Spring TestContext测试框架

本文参考了很多网上资料及Spring reference,详细介绍了Spring TestContext测试框架常用的一些东东.

若有疑问<[ymkyve501@gmail.com](mailto:ymkyve501@gmail.com)>

目录

[直接使用Junit测试Spring程序存在的不足 1](#_Toc263861317)

[Spring TestContext 测试框架体系结构 2](#_Toc263861318)

[**TestContext 核心类、支持类以及注解类** 2](#_Toc263861319)

[TestContext 2](#_Toc263861320)

[TestContextManager 2](#_Toc263861321)

[TestExecutionListener 2](#_Toc263861322)

[@TestExecutionListeners 3](#_Toc263861323)

[**@**ContextConfiguration 3](#_Toc263861324)

[@RunWith 3](#_Toc263861325)

[SpringJUnit4ClassRunner 3](#_Toc263861326)

[ContextCache 4](#_Toc263861327)

[ContextLoader 4](#_Toc263861328)

[ApplicationContextAware 4](#_Toc263861329)

[AbstractJUnit4SpringContextTests 4](#_Toc263861330)

[AbstractTransactionalJUnit4SpringContextTests 5](#_Toc263861331)

[常用注解 5](#_Toc263861332)

[@IfProfileValue 5](#_Toc263861333)

[@ProfileValueSourceConfiguration 5](#_Toc263861334)

[@DirtiesContext 6](#_Toc263861335)

[@ExpectedException 6](#_Toc263861336)

[@Timed 6](#_Toc263861337)

[@Repeat 7](#_Toc263861338)

[@Rollback 7](#_Toc263861339)

[@Transactional 7](#_Toc263861340)

[@NotTransactional 7](#_Toc263861341)

[@Autowired 8](#_Toc263861342)

[@Resource 8](#_Toc263861343)

[@Qualifier 9](#_Toc263861344)

[@TransactionConfiguration 9](#_Toc263861345)

[@BeforeTransaction 9](#_Toc263861346)

[@AfterTransaction 9](#_Toc263861347)

[总结 10](#_Toc263861348)

基于注解的TestContext测试框架,它采用注解技术可以让POJO成为Spring的测试用例,可以运行在Junit3.8 Junit4.4 TestNG等测试框架之下

# 直接使用Junit测试Spring程序存在的不足

1. **导致Spring容器多次初始化问题:**

根据 JUnit 测试用例的调用流程，每执行一个测试方法都会重新创建一个测试用例实例并调用其 setUp() 方法。由于在一般情况下，我们都在 setUp() 方法中初始化 Spring 容器，这意味着测试用例中有多少个测试方法，Spring 容器就会被重复初始化多少次。

1. **需要使用硬编码方式手工获取Bean**

在测试用例中，我们需要通过 ApplicationContext.getBean() 的方法从 Spirng 容器中获取需要测试的目标 Bean，并且还要进行造型操作。

1. **数据库现场容易遭受破坏**

测试方法可能会对数据库记录进行更改操作，破坏数据库现场。虽然是针对开发数据库进行测试工作的，但如果数据操作的影响是持久的，将会形成积累效应并影响到测试用例的再次执行。举个例子，假设在某个测试方法中往数据库插入一条 ID 为 1 的 t\_user 记录，第一次运行不会有问题，第二次运行时，就会因为主键冲突而导致测试用例执行失败。所以测试用例应该既能够完成测试固件业务功能正确性的检查，又能够容易地在测试完成后恢复现场，做到踏雪无迹、雁过无痕。

1. **不容易在同一事务下访问数据库以检验业务操作的正确性**

当测试固件操作数据库时，为了检测数据操作的正确性，需要通过一种方便途径在测试方法相同的事务环境下访问数据库，以检查测试固件数据操作的执行效果。如果直接使用 JUnit 进行测试，我们很难完成这项操作。

**Spring 测试框架是专门为测试基于 Spring 框架应用程序而设计的，它能够让测试用例非常方便地和 Spring 框架结合起来，以上所有问题都将迎刃而解。**

# Spring TestContext 测试框架体系结构

## **TestContext 核心类、支持类以及注解类**

TestContext 测试框架的核心由 org.springframework.test.context 包中三个类组成，分别是 TestContext 和 TestContextManager 类以及 TestExecutionListener 接口。

每次测试都会创建TestContextManager。TestContextManager管理了一个TestContext， 它负责持有当前测试的上下文。TestContextManager还负责在测试执行过程中更新TestContext的状态并代理到TestExecutionListener， 它用来监测测试实际的执行（如提供依赖注入、管理事务等等）

