Java Web Code Test

目录

[1. 概述 3](#_Toc31980)

[1.1. 常用的单元测试框架 3](#_Toc10392)

[1.2. 原始的测试方法 4](#_Toc11354)

[2. Spring test 框架进行测试 5](#_Toc8754)

[2.1. 注解测试单元 6](#_Toc15621)

[3. 为什么要Unit Test 7](#_Toc14441)

[3.1. 基于java开发的技术特点 7](#_Toc24534)

[3.2. Unit test要解决的问题 8](#_Toc25912)

[3.3. Unit test的主要任务 9](#_Toc18627)

[4. 测试工具 11](#_Toc20220)

[4.1. JUnit4 11](#_Toc17910)

[4.2. Assert 11](#_Toc12170)

[4.3. Mock/Stub 12](#_Toc1267)

[4.4. EasyMock 13](#_Toc5433)

[4.5. Mockito 13](#_Toc7701)

[4.6. 代码覆盖率 13](#_Toc21737)

[4.6.1. Jacoco 13](#_Toc226)

[4.6.2. Java Agent嵌入 14](#_Toc32140)

[4.6.3. Java Report 14](#_Toc28670)

1. 概述

单元测试和集成测试在我们的软件开发整个流程中占有举足轻重的地位，一方面，程序员通过编写单元测试来验证自己程序的有效性，另外一方面，管理者通过持续自动的执行单元测试和分析单元测试的覆盖率等来确保软件本身的质量。这里，我们先不谈单元测试本身的重要性，对于目前大多数的基于 Java 的企业应用软件来说，Spring 已经成为了标准配置，一方面它实现了程序之间的低耦合度，另外也通过一些配置减少了企业软件集成的工作量，例如和 Hibernate、Struts 等的集成。那么，有个问题，在普遍使用 Spring 的应用程序中，我们如何去做单元测试？或者说，我们怎么样能高效的在 Spring 生态系统中实现各种单元测试手段？这就是本文章要告诉大家的事情。

* 1. 常用的单元测试框架

单元测试目前主要的框架包括 Junit、TestNG，还有些 MOCK 框架，例如 Jmock、Easymock、PowerMock 等，这些都是单元测试的利器，但是当把他们用在 Spring 的开发环境中，还是那么高效么？还好，Spring 提供了单元测试的强大支持，主要特性包括：

1. 支持主流的测试框架 Junit 和 TestNG。
2. 支持在测试类中使用依赖注入 Denpendency Injection。
3. 支持测试类的自动化事务管理。
4. 支持使用各种注释标签，提高开发效率和代码简洁性。
5. Spring 3.1 更是支持在测试类中使用非 XML 配置方法和基于 Profile 的 bean 配置模式。
   1. 原始的测试方法

原始测试的代码如下：

*public class UserServiceTestOriginal {*

*private static IUserDao userDao;*

*@BeforeClass*

*public static void init()*

*{*

*/\*\*测试前手动加载配置文件，并获取Service Bean\*\*/*

*ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");*

*userDao = (IUserDao) context.getBean("userDaoImpl");*

*}*

*@Test*

*public void testAddUser()*

*{*

*TestUser testUser = new TestUser();*

*testUser.setUserName("JackyTest1");*

*testUser.setUserPassword("12345678");*

*TestUser equalUser = null;*

*try {*

*userDao.saveUser(testUser);*

*equalUser = (TestUser) userDao.getUserByUserName("JackyTest1");*

*System.out.println("Equal user:" + equalUser.toString());*

*assert (equalUser) != null;*

*} catch (Exception e) {*

*// TODO: handle exception*

*fail(e.getMessage());*

*}finally {*

*/\*\*手动删除测试项\*\*/*

*userDao.deleteUser(equalUser);*

*}*

*}*

*}*

对于这种不使用 Spring test 框架进行的单元测试，我们注意到，需要做这些工作：

1. 在测试开始之前，需要手工加载 Spring 的配置文件，并获取需要的 bean 实例
2. 在测试结束的时候，需要手工清空搭建的数据库环境，比如清除您插入或者更新的数据，以保证对下一个测试没有影响。

另外，在这个测试类中，我们还不能使用 Spring 的依赖注入特性。一切都靠手工编码实现。好，那么我们看看 Spring test 框架能做到什么。

1. Spring test 框架进行测试

继续使用 Junit4 测试框架，包括其 @Test 注释标签和相关的类和方法的定义，这些都不用变：

1. 需要通过 @RunWith(SpringJUnit4ClassRunner.class) 来启动 Spring 对测试类的支持。
2. 需要通过 @ContextConfiguration 注释标签来指定 Spring 配置文件或者配置类的位置。
3. 需要通过 @Transactional 来启用自动的事务管理。
4. 可以使用 @Autowired 自动织入 Spring 的 bean 用来测试。

