Chapitre 7
Les tests et Spring

1. Introduction

Les préoccupations centrées sur les tests sont fondamentales dans les développements d'aujourd'hui. Le TDD (*Test-Driven Development* ou en français développement piloté par les tests), qui consiste, lors de l'écriture de nos programmes, à écrire les tests unitaires avant d'écrire le code source d'un logiciel, est de plus en plus utilisé. Depuis quelques temps nous utilisons aussi des BDD (*Behavior Driven Development*) qui sont une évolution du TDD avec lequel on décrit les tests via des phrases qui sont ensuite utilisées avec un framework comme Cucumber.

Spring propose des API qui nous simplifient la mise en œuvre des tests unitaires (TU) et des tests d'intégration (TI). Il faut tester tout ce qui vaut le coup d'être testé. Si vous rencontrez des problèmes lors de la rédaction des TU et TI, c'est qu'il faut retravailler (refactoriser) votre code. Une bonne pratique consiste à écrire le test en même temps que la classe testée et même parfois à coder les tests avant l'implémentation des méthodes testées de façon à ce que l'architecture de l'application soit compatible avec les tests. Les tests unitaires sont des tests rapides qui permettent de valider des actions de bas niveau. Les tests d'intégration utilisent des jeux d'essais pour valider du code de plus haut niveau. Il existe aussi des tests qui simulent un utilisateur. Ces tests sont faits avec des outils comme Selenium.

Spring nous aide en mettant à disposition un ensemble d'API spécialisées pour les tests. Nous trouverons des API pour simuler les contextes d'exécution de nos applications. Nous aurons aussi des API pour modifier la configuration des objets Spring en mémoire. Pour tester avec des données, Spring nous aidera en nous donnant accès, de façon très simple, à des jeux d'essais utilisant une base de données en mémoire. En outre, nous avons accès à un contexte Spring de test qui peut surcharger le contexte de l'application testée.

Nous configurerons les tests avec les fichiers XML et les annotations en gardant à l'esprit que ces tests devront avoir un haut degré de maintenabilité.

Dans l'idéal, il faut, dès le départ, définir le niveau de couverture des tests TU et TI pour notre projet et les mettre en place le plus tôt possible car ils sont très structurants pour le code. Ils sont parfois complexes à mettre en œuvre et il faut provisionner du temps pour cela.

2. Les mock objects

Nous pouvons utiliser des simulacres d'objets nommés mocks pour ne pas avoir à invoquer des objets réels qui demanderaient un contexte d'exécution trop étendu.

Spring fournit un ensemble de mocks très complet. Ils sont plus simples à utiliser que les mocks EasyMock et MockObjects. On les utilise souvent avec le framework Mockito (http://site.mockito.org/).

Type de mock	Utilisation
Environnement	Classes liées à l'environnement d'exécution.
JNDI	Simule des ressources JNDI comme une datasource.
API des Servlets	Simule une servlet, utile avec Spring MVC.
API des Portlets	Simule les portlets Spring MVC (disparaît avec Spring 5+).
Support	Outils aidant pour l'introspection des objets.

2.1 Mocks spécialisés pour "environnement"

On simule les classes d'environnements.

Classe	Mock
Environnement	MockEnvironment
@PropertySource	MockPropertySource

Ces mocks permettent de simuler un environnement et un `PropertySource`.

2.2 Support

2.2.1 Utilités générales

La classe `ReflectionTestUtils` du package `org.springframework.test.util` apporte des aides pour l'introspection et la manipulation des objets. Tous les membres des classes deviennent accessibles, même les membres « private ».

Par exemple, pour une classe `Vehicule` :

```
public class Vehicule {
[accesseurs]
   private long id;
   private String modele;
}
```

Nous pouvons faire ce que nous voulons :

```
final Vehicule person = new Vehicule();
ReflectionTestUtils.setField(person, "id", new Long(99), long.class);
assertEquals("id", 99L, person.getId());
ReflectionTestUtils.setField(person, "modele", null, String.class);
assertNull("modele", person.getModele());
try {
   ReflectionTestUtils.setField(person, "id", null, long.class);
   fail("Devrait lever une exception !");
} catch (IllegalArgumentException aExp) {
   assert (aExp.getMessage()
               .contains("IllegalArgumentException"));
```

```
}
ReflectionTestUtils.invokeSetterMethod(person, "id", new
Long(99), long.class);
assertEquals("id", 99L, person.getId());

ReflectionTestUtils.invokeSetterMethod(person, "setId", new
Long(1), long.class);
assertEquals("id", 1L, person.getId());
try {
   ReflectionTestUtils.invokeSetterMethod(person, "id", null,
long.class);
   fail("Devrait lever une exception !");
} catch (IllegalArgumentException aExp) {
   assert (aExp.getMessage()
               .contains("IllegalArgumentException"));
   }
}
```

Il est ainsi possible d'intervenir sur des variables ou des méthodes qui ne sont normalement pas disponibles.

