Capítulo 7
Pruebas y Spring

1. Introducción

La preocupación centrada en las pruebas es fundamental para los desarrollos actuales. Cada vez se utiliza más el TDD (*Test-Driven Development* o desarrollo basado en pruebas), que consiste en que, al mismo tiempo que se escriben nuestros programas, se escriben también las pruebas unitarias antes de escribir el código fuente del software. Desde hace algún tiempo, también se han utilizado BDD (*Behavior Driven Development*), que son una evolución del TDD con el que las pruebas se describen a través de frases, por ejemplo, en castellano o inglés, con una sintaxis particular, que posteriormente se procesan con un framework como Cucumber. También existen los ATDD (*Acceptance Test-Driven Development*), para los que los criterios de aceptación se transcriben en las pruebas.

Spring ofrece diferentes API que simplifican la implementación de pruebas unitarias (TU) y de integración (TI). Es necesario verificar y probar todo lo que vale la pena probar. Si tiene problemas al escribir los TU o TI, es porque tiene que volver a trabajar (refactorizar) su código. Una buena práctica consiste en escribir la prueba al mismo tiempo que la clase que se está probando, y algunas veces incluso codificar las pruebas antes de implementar los métodos que se están probando, para que la arquitectura de la aplicación sea compatible con las pruebas. Las pruebas unitarias son pruebas rápidas, que permiten validar acciones de bajo nivel. Las pruebas de integración utilizan juegos de pruebas para validar código de nivel superior.

También hay pruebas que simulan un usuario. Estas pruebas se realizan con herramientas como Selenium.

Spring nos ayuda proporcionando un conjunto de API especializadas para las pruebas. Encontraremos API para simular los contextos de ejecución de nuestras aplicaciones. También dispondremos de API para modificar la configuración de objetos de Spring en memoria. Para probar con datos, Spring nos ayudará dándonos acceso, de una manera muy sencilla, a conjuntos de pruebas que utilizan una base de datos en memoria. Además, tenemos acceso a un contexto Spring de prueba, que puede sobrecargar el contexto de la aplicación que se está probando.

Configuraremos las pruebas con archivos XML y anotaciones, sin perder de vista que estas pruebas deberán tener un alto grado de mantenibilidad.

De manera ideal, debemos definir desde el principio el nivel de cobertura de las pruebas TU y TI dentro de nuestro proyecto y configurarlas lo antes posible porque son muy estructurales para el código. Algunas veces estas pruebas son complejas de implementar y es necesario reservar tiempo para hacerlo. El nivel de cobertura se puede probar con JaCoCo y es habitual que se comparta en el equipo usando Sonar.

2. Los mock objects

Cuando probamos una clase, queremos centrar nuestras pruebas en esa clase y tenemos que encontrar un sistema para no probar el resto de las clases que interactúan con la que se está probando. Podemos usar objetos simulados llamados *mocks* para no tener que invocar los objetos reales, lo que haría necesario un contexto de ejecución demasiado grande.

Spring ofrece un conjunto muy completo de mocks. Son más fáciles de usar que los mocks de EasyMock y MockObjects. A menudo, se utilizan con el framework Mockito (http://site.mockito.org/).

Tipo de mock	Uso
Entorno	Clases relacionadas con el entorno de ejecución.
JNDI	Simula recursos JNDI, como un origen de datos.

Tipo de mock	Uso
API de los Servlets	Simula un servlet, útil con Spring MVC.
API de los Portlets	Simula portlets Spring MVC (desaparece con Spring 5+).
Soporte	Herramientas para ayudar con la introspección de los objetos.

2.1 Mocks especializados por «entorno»

Se simulan clases de entorno.

Clase	Mock
Entorno	MockEnvironment
@PropertySource	MockPropertySource

Estos mocks permiten simular un entorno y un `PropertySource`.

2.2 Soporte

2.2.1 Utilidades generales

La clase `ReflectionTestUtils` del paquete `org.springframework.test.util` proporciona ayuda para la introspección y manipulación de objetos. Todos los miembros de la clase se vuelven accesibles, incluso los miembros «private».

Por ejemplo, para una clase `Vehiculo`:

```
public class Vehiculo {
[accesorios]
   private long id;
   private String modelo;
}
```

Podemos hacer lo que queramos:

```
final Vehiculo persona = new Vehiculo();
ReflectionTestUtils.setField(persona, "id", new Long(99),
long.class);
assertEquals("id", 99L, persona.getId());
ReflectionTestUtils.setField(persona, "modelo", null, String.class);
assertNull("modelo", persona.getModelo());
try {
   ReflectionTestUtils.setField(persona, "id", null, long.class);
   fail("Debería generar una excepción");
} catch (IllegalArgumentException aExp) {
   assert (aExp.getMessage()
               .contains("IllegalArgumentException"));
}
ReflectionTestUtils.invokeSetterMethod(persona, "id", new
Long(99), long.class);
assertEquals("id", 99L, persona.getId());

ReflectionTestUtils.invokeSetterMethod(persona, "setId", new
Long(1), long.class);
assertEquals("id", 1L, persona.getId());
try {
   ReflectionTestUtils.invokeSetterMethod(persona, "id", null,
long.class);
   fail("Debería generar una excepción");
} catch (IllegalArgumentException aExp) {
   assert (aExp.getMessage()
                .contains("IllegalArgumentException"));
   }
}
```

Por lo tanto, es posible intervenir sobre variables o métodos que normalmente no están disponibles. No deberíamos necesitar estos desbloqueadores porque los métodos privados se prueban utilizando pruebas de métodos públicos.