### TestContext

TestContext：封装测试执行的上下文，与当前使用的测试框架无关。

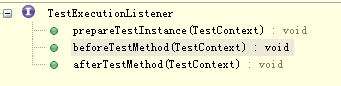
### TestContextManager

TestContextManager：Spring TestContext Framework的主入口点， 负责管理单独的TestContext并在定义好的执行点上向所有注册的TestExecutionListener发出事件通知： 测试实例的准备，先于特定的测试框架的前置方法，迟于后置方法。

### TestExecutionListener

TestExecutionListener：定义了一个监听器API与TestContextManager发布的测试执行事件进行交互， 而该监听器就是注册到这个TestContextManager上的。

Spring提供了TestExecutionListener的三个实现， 他们都是使用默认值进行配置的(通过@TestExecutionListeners注解)： DependencyInjectionTestExecutionListener、DirtiesContextTestExecutionListener及TransactionalTestExecutionListener， 他们对测试实例提供了依赖注入支持，处理@DirtiesContext注解，并分别使用默认的回滚语义对测试提供事务支持。



**DependencyInjectionTestExecutionListener**：该监听器提供了自动注入的功能，它负责解析测试用例中 @Autowried 注解并完成自动注入；

**DirtiesContextTestExecutionListener**：一般情况下测试方法并不会对 Spring 容器上下文造成破坏（改变 Bean 的配置信息等），如果某个测试方法确实会破坏 Spring 容器上下文，你可以显式地为该测试方法添加 @DirtiesContext 注解，以便 Spring TestContext 在测试该方法后刷新 Spring 容器的上下文，而 DirtiesContextTestExecutionListener 监听器的工作就是解析 @DirtiesContext 注解；

**TransactionalTestExecutionListener**：它负责解析 @Transaction、@NotTransactional 以及 @Rollback 等事务注解的注解。@Transaction 注解让测试方法工作于事务环境中，不过在测试方法返回前事务会被回滚。你可以使用 @Rollback(false) 让测试方法返回前提交事务。而 @NotTransactional 注解则让测试方法不工作于事务环境中。此外，你还可以使用类或方法级别的 @TransactionConfiguration 注解改变事务管理策略

### @TestExecutionListeners

用来注册TestExecutionListener

@TestExecutionListeners({DependencyInjectionTestExecutionListener.**class**, DirtiesContextTestExecutionListener.**class**})

### **@**ContextConfiguration

用来指定加载的Spring配置文件的位置,会加载默认配置文件

1. locations属性

**@**ContextConfiguration不带locations属性, GenericXmlContextLoader会基于测试类的名字产生一个默认的位置.

如果类名叫做com.example.MyTest，那么GenericXmlContextLoader就会从"classpath:/com/example/MyTest-context.xml"加载应用上下文。

**@**ContextConfiguration 若带有locations属性

@ContextConfiguration(locations= { "spring-service.xml","spring-service1.xml" }) 会去加载指定文件

1. inheritLocations属性

@ContextConfiguration的inheritLocations属性 (是否继承父类的locations),默认为true

@ContextConfiguration(locations={"/base-context.xml"})

**@ContextConfiguration(locations={"/extended-context.xml"})**

为true时,会将/base-context.xml, **/extended-context.xml 合并加载,若有重载的bean为子类为准**

**为false时,会**屏蔽掉父类的资源位置

### @RunWith

@RunWith 注解指定测试用例的运行器

@RunWith(SpringJUnit4ClassRunner.**class**)

### SpringJUnit4ClassRunner

Spring TestContext 框架提供了扩展于 org.junit.internal.runners.JUnit4ClassRunner 的 SpringJUnit4ClassRunner 运行器，它负责总装 Spring TestContext 测试框架并将其统一到 JUnit 4.4 框架中。

它负责无缝地将 TestContext 测试框架移花接木到 JUnit 4.4 测试框架中，它是 Spring TestContext 可以运行起来的根本所在

持有TestContextManager的引用

**private** **final** TestContextManager testContextManager;

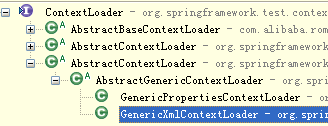
### ContextCache

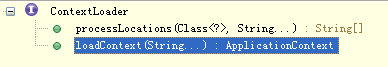
缓存ApplicationContext



### ContextLoader

**Interface 加载context接口**



****

GenericXmlContextLoader为默认的ContextLoader

### ApplicationContextAware

自动加载ApplicationContext

### AbstractJUnit4SpringContextTests

对集成了Spring TestContext Framework与JUnit 4.4环境中的ApplicationContext测试支持的基本测试类进行了抽取。

当你继承AbstractJUnit4SpringContextTests时，你就可以访问到protected的成员变量：

* + applicationContext：使用它进行显式的bean查找或者测试整个上下文的状态。

### AbstractTransactionalJUnit4SpringContextTests

对为JDBC访问增加便捷功能的AbstractJUnit4SpringContextTests的事务扩展进行抽象。 需要在ApplicationContext中定义一个javax.sql.DataSource bean和一个PlatformTransactionManager bean。