2.2.2 Spring MVC

La classe `ModelAndViewAssert` du package `org.springframework.test.web` apporte des aides pour tester les objets de type Spring MVC `ModelAndView`. Pour tester un contrôleur Spring MVC, on utilise `ModelAndViewAssert` combiné avec `MockHttpServletRequest`, `MockHttpSession` et ainsi de suite. Le package `org.springframework.mock.web` est basé sur l'API Servlet 3 depuis Spring 4.0. Nous verrons dans le chapitre sur Les tests et Spring de nombreux exemples d'utilisation de ces mocks.

2.3 Tests d'intégration

2.3.1 Vue d'ensemble

Il faut souvent tester les comportements d'un ensemble d'objets pour vérifier, par exemple, l'interaction entre les couches du socle ou les règles de gestion mettant en scène des jeux de données. L'idéal est de faire ces tests dans un sous-ensemble technique de l'environnement global d'exécution. Ce type de tests est regroupé dans les tests d'intégration. Spring permet de faire ces tests sans lancer le serveur d'application ou l'application au complet. C'est un de ses points forts.

L'idée des tests d'intégration est de fournir les éléments pour tester les différentes couches logicielles en fournissant un environnement d'exécution autonome. La bibliothèque de base pour les aides au codage des tests est dans le module `spring-test` qui est dans le package `org.springframework.test`.

Ces tests sont plus lents que les tests unitaires, mais plus rapides que les tests Selenium (les tests Selenium simulent une session d'un utilisateur en imitant son comportement sur l'application).

Les TI sont identifiés via l'aide d'annotations spécifiques afin de gérer, entre autre, le cache du contexte pour l'IoC, la gestion des transactions, la gestion des jeux d'essais en base.

2.3.2 Mise en cache du contexte de test

Le chargement du contexte serait relativement long dans le cas d'une suite de tests si on le rechargeait pour chaque test. Spring permet de charger un contexte et de l'utiliser une batterie de tests. Dans le cas pour lequel un test altérerait le contexte, il est possible de demander à Spring de recharger le contexte initial afin d'avoir toujours un environnement propre et stable. Il existe même des API pour gérer la dégradation du contexte Spring.

Nous spécifierons, au chargement d'une batterie de tests, la liste des fichiers de configuration à charger. En général, nous utiliserons un fichier spécifique qui complétera le fichier de configuration principal de l'application. Nous disposerons alors du contexte de l'application principale surchargé pour prendre en compte tous les éléments qui divergent entre l'environnement d'utilisation et l'environnement de test.

Nous chercherons juste à émuler la base pour la partie back et le serveur web pour la partie front. Nous utiliserons aussi parfois, pour des programmes qui exploitent des fichiers, des extraits de ces fichiers afin de ne tester que les cas fonctionnels passants et non passants.

2.3.3 Tests back et front

La partie back transforme des données physiques provenant de base de données, ou des fichiers ou des flux, en données exploitables par une partie front qui traite ou affiche les données.

Les tests des parties back

Pour un test d'intégration back, il faut émuler les sources d'informations physiques provenant de bases de données, de fichiers ou de flux.

Bases de données

Pour les bases de données, nous pouvons utiliser les classes Spring ou un framework tiers comme DBUnit ou Liquibase.

Remarque

Nous ne montrerons que les exemples en SQL intégrés au framework de Spring mais nous vous invitons à approfondir le sujet en étudiant les possibilités des autres frameworks.

Librairies de tests Spring

Pour les projets simples, n'utilisant pas de librairie d'ORM comme Hibernate et JPA et pour lequel le modèle de données est petit, nous pouvons utiliser le package `org.springframework.test.jdbc` qui contient la classe `JdbcTestUtils`, qui implémente des méthodes statiques utilitaires :

`countRowsInTable(..)`	Compte le nombre de lignes dans une table.
`countRowsInTableWhere(..)`	Compte le nombre de lignes d'une table avec une clause WHERE.

`deleteFromTables(..)`	Vide une table.
`deleteFromTableWhere(..)`	Supprime les lignes d'une table avec une clause WHERE.
`dropTables(..)`	Supprime les tables spécifiées.

2.4 Annotations

2.4.1 @ContextConfiguration

L'annotation principale est `@ContextConfiguration`. Elle permet de charger le contexte de tests de multiples façons. Nous spécifions l'emplacement des fichiers de configuration ainsi que les classes annotées `@Configuration` qui portent la configuration :

```
@ContextConfiguration("/test-config.xml")
public class XmlApplicationContextTests {
    // Contenu de la classe...
@ContextConfiguration(classes = TestConfig.class)
public class ConfigClassApplicationContextTests {
    // Contenu de la classe...
```

Il est également possible de spécifier la classe d'initialisation `ApplicationContextInitializer` :

```
@ContextConfiguration(initializers = CustomContextIntializer.class)
public class ContextInitializerTests {
// Contenu de la classe...
public interface ApplicationContextInitializer<C extends
ConfigurableApplicationContext>
```

Cette interface est utilisée en callback du chargement du contexte afin de charger les propriétés dans le contexte, notamment pour prendre en compte les paramètres `context-param` et `init-param` pour les applications web.