dos casos particulares y sin embargo muy típicos
  - **1: Petición HTTP para recuperar datos** (ej: datos de un pedido, lista de todos los clientes, todos los libros,…)
  - **2: Petición HTTP con entrada de datos y obtención de resultados** (vale, en realidad eso son 2 peticiones, pero en Spring se hace todo “en el mismo código”).

© 2006-2007 Depto. Ciencia de la Computación e IA

Caso 1: petición sin datos de entrada

- Ejemplo: un hotel que ofrece las “ofertas de la semana” en su página

```
package es.ua.jtech.spring.dominio;
import java.math.BigDecimal;
import java.util.Date;
public class Oferta {
		private BigDecimal precio;
		private Date fechaLimite;
		private TipoHabitacion tipoHab;
		private int minNoches;
		//..aquí vendrían los getters y
setters
}
```

```
package es.ua.jtech.spring.dominio;

public enum TipoHabitacion {
		individual,
		doble
}
```

 © 2006-2007 Depto. Ciencia de la Computación e IA

El Controller (≡ Action de Struts) (I)

- Podemos heredar de distintos tipos, el mejor aquí es AbstractController
- Necesitaremos algún objeto de negocio que “sepa sacar las ofertas de la semana” (sup. interface GestorOfertas). Usaremos dependency injection

```
package es.ua.jtech.spring.mvc;
import es.ua.jtech.spring.negocio.GestorOfertas;
Import org.springframework.web.servlet.mvc.AbstractController;
public class ListaOfertasController extends AbstractController {
		private GestorOfertas miGestor;
		public void setMiGestor(GestorOfertas miGestor) {
			this.miGestor = miGestor;
		}
}
```


© 2006-2007 Depto. Ciencia de la Computación e IA

El Controller (≡ Action de Struts) (II)

- Debemos configurar los 2 beans anteriores en el XML
- Usaremos un HandlerMapping por defecto, similar al de Struts: (/loquesea.mvc ⇨ Controller)

```
<beans xmlns="http://www.springframework.org/schema/beans"
                          ...>
  <bean name="/verOfertas.mvc" class="es.ua.jtech.spring.mvc.ListaOfertasController">
            <property name="miGestor" ref="miGestorOfertas"/>
  </bean>
  <bean id="miGestorOfertas" class="es.ua.jtech.spring.negocio.GestorOfertasDummy">
  </bean>
</beans>
```


© 2006-2007 Depto. Ciencia de la Computación e IA

El Controller (≡ Action de Struts) (III)

- ¡Nos falta que el Controller haga algo útil!: en el método handleRequestInternal.

```
package es.ua.jtech.spring.mvc;
//.. Faltan los import
public class ListaOfertasController extends AbstractController {
		private GestorOfertas miGestor;
		public void setMiGestor(GestorOfertas miGestor) {
			this.miGestor = miGestor;
		}
		@Override protected ModelAndView handleRequestInternal(HttpServletRequest arg0,
							HttpServletResponse arg1) throws Exception {
			ModelAndView mav = new ModelAndView("ofertas");
			List<Oferta> ofertas = miGestor.getOfertasActuales();

			mav.addObject("ofertas",ofertas);
			return mav;
		}
}
```


© 2006-2007 Depto. Ciencia de la Computación e IA

Resolver el nombre de la vista

- Necesitamos un ViewResolver, de los que hay varias implementaciones en Spring. Uno de los más sencillos es el InternalResourceViewResolver
- En este caso, por ejemplo, result ⇨/jsp/result.jsp

```
<bean id="viewResolver"

		class="org.springframework.web.servlet.view.InternalResourceViewResolver">
		<property name="preffix" value="/jsp/"/>
		<property name="suffix" value=".jsp"/>
</bean>
```


© 2006-2007 Depto. Ciencia de la Computación e IA

La página JSP

- No tiene nada especial de Spring

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core"prefix="c"%>
<html>
<head> <title>Ejemplo de vista</title> </head>
<body>
		<h1>Superofertas de la semana</h1>
		<c:forEach items="${ofertas}" var="o">
				Habitación ${o.tipoHab} un mínimo de ${o.minNoches} noches por
solo						${o.precio} &euro; la noche<br/>
		</c:forEach>
</body>
</html>
```


© 2006-2007 Depto. Ciencia de la Computación e IA

Caso 2: petición con procesamiento de datos de entrada

- Ejemplo: buscar ofertas por precio y tipo de habitación
- Esto en Struts normalmente lo haríamos con 2 acciones
  - Mostrar formulario
  - Validar datos, disparar lógica de negocio
- En Spring se puede hacer solo con un controller: SimpleFormController.
- Hay un paralelismo en validación de datos
  - ActionForm ⇔Command
  - Método validate() ⇔ clase que implementa Validator
  - Ambos usan ficheros .properties

 © 2006-2007 Depto. Ciencia de la Computación e IA

El command (≡ ActionForm de Struts)

- **Diferencia:** no tiene que heredar de nada especial ni implementar ningún interfaz especial, solo ser un *javabean*

