Capítulo 6. Integrando la Aplicación Web con la Capa de Persistencia

Esta es la Parte 6 del tutorial paso a paso sobre como desarrollar una aplicación web desde cero usando Spring Framework. En la Parte 1 hemos configurado el entorno y puesto en marcha una aplicación básica. En la Parte 2 hemos mejorado la aplicación que habíamos construido hasta entonces. En la Parte 3 hemos añadido toda la lógica de negocio y los tests unitarios, y en la Parte 4 hemos desarrollado la interface web. En la Parte 5 hemos desarrollado la capa de persistencia. Ahora es el momento de integrarlo todo junto en una aplicación web completa.

6.1. Modificar la Capa de Servicio

Si hemos estructurado nuestra aplicación adecuadamente, sólo tenemos que cambiar la capa de servicio para que haga uso de la persistencia en base de datos. Las clases de la vista y el controlador no tienen que ser modificadas, puesto que no deberían ser conscientes de ningún detalle de la implementación de la capa de servicio. Así que vamos a añadir persistencia a la implementación de ProductManager. Modifica la clase `SimpleProductManager` y añade una referencia a la interface `ProductDao` además de un método setter para esta referencia. Qué implementación usemos debe ser irrelevante para la clase ProductManager, a la cual se le inyectará el DAO de manera automática a través del mé todo llamado `setProductDao`. El método `getProducts` usará ahora este DAO para recuperar la lista de productos. Finalmente, el método `increasePrices` recuperará la lista de productos y, después de haber incrementado los precios, almacenará los productos de nuevo en la base de datos usando el método `saveProduct` definido en el DAO.

'springapp/src/main/java/com/companyname/springapp/service/SimpleProductManager.java':

```
package com.companyname.springapp.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import com.companyname.springapp.domain.Product;
import com.companyname.springapp.repository.ProductDao;

@Component
public class SimpleProductManager implements ProductManager {

    private static final long serialVersionUID = 1L;

    @Autowired
    private ProductDao productDao;

    public void setProductDao(ProductDao productDao) {
        this.productDao = productDao;
    }

    public List<Product> getProducts() {
        return productDao.getProductList();
    }

    public void increasePrice(int percentage) {
        List<Product> products = productDao.getProductList();
        if (products != null) {
            for (Product product : products) {
                double newPrice = product.getPrice().doubleValue() *
                                  (100 + percentage)/100;
                product.setPrice(newPrice);
                productDao.saveProduct(product);
            }
        }
    }
}
```

6.2. Resolver los tests fallidos

Hemos modificado `SimpleProductManager` y ahora, evidentemente, los tests fallan. Necesitamos proporcionar a `ProductManager` una implementación en memoria de `ProductDao`. Realmente no queremos usar el verdadero DAO puesto que queremos evitar tener acceso a la base de datos en nuestros tests unitarios. Añadiremos una clase llamada `InMemoryProductDao` que almacenará una lista de productos que serán definidos en el constructor. Esta clase en memoria tiene que ser pasada a `SimpleProductManager` en el momento de ejecutar los tests.

'springapp/src/test/java/com/companyname/springapp/repository/InMemoryProductDao.java':

```
package com.companyname.springapp.repository;

import java.util.List;

import com.companyname.springapp.domain.Product;

public class InMemoryProductDao implements ProductDao {
```

```
    private List<Product> productList;

    public InMemoryProductDao(List<Product> productList) {
        this.productList = productList;
    }

    public List<Product> getProductList() {
        return productList;
    }

    public void saveProduct(Product prod) {
    }

}
```

Y aquí esta la versión modificada de `SimpleProductManagerTests`:

'springapp/src/test/java/com/companyname/springapp/service/SimpleProductManagerTests.java':

```
package com.companyname.springapp.service;

import java.util.ArrayList;
import java.util.List;

import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

import com.companyname.springapp.domain.Product;
import com.companyname.springapp.repository.InMemoryProductDao;
import com.companyname.springapp.repository.ProductDao;

public class SimpleProductManagerTests {

    private SimpleProductManager productManager;

    private List<Product> products;

    private static int PRODUCT_COUNT = 2;

    private static Double CHAIR_PRICE = new Double(20.50);
    private static String CHAIR_DESCRIPTION = "Chair";

    private static String TABLE_DESCRIPTION = "Table";
    private static Double TABLE_PRICE = new Double(150.10);

    private static int POSITIVE_PRICE_INCREASE = 10;

    @Before
    public void setUp() throws Exception {
        productManager = new SimpleProductManager();
        products = new ArrayList<Product>();

        // stub up a list of products
        Product product = new Product();
        product.setDescription("Chair");
        product.setPrice(CHAIR_PRICE);
        products.add(product);

        product = new Product();
        product.setDescription("Table");
        product.setPrice(TABLE_PRICE);
        products.add(product);

        ProductDao productDao = new InMemoryProductDao(products);
        productManager.setProductDao(productDao);
        //productManager.setProducts(products);
    }

    @Test
    public void testGetProductsWithNoProducts() {
        productManager = new SimpleProductManager();
        productManager.setProductDao(new InMemoryProductDao(null));
        assertNull(productManager.getProducts());
    }

    @Test
    public void testGetProducts() {
        List<Product> products = productManager.getProducts();
        assertNotNull(products);
        assertEquals(PRODUCT_COUNT, productManager.getProducts().size());

        Product product = products.get(0);
        assertEquals(CHAIR_DESCRIPTION, product.getDescription());
```

```
        assertEquals(CHAIR_PRICE, product.getPrice());

        product = products.get(1);
        assertEquals(TABLE_DESCRIPTION, product.getDescription());
        assertEquals(TABLE_PRICE, product.getPrice());
    }

    @Test
    public void testIncreasePriceWithNullListOfProducts() {
        try {
            productManager = new SimpleProductManager();
            productManager.setProductDao(new InMemoryProductDao(null));
            productManager.increasePrice(POSITIVE_PRICE_INCREASE);
        }
        catch(NullPointerException ex) {
         fail("Products list is null.");
        }
    }

    @Test
    public void testIncreasePriceWithEmptyListOfProducts() {
        try {
            productManager = new SimpleProductManager();
            productManager.setProductDao(new InMemoryProductDao(new ArrayList<Product>()));
            //productManager.setProducts(new ArrayList<Product>());
            productManager.increasePrice(POSITIVE_PRICE_INCREASE);
        }
        catch(Exception ex) {
         fail("Products list is empty.");
        }
    }

    @Test
    public void testIncreasePriceWithPositivePercentage() {
        productManager.increasePrice(POSITIVE_PRICE_INCREASE);
        double expectedChairPriceWithIncrease = 22.55;
        double expectedTablePriceWithIncrease = 165.11;

        List<Product> products = productManager.getProducts();
        Product product = products.get(0);
        assertEquals(expectedChairPriceWithIncrease, product.getPrice(), 0);

        product = products.get(1);
        assertEquals(expectedTablePriceWithIncrease, product.getPrice(), 0);
    }
}
```

También necesitamos modificar `InventoryControllerTests` puesto que esta clase también usa `SimpleProductManager`. Aquí está la version modificada de `InventoryControllerTests`:

'springapp/src/test/java/com/companyname/springapp/web/InventoryControllerTests.java':

```
package com.companyname.springapp.web;

import java.util.ArrayList;
import java.util.Map;

import static org.junit.Assert.*;

import org.junit.Test;
import org.springframework.web.servlet.ModelAndView;

import com.companyname.springapp.domain.Product;
import com.companyname.springapp.repository.InMemoryProductDao;
import com.companyname.springapp.service.SimpleProductManager;


public class InventoryControllerTests {

    @Test
    public void testHandleRequestView() throws Exception{
        InventoryController controller = new InventoryController();
        SimpleProductManager spm = new SimpleProductManager();
        spm.setProductDao(new InMemoryProductDao(new ArrayList<Product>()));
        controller.setProductManager(spm);
        //controller.setProductManager(new SimpleProductManager());
        ModelAndView modelAndView = controller.handleRequest(null, null);
        assertEquals("hello", modelAndView.getViewName());
        assertNotNull(modelAndView.getModel());
        Map modelMap = (Map) modelAndView.getModel().get("model");
        String nowValue = (String) modelMap.get("now");
        assertNotNull(nowValue);
    }
}
```

Una vez realizados los cambios anteriores, comprobad que los tests `SimpleProductManagerTests` y `InventoryControllerTests` se ejecutan satisfactoriamente.

