Spring

Índice

1. Introducción

El objetivo de esta sesión de integración es introducir Spring en nuestro proyecto. Para poder aprovecharnos de las ventajas que nos da el *framework*, tendremos que refactorizar una pequeña parte del código.

Como el proyecto de integración en el futuro usará EJBs y otras tecnologías alternativas a Spring, **crearemos una rama** del mismo. Lo mejor es crear esta rama a partir del proyecto web tal y como lo tengáis ahora, si es que está en un estado consistente o bien que toméis la plantilla que se os dejó en el proyecto web (con una implementación parcial).Necesitaréis las clases de la capa de negocio.

Aviso:

Si no os ha dado tiempo a implementar la capa de negocio, bastará por el momento con que simplemente "hagan de puente" con los DAO: es decir, una llamada a un método de la capa de negocio llama directamente al DAO sin chequear nada ni nada de lógica de negocio.

Modificaremos los dos proyectos, por lo que habrá que hacer una rama para cada uno. Recordad que, para crear la nueva rama:

- Desde el menú de contexto de los proyectos, seleccionamos **Team > Branch**.
- El nombre de la nueva rama será: "b09-integracion-Spring".
- Introducir el nombre de la versión del proyecto: "v08-integracion-web
- Marcar la casilla *Start working in the branch immediately*.
- Pulsar **OK**.

Nuestros objetivos al introducir Spring van a ser los siguientes:

- Hacer uso del contenedor de beans para instanciar los objetos de la aplicación, en lugar de usar las factorías de objetos codificadas en Java que usábamos hasta el momento.
- Hacer accesibles los beans gestionados por Spring a las acciones de Struts
- Añadir transaccionalidad declarativa a las operaciones que así lo requieran, eliminando la necesidad de controlar las transacciones en el código Java.
- **De manera opcional**, hacer accesible algún método de negocio de la aplicación a clientes remotos.

2. Configuración y preparación del proyecto

Básicamente, necesitamos instalar las librerías de spring (contenidas en `spring.jar`) y configurar y arrancar el contenedor de beans. Hay que tener en cuenta que las capas gestionadas por Spring (negocio y datos, ya que la vista está gestionada por Struts) están en el proyecto común. Por tanto, tanto el propio Spring como la configuración de los beans deberían residir en el común. Sin embargo, el uso del contenedor de beans se hará en el proyecto web, por lo que el código que cree y acceda al contenedor debe residir en el web.

Por tanto debemos seguir estos pasos:

1. Copiar el archivo `spring.jar` a la carpeta `/lib` del proyecto común
2. Como el API de Spring también es necesario en el proyecto web, en las propiedades de éste, bajo la opción `J2EE dependencies`, añadir el `spring.jar` contenido en el proyecto común. Así nos aseguramos de que tanto se referencie al compilar el web como se copie en el `/WEB-INF/lib` durante el despliegue.
3. El XML con la configuración de los beans (archivo `beans.xml`) debería ir en el `resources` del proyecto común, pero también es necesario tenerlo en el proyecto web ya que allí es donde se va a instanciar el contenedor. Desgraciadamente, al referenciar un proyecto desde otro en Eclipse no copia los archivos individuales, sino que los mete en un .JAR, donde Spring no podrá encontrar el XML con la configuración de los beans. Lo más sencillo si no se usa Ant (aunque no lo mejor desde el punto de vista de la automatización) es copiar el mismo archivo en el `resources` tanto del proyecto común como del web.
4. Para poder editar de manera más cómoda los archivos de configuración de beans, es recomendable activar el plugin de Spring para eclipse en los proyectos. Para ello, recordad que:
   - Pulsando sobre la carpeta del proyecto con el botón derecho, hay que elegir la opción `Add spring project nature`
   - En las propiedades del proyecto, en el apartado `Spring`, añadir como nuevo `config file` el XML creado. Así se activará el chequeo de sintaxis y el autocompletado.
5. En el `web.xml` del proyecto web, añadimos el *listener* que cargará el contenedor de beans y lo hará accesible a las acciones de Struts. Justo detrás de la etiqueta `display-name`, y antes de los `servlet`, pondremos:

```
<!-- arranque del contenedor de Spring -->
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/classes/beans.xml</param-value>
</context-param>

<listener>
  <listener-class>
```

```
            org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>
```

3. Conversión de las capas de negocio y acceso a datos a Spring

En el código actual, cada clase es responsable de instanciar los otros objetos de la aplicación que necesita para trabajar. Vamos a liberar a las clases de sus responsabilidades pasándoselas al contenedor de Spring. De este modo, nos evitamos tener que codificar las factorías de objetos. Si queremos cambiar una implementación concreta de una interfaz por otra, lo haremos en el XML de configuración de Spring sin necesidad de recompilar una sola línea de código.