另外您不再需要：

1. 手工加载 Spring 的配置文件。
2. 手工清理数据库的每次变更。
3. 手工获取 application context 然后获取 bean 实例。

transactionManager 是 Spring test 框架用来做事务管理的管理器，需在ApplicationContext.xml对transactionManager进行配置（Hibernate事务配置）：

*<bean id="transactionManager" class="org.springframework.orm.hibernate5.HibernateTransactionManager">*

*<property name="sessionFactory" ref="sessionFactory" />*

*</bean>*

* 1. 注解测试单元

测试单元配置注解如下：

*/\*\*范例代码\*\*/*

*@RunWith(SpringJUnit4ClassRunner.class)*

*@ContextConfiguration(locations={"classpath:applicationContext.xml"})*

*@Transactional*

*public class UserServiceTestSpring {*

*@Autowired*

*private IUserDao userDaoImpl;*

*@Test*

*public void testDeleteUserByName() {*

*int ret = userDaoImpl.deleteUserByName("JackyTest");*

*if(ret != 1)*

*{*

*fail("Delete fialed!");*

*}*

*}*

}

对这个类解释一下：

1. @RunWith 注释标签是 Junit 提供的，用来说明此测试类的运行者，这里用了 SpringJUnit4ClassRunner，这个类是一个针对 Junit 运行环境的自定义扩展，用来标准化在 Spring 环境中 Junit4.5 的测试用例，例如支持的注释标签的标准化。
2. @ContextConfiguration 注释标签是 Spring test context 提供的，用来指定 Spring 配置信息的来源，支持指定 XML 文件位置或者 Spring 配置类名，这里我们指定 classpath 下的 /config/Spring-db1.xml 为配置文件的位置。
3. @Transactional 注释标签是表明此测试类的事务启用，这样所有的测试方案都会自动的 rollback，即您不用自己清除自己所做的任何对数据库的变更了。
4. @Autowired 体现了我们的测试类也是在 Spring 的容器中管理的，他可以获取容器的 bean 的注入，您不用自己手工获取要测试的 bean 实例了。
5. 为什么要Unit Test
   1. 基于java开发的技术特点
6. **容器管理的组件开发**

开发WEB应用程序，实质上就是在开发一系列组件。组件的类型有很多，JavaBean、Servlet、Filter、JSP Taglib、EJB、Spring Bean，等等。这些组件一般是不能独立运行的，需要将它们布署到WEB服务器，通过与WEB容器或EJB容器交互才能实现一定的业务逻辑，也就是说，组件依赖的许多对象是运行时由容器创建的，如HttpServletRequest、HttpServletResponse、ServletContext、SessionContext、FilterChain、PageContext，等等，这就面临着单元测试时这些容器对象如何生成的问题。对于这些组件的测试一般我们有两种单元测试方法：一种是对被测组件进行隔离测试，组件依赖的服务器环境对象用桩取代，它的缺点是构建桩太麻烦；另一种是将组件运行在真实服务器环境下，有别于系统测试，被测对象是我们主动在测试代码中创建的，它的优点是更接近于真实环境、免除了构建桩的工作量。

1. **页面显示的视图开发**

有很多技术用于将视图与业务逻辑分开，如Struts、JSF、WebWork、Spring等，页面开发的技术也有很多，如Html、XML、JSP、JavaScript、Velocity，等等，对于这些页面文件，技术上很难进行编译和静态检查，对它们的测试虽然有一些工具支持，但效果均不理想，而且代码Review大家反映也很难发现实质性的页面问题，如何确保页面文件的质量一直是单元测试要解决的问题。

1. **框架复杂**

WEB开发使用的框架很少有自己独立设计的，一般都会使用现成的架构，如Struts、JSF、WebWork、EJB、Spring、Hibernate、iBATIS等等，不同的框架，能够支持单元测试的程度也是不一样的。

1. **数据库访问技术多**

业软的WEB应用开发很少有不访问数据库的，因为多是面向业务的开发，业务离不开数据存储。项目组可能会选择不同的数据库访问技术，如JDBC、EntityBean、Spring、Hibernate、iBATIS等，对于涉及数据库访问的代码如何做单元测试也是我们需要解决的问题。

* 1. Unit test要解决的问题

有许多的单元测试技术和工具，综合起来，无非是为了解决以下问题。

1. **驱动（Driver）—驱动被测单元**

单元不能独立运行，必须实现调用它们的代码，我们称其为驱动代码，其实最简单的驱动就是实现main方法，大家常用的驱动典型工具就是JUnit。

1. **构建桩（Stub）—模拟被测单元依赖的对象**

被测的孤立单元通常会对其它对象有依赖，这种依赖通常表现在：依赖对象通过被测方法参数传入或者被测对象保存有依赖的对象引用，然后在被测方法中调用了依赖对象的方法。构造依赖对象，一方面我们可以直接将开发完成的并且之前已经过单元测试的代码直接拿过来用，另一方面，也是更常用的方法，就是自己构建模拟对象，我们称其为桩，但自己写桩很麻烦，工作量大，现成构建桩的理想工具是EasyMock。

