Com definir tests en context Spring

A qui va dirigit

Aquest how-to va dirigit a tots aquells desenvolupadors encarregats de la creació de la capa de negoci d'una aplicació Canigó.

En aquest document es presenta el procediment per definir tests de funcionalitat de negoci amb un context Spring complet inicialitzat.

Versió de Canigó

Els passos descrits en aquest document poden ser d'aplicació a les **versions 2.x de Canigó**.

Requeriments

Podeu aplicar aquest procediment tant bon punt disposeu de les interfícies de negoci de la vostra aplicació, i tingueu una implementació de les mateixes configurada correctament en Spring. No és necessari que existeixi la capa de presentació.

Context

La definició de tests a Canigó es fa amb el framework de test JUnit versió 3.8.1. En principi és possible fer tests unitaris, sense context Spring. En aquest tipus de tests, la instanciació i inicialització amb valors dels objectes a testejar ha de fer-se a mà. L'àmbit apropiat dels tests unitaris és per classes aïllades, a nivell mètode i amb preparació de paràmetres ad-hoc.

Amb aquest howTo es pretén possibilitar la definició de test d'integració, no unitaris.

Objectius

L'objectiu fonamental és facilitar la definició de tests d'integració (no unitaris) a nivell de capa de negoci, carregant la configuració Spring de l'aplicació tal i com es faria al servidor i per tant amb accés als "beans" configurats.

Procediment

1. **Modificacions al vostre projecte.**

    a. **Modificar el fitxer "pom.xml" de definició Maven del projecte.**

    Cal que afegiu una nova dependència de "scope" tipus "test" a la secció "dependencies":

```
<dependency>
        <groupId>canigo</groupId>
        <artifactId>canigo-test-base</artifactId>
        <version>1.0.0</version>
        <scope>test</scope>
</dependency>
```

    La dependència "canigo-test-base" està publicada al repositori Maven del CTTI

b. **Configureu l'accés a dades per les proves.**

En l'execució en servidor la connexió a la base de dades ve proveïda per un pool configurat al mateix servidor.

En el moment d'executar els tests no es disposarà d'aquest pool i per tant cal configurar un accés de dades específic per test. Per fer aquesta configuració n'hi ha prou amb crear un fitxer "testProperties.properties" com el següent:

```
test.hibernate.dialect=org.hibernate.dialect.Oracle9Dialect
test.hibernate.show_sql=true
test.hibernate.connection.driver_class=oracle.jdbc.driver.OracleDriver
test.hibernate.connection.url=jdbc\:oracle\:thin\://@localhost\:1521\:XE
test.hibernate.hbm2ddl.auto=update
test.hibernate.connection.username=canigo
test.hibernate.connection.password=canigo
test.hibernate.mapping.locations=classpath\:/hibernate/mappings/*.hbm.xml
test.struts.config.location=struts/struts-config.xml
```

- Exemple de propietats de configuració per Oracle local -

Aquest fitxer ha de trobar-se en el classpath de test; normalment a "src/test/resources".

La propietat `test.hibernate.hbm2ddl.auto` permet indicar a hibernate la política de creació de taules de la base de dades de test. En concret amb el valor *update* s'assegura que les taules es corresponen amb els *mappings*, modificant-les si cal.

A continuació trobareu la configuració anterior modificada per funcionar amb una base de dades de proves HSQLDB en memòria:

```
test.hibernate.dialect=org.hibernate.dialect.HSQLDialect
test.hibernate.show_sql=true
test.hibernate.connection.driver_class=org.hsqldb.jdbcDriver
test.hibernate.connection.url=jdbc:hsqldb:mem:sample
test.hibernate.hbm2ddl.auto=create
test.hibernate.connection.username=sa
test.hibernate.connection.password=
test.hibernate.mapping.locations=classpath\:/hibernate/mappings/*.hbm.xml
test.struts.config.location=struts/struts-config.xml
```

- Exemple de propietats de configuració per HSQLDB en memòria -

Aquesta configuració té l'avantatge que no cal tenir una base de dades instal·lada ja que treballa sobre una versió en memòria. En contra, pot no reproduir exactament el comportament de la base de dades final.

Les configuracions presentades són exemples, i la configuració idònia per la vostra aplicació ha de ser determinada per l'arquitecte de l'aplicació.

c. (Opcional) Modificar la configuració Spring interna del test.

El mecanisme de test presentat en aquest how-to requereix d'una configuració d'Spring específica (canigo-test-application-context.xml). Aquest fitxer es troba dins del propi jar (*canigo-test-base-1.0.0.jar*) i normalment no cal modificar-la; si heu de fer-ho n'hi ha prou amb crear un nou fitxer amb aquest nom en el classpath(no cal modificar el jar). Com a punt de partida millor que copieu el contingut del fitxer original.

2. Creació de les classes de test

Segons les necessitats de l'aplicació es poden fer tantes classes de test com es vulgui.

La ubicació habitual de les classes de test es sota "src/test/java". Us caldrà triar un nom de package apropiat, que normalment es correspondrà amb el nom del package de les classes que es sotmeten a test.

Aquestes classes han d'extendre *net.gencat.ctti.canigo.test.CanigoTest*.