6.3. Crear un nuevo contexto de aplicación para configurar la capa de servicio

Hemos visto antes que es tremendamente fácil modificar la capa de servicio para usar persistencia en base de datos. Esto es así porque está despegada de la capa web. Ahora es el momento de despegar también la configuración de la capa de servicio de la capa web. Eliminaremos la configuración de `productManager` y la lista de productos del archivo de configuración '**app-config.xml**'. Así es como este archivo quedaría ahora:

'**springapp/src/main/webapp/WEB-INF/spring/app-config.xml**':

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:mvc="http://www.springframework.org/schema/mvc"
 xmlns:context="http://www.springframework.org/schema/context"
 xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd
  http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
  http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">

    <bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">
        <property name="basename" value="messages"/>
    </bean>

    <!-- Scans the classpath of this application for @Components to deploy as beans -->
    <context:component-scan base-package="com.companyname.springapp.web" />

    <!-- Configures the @Controller programming model -->
    <mvc:annotation-driven/>

    <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="viewClass" value="org.springframework.web.servlet.view.JstlView"></property>
        <property name="prefix" value="/WEB-INF/views/"></property>
        <property name="suffix" value=".jsp"></property>
    </bean>

</beans>
```

Todavía necesitamos configurar la capa de servicio y lo haremos en nuestro propio archivo de contexto de aplicación. Este archivo se llama '**applicationContext.xml**' y será cargado mediante un servlet listener que definiremos en '**web.xml**'. Todos los bean configurados en este nuevo contexto de aplicación estarán disponibles desde cualquier contexto del servlet.

'**springapp/src/main/webapp/WEB-INF/web.xml**':

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring/applicationContext.xml</param-value>
  </context-param>

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

  <display-name>Springapp</display-name>

  <servlet>
    <servlet-name>springapp</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
 <init-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
   /WEB-INF/spring/app-config.xml
  </param-value>
 </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>springapp</servlet-name>
    <url-pattern>*.htm</url-pattern>
  </servlet-mapping>

</web-app>
```

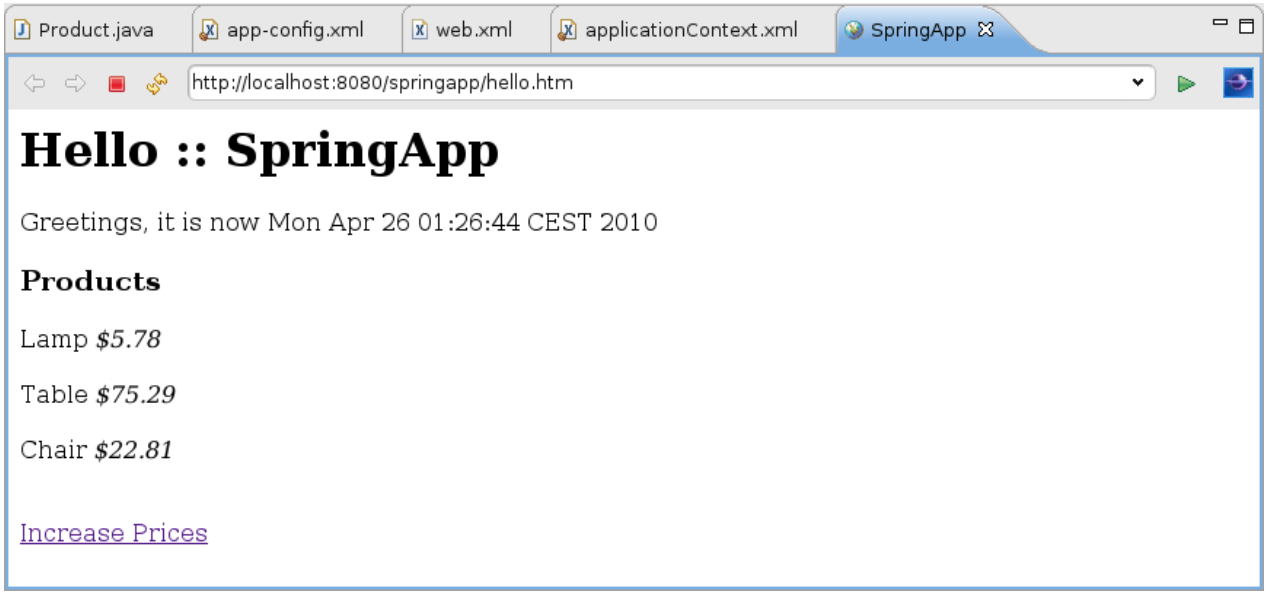
Ahora creamos un nuevo archivo '**applicationContext.xml**' en el directorio '**/WEB-INF/spring**'".

