## Spring jUnit Annotations - Contents:

|  |  |
| --- | --- |
| **Annotation** | **Package Detail/Import statement** |
| [@RunWith](http://www.techferry.com/articles/spring-jUnit-annotations.html#RunWith) | import org.junit.runner.RunWith; |
| [@ContextConfiguration](http://www.techferry.com/articles/spring-jUnit-annotations.html#ContextConfiguration) | import org.springframework.test.context.ContextConfiguration; |
| [@Test](http://www.techferry.com/articles/spring-jUnit-annotations.html#Test) | import org.junit.Test; |
| [@DirtiesContext](http://www.techferry.com/articles/spring-jUnit-annotations.html#DirtiesContext) | import org.springframework.test.annotation.DirtiesContext; |
| [@Timed](http://www.techferry.com/articles/spring-jUnit-annotations.html#Timed) | import org.springframework.test.annotation.Timed; |

We are now ready to test our Spring based application using jUnit and Spring Unit testing framework. The following jUnit and Spring annotations will be used to accomplish this.

### @RunWith

When a class is annotated with @RunWith or extends a class annotated with @RunWith, JUnit will invoke the class it references to run the tests in that class instead of the runner built into JUnit. Let us configure jUnit to use Spring jUnit Class runner.

@RunWith(SpringJUnit4ClassRunner.class)

@ContextConfiguration(locations = { "/spring-servlet-test.xml" })

public class CompanyServiceTest {

...

}

### @ContextConfiguration

Set the spring ApplicationContext for your test classes using @ContextConfiguration annotation.

@RunWith(SpringJUnit4ClassRunner.class)

@ContextConfiguration(locations = { "/spring-servlet-test.xml" })

public class CompanyServiceTest {

...

}

|  |  |
| --- | --- |
| http://www.techferry.com/articles/images/tips.gif | @ContextConfiguration provides support for inheriting resource locations or configuration classes declared by superclasses by default. |

### @Test

Annotate all your jUnit unit tests with @Test. Also note that we can wire other spring beans in our jUnit test classes using @Autowired annotation.

@Autowired

private CompanyService companyService;

@Test

public void testFindByName() {

Company company = companyService.findByName("techferry");

if (company != null) {

assertEquals("prospect", company.getStatus().getName());

}

}

### @DirtiesContext

Annotate @DirtiesContext to indicate that it dirties the ApplicationContext. This will trigger context reloading before execution of next test.

### @Timed

Indicates that the annotated test method must finish execution in a specified time period (in milliseconds). If the text execution time exceeds the specified time period, the test fails.

@Timed(millis=1000)

import junit.framework.Assert;

import org.junit.Test;

import org.junit.runner.RunWith;

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

import org.springframework.test.context.ContextConfiguration;

import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)

@ContextConfiguration("classpath:spring-config.xml")

public class SpringAppTests {

@Autowired

private HelloService helloService;

@Test

public void testSayHello() {

Assert.assertEquals("Hello world!", helloService.sayHello());

}

}

Testing Some Basic Spring Beans

import static org.junit.Assert.\*;

import org.junit.Test;

import org.junit.runner.RunWith;

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

import org.springframework.context.support.FileSystemXmlApplicationContext;

import org.springframework.test.context.ContextConfiguration;

import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import org.springframework.context.ApplicationContext;

import com.codesolid.tutorials.UserStory;

import com.codesolid.tutorials.User;

import static org.junit.Assert.assertEquals;

import static org.springframework.test.web.servlet.setup.MockMvcBuilders.webAppContextSetup;

@RunWith(SpringJUnit4ClassRunner.class)

@ContextConfiguration("file:src/main/resources/spring-config.xml")

public class ContextTests {

@SuppressWarnings("SpringJavaAutowiringInspection")

@Autowired

protected ApplicationContext ac;

// Using ApplicationContext

// The getBean method is the workhorse of the Spring ApplicationContext.