CanigoTest és una extensió de la classe TestCase de JUnit. Això implica que es consideren mètodes de test aquells mètodes públic que comencen pel prefix "test". A l'exemple:

```
public void testExemple() throws Exception {
```

Per altra banda, cal tenir present que a JUnit existeix un mètode convencional de preparació del test anomenat "setup", que a la classe CanigoTest realitza la càrrega inicial de Spring en mode de test (amb la configuració especificada al fitxer de properties "testProperties.properties"). Si es fa un override de setup per necessitats del test cal que es cridi a super.setup().

```
package net.gencat.ctti.canigo.aplicacio1.bo;
import net.gencat.ctti.canigo.test.CanigoTest;
import org.springframework.context.ApplicationContext;
public class ExempleTest extends CanigoTest {

        //Aquest es un métode de test JUnit.
        public void testExemple() throws Exception {

        }
//Aquest es un métode de test JUnit.
        public void testExemple2() throws Exception {

        }
}
```

- Esquelet d'una classe de test -

En aquest punt és recomanable fer una primera execució de tests. Aquesta execució permet verificar que la càrrega de context es fa sense problemes (veure el següent punt).

Com definir tests en context Spring

3. Execució inicial via Eclipse

Un cop definit un mètode buit de test és el moment d'intentar l'execució del test per comprovar que la configuració de Spring és carrega correctament. A Eclipse es pot executar la nova classe de Test amb l'opció "Run As.../JUnit Test" del menú de context de la classe de test (botó dret del mouse).

Si en aquest punt l'execució del test apareix com a incorrecta, normalment serà degut a una configuració errònia de Spring; a la vista "JUnit" d'Eclipse trobareu l'excepció que pot haver causat l'error. Sovint els errors de configuració de Spring es poden detectar per endavant amb una configuració correcta del plugin de Spring (Spring-Ide)

Alternativament, podeu augmentar el nivell de detall del "logging" modificant la configuració que es troba a "src/main/resources/log4j/log4j.xml".

Si aconseguiu executar el mètode buit de test podeu passar a definir els mètodes de test efectius.

4. Implementació dels mètodes de test. Definició dels passos

En la execució d'una aplicació Canigó en el servidor, la invocació de mètodes de negoci es fa dins d'un context Hibernate amb característiques específiques:

- *Patró OpenSessionInView*: Es disposa d'una mateixa sessió Hibernate per tota l'execució d'un request
- *Transaccional declarativa d'Spring*: En la execució de mètodes de negoci es té en compte la configuració de transaccionalitat

Per tal que els tests reprodueixin correctament el comportament d'aquest entorn definim el concepte de pas de test.

Un pas de test és la unitat d'execució que implementa el patró OpenSessionInView amb suport de transaccionalitat dins d'un mètode de test.

Un mètode de test està format per un o més passos.

Per tal d'implementar un pas caldrà afegir un bloc de codi com el següent:

```
new CanigoPasDeTest(){
        public void executa(ApplicationContext context) {
         //Aqui posem el codi de test
        }
};
```

Com definir tests en context Spring

A continuació es presenta un mètode amb dos passos de test.

```
public void testExemple2() throws Exception {
        //pas1
         new CanigoPasDeTest(){
                public void executa(ApplicationContext context) {
                 //Aqui posem el codi de test del pas 1
                }
        };
        //pas2
         new CanigoPasDeTest(){
                public void executa(ApplicationContext context) {
                 //Aqui posem el codi de test del pas 2
                }
        };

}
```

- Esquelet d'un mètode de test amb dos passos -

5. Implementació del passos de test

Un pas de test serveix per simular la invocació de mètodes de negoci des d'un mètode d'una Action. Cal tenir present que per defecte els mètodes d'una Action no són transaccionals, i que la manera d'invocar més d'un mètode de negoci en una mateixa transacció és crear un nou mètode a l'objecte de Negoci que encapsuli les invocacions de segon nivell. Per tant, normalment un pas de test tindrà una invocació transaccional com a molt, o bé varies invocacions de mètodes de negoci que no van a la mateixa transacció.

El procediment per provar un mètode de negoci en un pas de test típicament implica els següents passos:

- Obtenció del bean de Spring que representa al objecte de negoci a partir del paràmetre "context".

```
CategoryBO categoryBO = (CategoryBO) context.getBean("categoryBO", CategoryBO.class);
```

Atenció: cal obtenir el bean del BO configurat transaccionalment:

```
<bean id="categoryBOTarget"
                                                                    class
="net.gencat.ctti.canigo.samples.prototip.model.bo.impl.CategoryBOImpl">
            <property name="dao" ref="universalHibernateDAO"/>
</bean>
<!-- a continuació el bean amb proxy transaccional -->
        <bean id="categoryBO"  parent="baseDaoProxy">
                <property name="target"><ref bean="categoryBOTarget"/></property>
        </bean>
```

Com definir tests en context Spring

Si heu de fer més d'un pas és millor declarar la variable que referència el BO com a camp de la classe del test:

```
public class ExempleTest extends CanigoTest {
private CategoryBO categoryBO;
....
```

i llavors podeu fer la inicialització en el override del mètode "setup"

```
protected void setUp() throws Exception {
                super.setUp();
                categoryBO = (CategoryBO) context.getBean("categoryBO",
CategoryBO.class);
                assertNotNull(categoryBO);
        }
```

- Preparació de paràmetres pel mètode. Normalment significarà instanciar i donar valors a objectes del model:

```
                public void executa(ApplicationContext context) {
                        Category category = new Category();
                        category.setId("TEST");
                        category.setName("TEST");
```

Es possible que calgui obtenir objectes del model en un pas de test previ. En aquest cas haureu de crear camps del test, per que hi pugui haver visibilitat d'un pas a l'altre.

- Invocació del mètode

```
                public void executa(ApplicationContext context) {
                        Category category = new Category();
                        category.setId("TEST");
                        category.setName("TEST");
                        categoryBO.save(category);
                }
```

Com definir tests en context Spring

- Assercions contra els resultats o les excepcions del mètode.

```
public void executa(ApplicationContext context) {
        Category category = new Category();
        category.setId("TEST");
        Category result = categoryBO.load(category);
        assertNotNull(result);
        assertEquals("TEST", result.getName());
}
```