Pasos a seguir:

1. En el código Java de las clases, cuando se necesite una referencia a otro objeto de negocio o datos, suponer que va a ser gestionada por Spring. En nuestro caso, esto nos lleva a sustituir por ejemplo en `UsuarioBO` el código del estilo:

```
FactoriaDAOs fd = FactoriaDAOs.getInstance();
IUsuarioDAO iu = fd.getUsuarioDAO();
```

   por algo como:

```
private IUsuarioDAO iu;

public void setIu(UsuarioDAO iu) {
        this.iu = iu;
}
```

2. En el fichero de configuración de beans (archivo `beans.xml` de la carpeta `resources` en ambos proyectos), por cada objeto de negocio o acceso a datos, tendremos que definir un bean. Cuando un objeto dependa de otros tendremos que poner estas dependencias como propiedades del bean si queremos que Spring las gestione. Siguiendo con el ejemplo anterior, la configuración tendría algo como:

```
<bean id="usuarioBO" class="es.ua.jtech.proyint.bo.usuario.UsuarioBO">
        <property name="iu">
                <ref bean="usuarioDAO"/>
        </property>
</bean>

<!-- ¡¡faltaría la configuración de UsuarioDAO!! -->
```

3. A partir de este momento, las clases `FactoriaXXX` ya no son necesarias y podemos eliminarlas de nuestra aplicación.

Podríamos aprovechar las factorías de objetos que ya tenemos codificadas en Java si por las razones que fueran prefiriéramos usarlas en lugar de poner la clase concreta en el fichero de configuración. En lugar de poner el atributo `class` con la clase concreta del bean bastará decirle a Spring que hay que obtenerlo de una factoría (atributo `factory-bean`) llamando a un cierto método (atributo `factory-method`). En nuestro ejemplo:

```
<bean id="usuarioBO" factory-bean="factoriaBOs"
factory-method="getUsuarioBO">
        <property name="iu">
                <ref bean="usuarioDAO"/>
        </property>
</bean>

<!-- ¡¡faltaría la configuración de UsuarioDAO!! -->
```

Podéis obtener un gráfico con los beans creados y sus dependencias con el plugin de Spring para eclipse si abrís la vista llamada `"Spring beans"` y pulsáis con el botón derecho sobre el fichero de configuración, eligiendo la opción `"Show graph"` en el menú contextual.

Es fundamental comprobar que todo funciona correctamente. Aunque para ello habría que cambiar todos los test de cactus para que ahora accedan a los objetos de negocio y datos llamando al contenedor de beans no sería realista intentar hacer esto en la sesión por cuestiones de tiempo. Por ello, bastará con hacer al menos un test que obtenga el bean `usuarioBO` y llame a algún método de negocio (con esto indirectamente comprobamos la capa de acceso a datos).

4. Combinación de Struts con Spring

Sería demasiado laborioso refactorizar la capa de presentación entera para usar Spring MVC, y probablemente no estaría justificado, ya que con Struts se han resuelto razonablemente bien todos los requerimientos de la aplicación a este nivel. Por tanto, optamos por combinar la capa de presentación en Struts con el resto de capas en Spring.

Las acciones de Struts necesitan acceder a los beans de Spring a través del contenedor. Una primera posibilidad es usar código como el que sigue para instanciar los beans. Por ejemplo, en la acción de login podríamos hacer:

```
public class AccionLogin extends Action {

        private static Log logger =
LogFactory.getLog(AccionLogin.class.getName());

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

                ActionMessages errors = new ActionMessages();
                HttpSession sesion = request.getSession(true);
                String forward = "";

                LoginForm aForm = (LoginForm) form;

                ServletContext sc = getServlet().getServletContext();
                WebApplicationContext wac =
WebApplicationContextUtils.getWebApplicationContext(sc);
                IUsuarioBO iu = (IUsuarioBO) wac.getBean("usuarioBO");

                try {
                        UsuarioTO usuario =
iu.autenticaUsuario(aForm.getLogin(), aForm.getPassword());
                        ...
```