'**springapp/src/main/webapp/WEB-INF/spring/applicationContext.xml**':

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:context="http://www.springframework.org/schema/context"
 xmlns:p="http://www.springframework.org/schema/p"
 xmlns:tx="http://www.springframework.org/schema/tx"
 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
  http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd
  http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">


    <!-- holding properties for database connectivity /-->
    <context:property-placeholder location="classpath:jdbc.properties"/>

    <!-- enabling annotation driven configuration /-->
    <context:annotation-config/>

    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
      <property name="driverClassName" value="${jdbc.driverClassName}"/>
      <property name="url" value="${jdbc.url}"/>
      <property name="username"  value="${jdbc.username}"/>
      <property name="password" value="${jdbc.password}"/>
    </bean>

    <bean id="entityManagerFactory"
 class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"
 p:dataSource-ref="dataSource"
 p:jpaVendorAdapter-ref="jpaAdapter">
 <property name="loadTimeWeaver">
            <bean class="org.springframework.instrument.classloading.InstrumentationLoadTimeWeaver"/>
 </property>
 <property name="persistenceUnitName" value="springappPU"></property>
    </bean>

    <bean id="jpaAdapter"
 class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter"
 p:database="${jpa.database}"
 p:showSql="${jpa.showSql}"/>

    <bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager"
 p:entityManagerFactory-ref="entityManagerFactory"/>

    <tx:annotation-driven transaction-manager="transactionManager"/>

    <!-- Scans the classpath of this application for @Components to deploy as beans -->
    <context:component-scan base-package="com.companyname.springapp.repository" />
    <context:component-scan base-package="com.companyname.springapp.service" />

</beans>
```

6.4. Test final de la aplicación completa

Ahora es el momento de ver si todas estas piezas funcionan juntas. Construye y despliega la aplicación finalizada y recuerda tener la base de datos arrancada y funcionando. Esto es lo que deberías ver cuando apuntes tu navegador web a la aplicación:



La aplicación completa

La aplicación aparece exactamente como lo hacía antes. Sin embargo, hemos añadido la persistencia en base de datos, por lo que si cierras la aplicación tus incrementos de precio no se perderán sino que estarán todavía allí cuando vuelvas a cargar la aplicación.

6.5. Resumen

Hemos completado las tres capas de la aplicación -- la capa web, la capa de servicio y la capa de persistencia. En esta última parte hemos reconfigurado la aplicación.

- Primero hemos modificado la capa de servicio para usar la interface ProductDAO.
- Después hemos tenido que arreglar algunos fallos en los tests de la capa de servicio y la capa web.
- A continuación, hemos introducido un nuevo applicationContext para separar la configuración de la capa de servicio y de la capa de persistencia de la configuración de la capa web.
- Finalmente hemos desplegado la aplicación y testeado que aún funciona.

A continuación puedes ver una captura de pantalla que muestra el aspecto que debería tener la estructura de directorios del proyecto después de seguir todas las instrucciones anteriores.