当你继承AbstractTransactionalJUnit4SpringContextTests类时，你就可以访问到下列protected的成员变量：

* + applicationContext：继承自父类AbstractJUnit4SpringContextTests。 使用它执行bean的查找或者测试整个上下文的状态
  + simpleJdbcTemplate：当通过查询来确认状态时非常有用。例如，应用代码要创建一个对象， 然后使用ORM工具将其持久化，这时你想在测试代码执行前后对其进行查询，以确定数据是否插入到数据库中。 （Spring会保证该查询运行在相同事务内。）你需要告诉你的ORM工具‘flush’其改变以正确完成任务，例如， 使用HibernateSession接口的flush()方法。

**提示**

这些类仅仅为扩展提供了方便。 如果你不想将你的测试类绑定到Spring的类上 - 例如，如果你要直接扩展你想测试的类 - 只需要通过@RunWith(SpringJUnit4ClassRunner.class)、 @ContextConfiguration、@TestExecutionListeners等注解来配置你自己的测试类就可以了。

## 常用注解

Spring框架在org.springframework.test.annotation 包中提供了常用的Spring特定的注解集，如果你在Java5或以上版本开发，可以在测试中使用它。

### @IfProfileValue

提示一下，注解测试只针对特定的测试环境。 如果配置的ProfileValueSource类返回对应的提供者的名称值， 这个测试就可以启动。这个注解可以应用到一个类或者单独的方法。

@IfProfileValue(name="java.vendor", value="Sun Microsystems Inc.")

**public** **void** testJoin() {

System.*out*.println(System.*getProperty*("java.vendor"));

}

### @ProfileValueSourceConfiguration

类级别注解用来指定当通过@IfProfileValue注解获取已配置的profile值时使用何种ProfileValueSource。 如果@ProfileValueSourceConfiguration没有在测试中声明，将默认使用SystemProfileValueSource

@ProfileValueSourceConfiguration(CustomProfileValueSource.class)

SystemProfileValueSource中的代码

**public** String get(String key) {

Assert.*hasText*(key, "'key' must not be empty");

**return** System.*getProperty*(key);

}

### @DirtiesContext

在测试方法上出现这个注解时，表明底层Spring容器在该方法的执行中被“污染”，从而必须在方法执行结束后重新创建（无论该测试是否通过）。

@DirtiesContext

**public** **void** testJoinC() {

Assert.*assertEquals*(**true**, **false**);

}

### @ExpectedException

表明被注解方法预期在执行中抛出一个异常。预期异常的类型在注解中给定。如果该异常的实例在测试方法执行中被抛出， 则测试通过。同样的如果该异常实例没有在测试方法执行时抛出，则测试失败。

@ExpectedException(MemberServiceException.**class**)

**public** **void** testChangePassword() {}

协同使用Spring的@ExpectedException注解与JUnit 4的@Test(expected=...)会导致一个不可避免的冲突。 因此当与JUnit 4集成时，开发者必须选择其中一个，在这种情况下建议使用显式的JUnit 4配置。

### @Timed

表明被注解的测试方法必须在规定的时间区间内执行完成（以毫秒记）。如果测试执行时间超过了规定的时间区间，测试就失败了。

注意该时间区间包括测试方法本身的执行，任何重复测试（参见 @Repeat），还有任何测试fixture的set up或tear down时间。

@Timed(millis=100)

**public** **void** testPersonByVerifiedMobile(){}

Spring的@Timed注解与JUnit 4的@Test(timeout=...)支持具有不同的语义。 特别地，鉴于JUnit 4处理测试执行超时（如通过在一个单独的线程中执行测试方法）的方式， 我们不可能在一个事务上下文中的测试方法上使用JUnit的@Test(timeout=...)配置。因此， 如果你想将一个测试方法配置成计时且具事务性的， 你就必须联合使用Spring的@Timed及@Transactional注解。 还值得注意的是@Test(timeout=...)只管测试方法本身执行的次数，如果超出的话立刻就会失败； 然而，@Timed关注的是测试执行的总时间（包括建立和销毁操作以及重复），并且不会令测试失败。