2.2.2 Spring MVC

La clase `ModelAndViewAssert` del paquete `org.springframework.test.web` proporciona ayuda para probar objetos de tipo Spring MVC `ModelAndView`. Para probar un controlador Spring MVC, se utiliza `ModelAndViewAssert` en combinacióncon `MockHttpServletRequest`, `MockHttpSession`, etc. El paquete `org.springframework.mock.web` se basa en la API Servlet 3 desde Spring 4.0. En este capítulo vamos a ver muchos ejemplos de cómo usar estos mocks.

2.3 Pruebas de integración

2.3.1 Visión general

A menudo es necesario probar los comportamientos de un conjunto de objetos para verificar, por ejemplo, la interacción entre las capas de la base de datos o las reglas de gestión que presentan conjuntos de datos. Idealmente, estas pruebas se deben realizar en un subconjunto técnico del entorno global de ejecución. Este tipo de pruebas se agrupa en las pruebas de integración. Spring permite hacer estas pruebas sin iniciar el servidor de aplicaciones o la aplicación completa. Este es uno de sus puntos fuertes.

La idea de las pruebas de integración es proporcionar los elementos para probar las diferentes capas de software, proporcionando un entorno de ejecución independiente. La librería básica para las ayudas de codificación de las pruebas se encuentra en el módulo spring-test del paquete `org.springframework.test`.

Estas pruebas son más lentas que las unitarias, pero más rápidas que las de Selenium (las pruebas de Selenium simulan una sesión de un usuario imitando su comportamiento en la aplicación).

Las TI se identifican utilizando anotaciones específicas para administrar, entre otras cosas, la caché del contexto para el IoC, la gestión de transacciones y la gestión de los juegos de pruebas de la base de datos.

2.3.2 Almacenamiento en caché del contexto de prueba

Para el caso de un conjunto de pruebas, la carga del contexto sería relativamente costosa si se volviera a cargar para cada prueba. Spring permite cargar un contexto y usarlo en una batería de pruebas. En caso de que una prueba altere el contexto, es posible pedirle a Spring que vuelva a cargar el contexto inicial para tener siempre un entorno limpio y estable. Incluso hay API para gestionar la degradación del contexto Spring.

Cuando se carga una batería de pruebas, especificaremos la lista de archivos de configuración que se van a cargar. En general, utilizaremos un archivo específico que complementará al archivo de configuración principal de la aplicación. A continuación, dispondremos del contexto de la aplicación principal sobrecargado para tener en cuenta todos los elementos diferentes entre el entorno de usuario y el de test.

Solo buscamos emular la base para la parte back y el servidor web para la parte front. Para programas que explotan archivos, algunas veces también utilizaremos extractos de estos archivos para probar solo los casos funcionales que pasan o no pasan.

2.3.3 Pruebas back y front

La parte back transforma los datos físicos de las bases de datos, archivos o flujos, en datos que puede utilizar una parte front que procesa o muestra los datos.

Pruebas de las partes back

Para una prueba de integración back, debe emular las fuentes de información física que provienen de bases de datos, archivos o flujos.

Bases de datos

Para las bases de datos, podemos usar clases Spring o un framework de terceros, como DbUnit o Liquibase.

Observación

Solo mostraremos los ejemplos en SQL integrados en el framework de Spring, pero le invitamos a profundizar en este asunto estudiando las posibilidades de otros frameworks.

Librerías de pruebas de Spring

Para proyectos sencillos que no usan una librería ORM, como Hibernate o JPA, y para las que el modelo de datos es pequeño, podemos usar el paquete `org.springframework.test.jdbc`, que contiene la clase `JdbcTestUtils`, la cual implementa métodos estáticos de utilidad:

`countRowsInTable(..)`	Cuenta el número de filas de una tabla.
`countRowsInTableWhere(..)`	Cuenta el número de filas de una tabla con una cláusula WHERE.
`deleteFromTables(..)`	Vacía una tabla.
`deleteFromTableWhere(..)`	Elimina filas de una tabla con una cláusula WHERE.
`dropTables(..)`	Elimina las tablas especificadas.

2.4 Anotaciones

2.4.1 @ContextConfiguration

La anotación principal es `@ContextConfiguration`. Permite cargar el contexto de pruebas de diferentes maneras. Especificamos la ubicación de los archivos de configuración, así como las clases anotadas `@Configuration` que llevan la configuración:

```
@ContextConfiguration("/test-config.xml")
public class XmlApplicationContextTests {
    // Contenido de la clase...
@ContextConfiguration(clases = TestConfig.class)
public class ConfigClassApplicationContextTests {
    // Contenido de la clase...
```