1. **验证（Verify）**

用例执行是否成功，需要在测试中添加验证点，需要将预期结果与测试执行获得的实际结果进行比较，为此JUnit为我们提供了验证的基本逻辑框架，其它工具可以基于它实现更复杂的验证逻辑，如DBUnit实现的对数据库表数据的验证。

1. **用例管理**

常有同事提到用main方法也能实现对被测单元的驱动，但我觉得最大的不足是无法实现对用例的有效管理，为此JUnit为我们提供了用例管理的基础框架，通过引入测试套的概念将用例有效地组织起来。

1. **结果输出（Report）**

测试结束后要能够将本次运行的结果情况形成报告，并以图形化直观的形式报告给用户。JUnit也为我们做到了，尤其是IDE与JUnit的集成，使我们在开发过程中做单元测试变得更加方便。

1. **覆盖率检查**

公司要求被测代码要求达到语句的100%覆盖，是否覆盖到了，我们可以借助覆盖率检查工具，做测试执行的同时进行覆盖率检查，对未覆盖到的代码可能会发现两类问题：不可达代码，这样的代码需要优化；用例设计不充分，这时就要及时补充用例。常用的覆盖率检查工具有PureCoverage、Cobertura。

1. **测试自动化**

实现测试自动化的好处大家都很明白，方便回归测试，节省了工作量；另一个好处是便于对测试的监控，这一点我们在后面会谈到。我们模索系统测试自动化已有多年，但效果都不理想，主要原因我觉得和系统测试本身的特点有关，因为系统测试是站在用户角度看系统功能的整体表现（这其中最讨厌的是我们经常还有需求变更，如何做到以不变应万变，我们曾经尝试过，但效果均不理想）。但单元测试不同于系统测试，单元不能独立运行，需要我们实现驱动代码，它的这个。

* 1. Unit test的主要任务

单元测试是编写测试代码，用来检测特定的、明确的、细颗粒的功能。单元测试并不一定保证程序功能是正确的，更不保证整体业务是准备的。

单元测试不仅仅用来保证当前代码的正确性，更重要的是用来保证代码**修复**、**改进**或**重构**之后的正确性。

一般来说，单元测试任务包括

1. 接口功能测试：用来保证接口功能的正确性。
2. 局部数据结构测试（不常用）：用来保证接口中的数据结构是正确的
3. 比如变量有无初始值
4. 变量是否溢出
5. 边界条件测试
6. **变量没有赋值**（即为NULL）
7. 变量是数值（或字符)
8. **主要边界**：最小值，最大值，无穷大（对于DOUBLE等）
9. **溢出边界**（期望异常或拒绝服务）：最小值-1，最大值+1
10. **临近边界**：最小值+1，最大值-1
11. 变量是字符串
12. 引用“字符变量”的边界
13. **空字符串**
14. 对字符串长度应用“数值变量”的边界
15. 变量是集合
16. **空集合**
17. **对集合的大小应用“数值变量”的边界**
18. **调整次序**：升序、降序
19. 变量有规律
20. 比如对于Math.sqrt，给出n^2-1，和n^2+1的边界
21. 所有独立执行通路测试：保证每一条代码，每个分支都经过测试
22. 代码覆盖率
23. **语句覆盖**：保证每一个语句都执行到了
24. 判定覆盖（分支覆盖）：保证每一个分支都执行到
25. 条件覆盖：保证每一个条件都覆盖到true和false（即if、while中的条件语句）
26. 路径覆盖：保证每一个路径都覆盖到
27. 相关软件
28. Cobertura：语句覆盖
29. Emma: Eclipse插件Eclemma
30. 各条错误处理通路测试：保证每一个异常都经过测试
31. 测试工具
    1. JUnit4

与JUnit3不同，JUnit4通过注解的方式来识别测试方法。目前支持的主要注解有：

1. @BeforeClass 全局只会执行一次，而且是第一个运行
2. @Before 在测试方法运行之前运行
3. @Test 测试方法
4. @After 在测试方法运行之后允许
5. @AfterClass 全局只会执行一次，而且是最后一个运行
6. @Ignore 忽略此方法
   1. Assert