Pero el código anterior, además de no ser muy elegante, introduce en nuestras acciones de Struts una dependencia no deseable del API de Spring. ¿Qué pasaría si quisiéramos cambiar de Spring a otro contenedor que use también *dependency injection* pero no tenga el mismo API?. Una solución mejor es usar *dependency injection* en nuestras acciones de Struts, para que el código quede como:

```
public class AccionLogin extends Action {
```

```
        private static Log logger =
LogFactory.getLog(AccionLogin.class.getName());

        private IUsuarioBO iu;

        public void setIu(IUsuarioBO iu) {
                this.iu = iu;
        }

        @Override
        public ActionForward execute(ActionMapping mapping, ActionForm
form,
                        HttpServletRequest request,
HttpServletResponse res)
                        throws Exception {
                ActionMessages errors = new ActionMessages();
                HttpSession sesion = request.getSession(true);
                String forward = "";

                LoginForm aForm = (LoginForm) form;

                try {
                        UsuarioTO usuario =
iu.autenticaUsuario(aForm.getLogin(), aForm
                                        .getPassword());
                        ...
```

Desde luego, hemos tenido que modificar algo el código con respecto al original, pero no hay ni rastro del API de Spring. No obstante ¿cómo es posible que Spring resuelva la dependencia si esto es una acción de Struts y no un bean de Spring?. La respuesta está en que **necesitamos algo de configuración adicional para que Spring tome la acción de Struts como si fuera un bean y le inyecte las dependencias**. Necesitamos:

1. Configurar un plugin de Struts necesario para integrarlo con Spring. Al final de nuestro `struts-config.xml`, en la parte donde tenemos colocada ahora la etiqueta `plug-in` para el *validator*, pondremos:

   ```
   <plug-in
   className="org.springframework.web.struts.ContextLoaderPlugIn">
           <set-property property="contextConfigLocation"
                                   value="/WEB-INF/classes/beans.xml"/>
   </plug-in>
   ```

   Suponiendo que efectivamente nuestro archivo de configuración de beans se llama `beans.xml` y que está colocado en `resources` (es decir, que al compilar, eclipse lo colocará en `/WEB-INF/classes`).

2. Sustituir el controlador normal de Struts por otro propio de Spring, que se ocupará de

resolver las dependencias pasando al final el control a Struts. En el `struts-config.xml`, justo detrás de los `<action-mappings/>`, pondremos:

```
<controller
processorClass="org.springframework.web.struts.DelegatingRequestProcessor"/>
```

3. Por cada acción de Struts debemos definir un bean de Spring. Esto podríamos hacerlo en el mismo archivo `beans.xml`, pero desde el punto de vista lógico, ya que son beans de la capa de presentación, es mejor hacerlo en un XML aparte. Lo haremos en el proyecto web, dentro de la carpeta `resources`, en el archivo `strutsbeans.xml` (por el momento vacío). Para que se carguen todos los beans cuando se arranque la aplicación web tendremos que modificar ligeramente el *listener* que pusimos al principio en el `web.xml`:

```
<!-- arranque del contenedor de Spring -->
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/classes/beans.xml,/WEB-INF/classes/strutsbeans.xml</param-value>
</context-param>

<listener>
  <listener-class>
            org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>
```

Nota:

Antes de ponernos a editar el `strutsbeans.xml` conviene que le digamos al plugin Spring de eclipse que también es un fichero de configuración de beans. Para ello, en las propiedades del proyecto web, dentro del apartado `Spring`, añadiremos este como nuevo archivo dentro de la solapa `"Config files"`. Además, para poder referirnos desde un archivo a los beans contenidos en el otro, en la solapa `"Config sets"` crearemos un *set* nuevo (con el nombre que queramos) en el que marcaremos tanto `beans.xml` como `strutsbeans.xml`

4. Ahora ya podemos definir en `strutsbeans.xml` un bean por acción. A cada bean hay que darle un `name` que coincidirá con el `path` que tiene la acción en el `struts-config.xml`. La `class` del bean será el `type` que viene en el `struts-config.xml`. Es decir, que si en el `struts-config.xml` tenemos:

```
<action path="/accionLogin"
```

```
type="es.ua.jtech.proyint.presentacion.acciones.loginlogout.AccionLogin"
        name="loginForm" scope="request" input="/index.jsp">
        <forward name="admin" path="/accionListarUsuarios.do" />
        <forward name="biblio" path="/accionListarTodosLibros.do" />
        <forward name="socio" path="/jsp/socio/index.jsp" />
</action>
```

Necesitaremos este bean en `strutsbeans.xml`

```
<bean name="/accionLogin"
class="es.ua.jtech.proyint.presentacion.acciones.loginlogout.AccionLogin">
   <property name="iu">
          <ref bean="usuarioBO"/>
   </property>
</bean>
```

Nótese que la propiedad del bean se pone para que Spring resuelva automáticamente la dependencia.