```
package es.ua.jtech.spring.mvc;
import java.math.BigDecimal;
import es.ua.jtech.spring.dominio.TipoHabitacion;

public class BusquedaOfertas {
		private BigDecimal precioMax;
		private TipoHabitacion tipoHab;
		//..ahora vendrían los getters y setters
}
```


© 2006-2007 Depto. Ciencia de la Computación e IA

El controller

- Ahora heredaremos de SimpleFormController
- Tenemos que decirle
  - Qué clase es el Command
  - Nombre simbólico para el Command
  - Qué vista mostrar si hay error de validación
  - Qué vista mostrar si todo va bien
- Ahora el trabajo se hace en un método llamado onSubmit(), que se llama una vez se han rellenado los datos (no tenemos que preocuparnos de mostrar el formulario inicialmente)

© 2006-2007 Depto. Ciencia de la Computación e IA

El controller

```
package es.ua.jtech.spring.mvc;
//Faltan los import...
public class BuscarOfertasController extends SimpleFormController {
		private GestorOfertas miGestorOfertas;
		public BuscarOfertasController() {
				setCommandName("busquedaOfertas");
				setCommandClass(BusquedaOfertas.class);

				setFormView("buscarOfertas");
				setSuccessView("resultBuscarOfertas");
		}
		public void setMiGestorOfertas(GestorOfertas miGestorOfertas) {
								this.miGestorOfertas =
miGestorOfertas;
		}
		@Override protected ModelAndView onSubmit(Object command) throws
Exception {
				BusquedaOfertas bo = (BusquedaOfertas) command;
				List<Oferta> encontradas =
miGestorOfertas.buscarOfertas(bo.getPrecioMax(),
```


© 2006-2007 Depto. Ciencia de la Computación e IA

La validación

- Igual que en Struts, hay dos tipos:
  - Programada
  - Declarativa: usa jakarta commons validator…¿familiar?
- Veremos aquí la programada, la otra es cuestión de configuración
- Donde en Struts programarías un método validate() en un ActionForm aquí es un objeto aparte(Validator)

© 2006-2007 Depto. Ciencia de la Computación e IA

El objeto Validator

```
package es.ua.jtech.spring.mvc;
import org.springframework.validation.Errors;
import org.springframework.validation.ValidationUtils;
import org.springframework.validation.Validator;
public class OfertaValidator implements Validator {
		public boolean supports(Class arg0) {
				return arg0.isAssignableFrom(BusquedaOfertas.class);
		}
		public void validate(Object obj, Errors errors) {
				ValidationUtils.rejectIfEmpty(errors, "precioMax",
"precioVacio");
				 BusquedaOfertas bo = (BusquedaOfertas) obj;
				//comprobar que el precio no esté vacío (para que no haya null
pointer
				//más abajo)
				if (bo.getPrecioMax()==null) return;
				//comprobar que el número sea positivo
				if (bo.getPrecioMax().floatValue()<0)
						errors.rejectValue("precioMax", "precNoVal");
```


© 2006-2007 Depto. Ciencia de la Computación e IA

Los mensajes de error

- Configuración: ¿dónde están los mensajes?

```
<bean id="messageSource"
        class="org.springframework.context.support.ResourceBundleMessageSour
ce">    <property name="basename">
                <value>mensajes</value>
        </property>
</bean
```

- Archivo mensajes.properties (en el CLASSPATH)

```
precioVacio = el precio está vacío
precNoVal = precio no válido
typeMismatch.precioMax = el precio no es un número
```


© 2006-2007 Depto. Ciencia de la Computación e IA

Falta configurar el controller

- Referencia al validator definido

```
<bean name="/buscarOfertas.mvc"
        class="es.ua.jtech.spring.mvc.BuscarOfertasController">
        <property name="miGestorOfertas" ref="miGestorOfertas"/>
        <property name="validator">
                <bean class="es.ua.jtech.spring.mvc.OfertaValidator"/>
        </property>
</bean>
```


© 2006-2007 Depto. Ciencia de la Computación e IA

La vista con las taglibs de Spring

```
<%@ taglib uri="http://www.springframework.org/tags" prefix="spring" %>
<html>
	<head></head>
<body>
	<h1>Búsqueda de ofertas</h1>
	<form action="" method="post">
		<spring:bind path="busquedaOfertas.precioMax">
			Precio máximo: <input type="text"
name="precioMax"
	value="${status.value}"/>
						${status.errorMessage} <br/>
		</spring:bind>
		Tipo de habitación:
		<select name="tipoHab">
			<option>individual</option>
<option>doble</option>					</select> <br/>
		<input type="submit" value="Buscar"/>
	</form>
</body>
</html>
```


© 2006-2007 Depto. Ciencia de la Computación e IA

¿Preguntas...?