// The user field in the UserStory object was set up (dependency injected)

// by Spring:

@Test

public void testUserNotNull() {

UserStory story = (UserStory) ac.getBean("userStory");

assertNotNull(story.getUser());

}

// Not using ApplicationContext

// Regular non-Spring instantiation

// Instantiating our own UserStory, you can see that the user

// is null.

@Test

public void testUserStoryNotFromContext() {

UserStory story = new UserStory();

assertNull(story.getUser());

}

// Not using ApplicationContext

// Regular non-Spring instantiation

// The default role if you just call the constructor

// is "User".

@Test

public void testNonSpringUserDefaultRole()

{

User u = new User();

assertEquals(u.getRole(), "User");

}

// Using ApplicationContext

// Since Spring instantiated this according to the property

// from spring-config.xml, it overrode the default "user"

// role with "SuperGenius User". (This presumably

// is Wile E. Coyote.)

@Test

public void testUserRoleIsAsExpected() {

UserStory story = (UserStory) ac.getBean("userStory");

assertEquals(story.getUser().getRole(), "SuperGenius User");

}

}

## Getting an Application Context in a junit.org Test Runner

package com.codesolid.tutorials.tests;

import com.codesolid.tutorials.UserStory;

import org.junit.Test;

import org.junit.Before;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.FileSystemXmlApplicationContext;

import static org.junit.Assert.\*;

/\* Here we run one of the same tests we run in ContextTests, but we separate it out here to demonstrate

\* we can do it with a POJU (Plain Old JUnit :) test runner.

\*/

public class JUnitContextTests {

ApplicationContext ac;

@Before

public void setUp()

{

ac = new FileSystemXmlApplicationContext("file:src/main/resources/spring-config.xml");

}

@Test

public void testUserCorrectFromPlainOldJUnitTest() {

UserStory story = (UserStory) ac.getBean("userStory");

// Spring is working fine using this app context

assertEquals(story.getUser().getRole(), "SuperGenius User");

// In this case our User is not wired up

UserStory story2 = new UserStory();

assertNull(story2.getUser());

}

@Test

public void testPrewiredUserCorrect() {

UserStory story = (UserStory) ac.getBean("userStory");

assertEquals(story.getUser().getRole(), "SuperGenius User");

}

}

**USING ANNOTATION BASED CONFIGURATION**

import static org.junit.Assert.assertEquals;

import javax.transaction.Transactional;

import org.junit.Test;

import org.junit.runner.RunWith;

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

import org.springframework.orm.hibernate4.HibernateTemplate;

import org.springframework.test.context.ContextConfiguration;

import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import org.springframework.test.context.transaction.TransactionConfiguration;

import com.concretepage.config.AppConfig;

import com.concretepage.dao.IPersonDao;

import com.concretepage.entity.Person;

@RunWith(SpringJUnit4ClassRunner.class)

@ContextConfiguration(classes = AppConfig.class)

@TransactionConfiguration(defaultRollback = true)

@Transactional

public class Spring4JUnit4Test {

@Autowired

private IPersonDao personDao;

@Autowired

private HibernateTemplate hibernateTemplate;

@Test

public void savePersonTest(){

personDao.savePerson();

Person person = hibernateTemplate.get(Person.class, 1);

assertEquals("Ram", person.getName());

}

}

### @RunWith(SpringJUnit4ClassRunner.class)

Spring provides SpringJUnit4ClassRunner that implements the functionality of JUnit4ClassRunner. JUnit4ClassRunner is provided by JUnit. In the demo @Test is provided by SpringJUnit4ClassRunner.

### @ContextConfiguration(classes = AppConfig.class)

@ContextConfiguration annotation uses the application context being used in the application. In our case, we are using AppConfig.java for bean definition.

### @Transactional

@Transactional annotation tells the test case that if transaction does not complete then revert other transactions.

### @TransactionConfiguration(defaultRollback = true)

Suppose if transaction is completed successfully, and test case has finished the data testing and then if we want to remove it from database, then TransactionConfiguration annotation works. To achieve it **defaultRollback** must be true.