Nótese que al hacer esto, lo que estamos consiguiendo en realidad es que a la petición "accionLogin.do" responda Spring en lugar de Struts, que se encargará de instanciar la acción de Struts de la clase necesaria, inyectarle las dependencias y finalmente pasar el control a Struts.

Nota:

Esta no es la única posibilidad que existe para que una acción de Struts tenga acceso a los beans de Spring con inyección de dependencias. También podemos hacer que nuestra acción de Struts herede de una clase propia de Spring, pero esto implica cambios en el código fuente y sobre todo dependencia del API de Spring.

5. Transaccionalidad declarativa

Codificar los métodos será más sencillo si dejamos que Spring se ocupe de gestionar las transacciones y hacer *rollback* en caso necesario. En nuestro caso, los métodos más claramente candidatos a beneficiarse de esto son los que se ocupan de realizar un préstamo y/o una reserva.

Se propone **hacer transaccional como mínimo el proceso de reserva de un libro** con la anotación `@Transactional`. Para ello habrá que hacer los siguientes cambios en el código Java:

- Elegir el nivel al que queremos hacer la transacción. Se puede hacer en negocio, si es que tenemos implementado `LibroBO` o bien en el DAO. Lo único que hay que tener en cuenta es que el método que deseamos hacer transaccional debe ser `public`.
- Sustituir en el DAO los métodos que ahora obtienen y cierran la conexión por los que proporciona Spring a través de `DataSourceUtils`.
- Eliminar el código que gestiona manualmente la transacción, es decir quitar el `conn.setAutoCommit(false)` y el `conn.rollback()` que se llama cuando se genera una `SQLException`
- Para que sea más fácil comprobar que se hace el rollback, cambiar el orden de las operaciones para que primero se cambie el estado del usuario y luego se inserte el registro en la tabla de operaciones. Así será sencillo hacer que la transacción falle haciendo una reserva de un usuario correcto pero de un isbn inexistente.

Por supuesto, también habrá que introducir las modificaciones necesarias para activar las transacciones declarativas en el archivo `beans.xml`.

Finalmente, comprobar que la transaccionalidad funciona y que efectivamente se deshace el cambio de estado del usuario si hay una DAOException.

6. Acceso remoto (OPCIONAL)

De manera opcional, se propone **hacer accesible a clientes remotos el método para listar todos los libros de `LibroBO`** (o de `LibroDAO`, si es que todavía no tienes terminada la capa de negocio).

Para ello necesitarás definir un nuevo interfaz que solo tenga esta operación, hacer que tu clase implemente **también** esta interfaz y configurar adecuadamente el bean en el XML, haciendo que corresponda con esta clase y con el nuevo interfaz (para que de forma remota solo sea visible el método de listar libros y no el resto).

Nota:

Hacer accesible el DAO como bean remoto solo es aceptable como solución práctica para no obligarte a implementar el objeto `LibroBO` si no lo tienes hecho, pero no es la mejor solución si queremos que un tercero acceda a nuestros métodos, ya que siempre debería pasar por la capa de negocio. La excepción sería una aplicación distribuida en la que la capa de negocio estuviera en una máquina y la de acceso a datos en otra distinta.¡Pero precisamente para este tipo de aplicaciones es para las que son indicados los EJBs!

7. Entrega

Como habéis visto, en este proyecto no se han creado nuevas clases, solo se han refactorizado las existentes, borrado las que no se necesitaban y creado ficheros de configuración.

- Habrá 2 ficheros de definición de beans: strutsbeans.xml y beans.xml. De este último habrá una copia también en el proyecto común.
- Las clases `FactoriaFuenteDatos`, `FactoriaBOs` y `FactoriaDAOs` no son necesarias con Spring.
- Todas las acciones de Struts deben obtener los BOs o los DAOs que necesiten a través de Spring
- Todos los BOs deben obtener los DAOs o los otros BOs que necesiten a través de Spring
- Todos los DAOs deben obtener el datasource a través de Spring