Junit3和Junit4都提供了一个Assert类（虽然package不同，但是大致差不多）。Assert类中定义了很多静态方法来进行断言。列表如下：

1. assertTrue(String message, boolean condition) 要求condition == true。
2. assertFalse(String message, boolean condition) 要求condition == false。
3. fail(String message) 必然失败，同样要求代码不可达。
4. assertEquals(String message, XXX expected,XXX actual) 要求expected.equals(actual)。
5. assertArrayEquals(String message, XXX[] expecteds,XXX [] actuals) 要求expected.equalsArray(actual)。
6. assertNotNull(String message, Object object) 要求object!=null。
7. assertNull(String message, Object object) 要求object==null。
8. assertSame(String message, Object expected, Object actual) 要求expected == actual。
9. assertNotSame(String message, Object unexpected,Object actual) 要求expected != actual。
10. assertThat(String reason, T actual, Matcher matcher) 要求matcher.matches(actual) == true。
    1. Mock/Stub

Mock和Stub是两种测试代码功能的方法。Mock测重于对功能的模拟。Stub测重于对功能的测试重现。比如对于List接口，Mock会直接对List进行模拟，而Stub会新建一个实现了List的TestList，在其中编写测试的代码。  
强烈建议优先选择Mock方式，因为Mock方式下，模拟代码与测试代码放在一起，易读性好，而且扩展性、灵活性都比Stub好。  
比较流行的Mock有：

1. [JMock](http://jmock.org/" \o "JMock" \t "https://www.cnblogs.com/AloneSword/p/_blank)
2. [EasyMock](http://www.easymock.org/" \o "EasyMock" \t "https://www.cnblogs.com/AloneSword/p/_blank)
3. [Mockito](http://blog.thihy.info/post/mockito.googlecode.com" \o "mockito" \t "https://www.cnblogs.com/AloneSword/p/_blank)
4. [powermock](http://code.google.com/p/powermock/" \o "powermock" \t "https://www.cnblogs.com/AloneSword/p/_blank)

其中EasyMock和Mockito对于Java接口使用接口代理的方式来模拟，对于Java类使用继承的方式来模拟（也即会创建一个新的Class类）。Mockito支持spy方式，可以对实例进行模拟。但它们都不能对静态方法和final类进行模拟，powermock通过修改字节码来支持了此功能。

* 1. EasyMock

IBM上有几篇介绍EasyMock使用方法和原理的文章：[EasyMock 使用方法与原理剖析](http://www.ibm.com/developerworks/cn/opensource/os-cn-easymock/" \o "EasyMock 使用方法与原理剖析" \t "https://www.cnblogs.com/AloneSword/p/_blank)，[使用 EasyMock 更轻松地进行测试](http://www.ibm.com/developerworks/cn/java/j-easymock.html" \o "使用 EasyMock 更轻松地进行测试" \t "https://www.cnblogs.com/AloneSword/p/_blank)。  
 EasyMock把测试过程分为三步：录制、运行测试代码、验证期望。  
录制过程大概就是：期望method(params)执行times次（默认一次），返回result（可选），抛出exception异常（可选）。  
 验证期望过程将会检查方法的调用次数。

EasyMock还支持严格的检查，要求执行的方法次序与期望的完全一致。

* 1. Mockito

Mockito是Google Code上的一个开源项目，Api相对于EasyMock更好友好。与EasyMock不同的是，Mockito没有录制过程，只需要在“运行测试代码”之前对接口进行Stub，也即设置方法的返回值或抛出的异常，然后直接运行测试代码，运行期间调用Mock的方法，会返回预先设置的返回值或抛出异常，最后再对测试代码进行验证。官方提供了很多样例，基本上包括了所有功能。

* 1. 代码覆盖率

比较流行的工具是Emma和[Jacoco](http://www.eclemma.org/jacoco/trunk/doc/" \t "https://www.cnblogs.com/AloneSword/p/_blank),Ecliplse插件有eclemma。eclemma2.0之前采用的是Emma，之后采用的是Jacoco。这里主要介绍一下Jacoco。Eclmama由于是Eclipse插件，所以非常易用，就不多做介绍了。

* + 1. Jacoco

Jacoco可以嵌入到Ant、Maven中，也可以使用Java Agent技术监控任意Java程序，也可以使用Java Api来定制功能。  
Jacoco会监控JVM中的调用，生成监控结果（默认保存在jacoco.exec文件中），然后分析此结果，配合源代码生成覆盖率报告。需要注意的是：监控和分析这两步，必须使用相同的Class文件，否则由于Class不同，而无法定位到具体的方法，导致覆盖率均为0%。

* + 1. Java Agent嵌入

首先，需要下载jacocoagent.jar文件，然后在Java程序启动参数后面加上 -javaagent:[yourpath/]jacocoagent.jar=[option1]=[value1],[option2]=[value2]，默认会在JVM关闭时（注意不能是kill -9），输出监控结果到jacoco.exec文件中，也可以通过socket来实时地输出监控报告（可以在Example代码中找到简单实现）。

* + 1. Java Report

可以使用Ant、Mvn或Eclipse来分析jacoco.exec文件，也可以通过API来分析。