### @Repeat

表明被注解的测试方法必须重复执行。执行的次数在注解中声明。

注意重复执行范围包括包括测试方法本身的执行，以及任何测试fixture的set up或tear down。

@Test

@Timed(millis=100)

@Repeat(10)

**public** **void** testPersonByVerifiedMobile()

### @Rollback

表明被注解方法的事务在完成后是否需要被回滚。 如果true，事务将被回滚，否则事务将被提交。 使用@Rollback接口来在类级别覆写配置的默认回滚标志。

@Rollback(**false**)

**public** **void** testPersonByVerifiedMobile()

### @Transactional

出现该注解表明测试方法必须在事务中执行。

### @NotTransactional

出现该注解表明测试方法必须不在事务中执行。

@NotTransactional

**public** **void** testPersonByVerifiedMobile()

### @Autowired

按类型注入(可以写在属性上也可以写在setter上)

@Autowired

Join join11;

@Autowired

**public** **void** setJoin11(Join join11) {

**this**.join11 = join11;

}

### @Resource

JSR 250中注解. 与@Autowired类似

@Resource

**public** **void** setJoin(Join join) {

**this**.join = join;

}

@Resource

Join join;

@Autowired 与@Resource 都可以用来装配bean.  都可以写在字段上,或写在set方法上

@Autowired (srping提供的) 默认按类型装配

@Resource ( j2ee提供的 ) 默认按名称装配,当找不到(不写name属性)名称匹配的bean再按类型装配.

可以通过@Resource(name="beanName") 指定被注入的bean的名称, 要是指定了name属性, 就用 字段名 去做name属性值,一般不用写name属性.

@Resource(name="beanName")指定了name属性,按名称注入但没找到bean, 就不会再按类型装配了.

推荐@Autowired 与@Resource作用有字段 上,  就不用写set方法了,也不要写@Resource(name="beanName").

@Retention(RetentionPolicy.*RUNTIME*)

@Target({ElementType.*CONSTRUCTOR*, ElementType.*FIELD*, ElementType.*METHOD*})

**public** **@interface** Autowired

### @Qualifier

在需要更多控制的时候，任何autowired的域、构造参数、或者方法参数可以进一步加注@Qualifier注解。qualifier可以包含一个字符串值，在这种情况下，Spring会试图通过名字来找到对应的对象。

@Autowired

@Qualifier("join")

Join join111;

@Qualifier作为一个独立注解存在的主要原因是它可以被应用在构造器参数或方法参数上，但上文提到的@Autowired注解只能运用在构造器或方法本身。

**public** SpringTest(@Qualifier("join")Join join){

}

看下@Qualifier的定义

@Target({ElementType.*FIELD*, ElementType.*PARAMETER*, ElementType.*TYPE*, ElementType.*ANNOTATION\_TYPE*})

@Retention(RetentionPolicy.*RUNTIME*)

@Inherited

@Documented

**public** **@interface** Qualifier

### @TransactionConfiguration

为配置事务性测试定义了类级别的元数据。特别地，如果需要的PlatformTransactionManager不是“transactionManager”的话， 那么可以显式配置驱动事务的PlatformTransactionManager的bean名字。此外， 可以将defaultRollback标志改为false。通常， @TransactionConfiguration与@ContextConfiguration搭配使用。

@TransactionConfiguration(transactionManager="txMgr", defaultRollback=**false**)

**public** **class** SpringTest

### @BeforeTransaction

表明被注解的public void方法应该在测试方法的事务开始之前执行， 该事务是通过@Transactional注解来配置的。

### @AfterTransaction

表明被注解的public void方法应该在测试方法的事务结束之后执行， 该事务是通过@Transactional注解来配置的。

标注 @Before 或 @After 注解的方法和测试方法运行在同一个事务中，但有时我们希望在测试方法的事务开始之前或完成之后执行某些方法以便获取数据库现场的一些情况。这时，可以使用 Spring TestContext 的 @BeforeTransaction 和 @AfterTransaction 注解来达到目录（这两个注解位于 org.springframework.test.context.transaction 包中）。

# 总结

Spring 的 TestContext 测试框架不但可以整合到 JUnit 4.4 测试框架上，而且还可以整合到 JUnit 3.8 以及 TestNG 等测试框架上。目前已经提供了对 JUnit 3.8 以及 TestNG 的支持，你可以分别在 org.springframework.test.context.junit38 和 org.springframework.test.context.testng 包下找到整合的帮助类。

参考资料:

1. <http://www.ibm.com/developerworks/cn/java/j-lo-spring25-test/?S_TACT=105AGX52&S_CMP=techcsdn>
2. Spring-Reference\_zh\_CN.chm