Spring框架下+Jmokit 进行UnitedTest

目录

[1主要测试范围 1](#_Toc451364825)

[2测试范围详解 1](#_Toc451364826)

[3测试方法示例 1](#_Toc451364827)

[3.1项目基本情况说明 1](#_Toc451364828)

[3.2项目测试方法说明 1](#_Toc451364829)

[3.2.1 与spring框架相关测试 1](#_Toc451364830)

[3.2.2 与spring框架无关测试 1](#_Toc451364831)

[3.2.3 与spring框架相关的内部功能模块包命名 1](#_Toc451364832)

[4 关于测试覆盖率的兼容性 1](#_Toc451364833)

# 1主要测试范围

首先明确本文所涉及的测试只涵盖业务相关逻辑与功能逻辑，并不包括外部相关模块的测试，因为外部相关模块是服务于内部功能模块，内部功能模块的逻辑需要覆盖外部模块的正常情况与非正常情况处理。

结论：**只测**业务相关逻辑与功能逻辑即内部功能模块，外部相关模块**不测试**，留到最终集成测试进行。

# 2测试范围详解

如上所述，我们将系统各个模块分为了两大类，外部相关模块与内部功能模块。

1.内部功能模块：不会对外部系统进行操作的模块。---这个方便每次都测

2.外部相关模块：直接对外部系统进行操作的模块，如database jms queue http请求 等 ---依赖第三方难以每次都测。相关指的是与系统外的其它系统有联系。

另外，内部功能模块对于外面相关模块的依赖需要使用组合方式，即可以通过构造函数或者setter进行设置，即不能出现hard code或者方法内部的new或者直接调用外部类的涉及外部使用的静态方法。建议使用接口来做隔离。

整体结构如下图所示：



从上图可以看出，外部相关模块都涉及对外部系统的访问，很难测试！

外部相关模块：用来请求系统外的服务，那么我们只关心它的返回值情况，可以分为以下两类。A正常返回。B异常返回。对于这两大类，我们系统全部需覆盖来完成正常工能和达到健壮性。但在测试的时候一直保持对外连接或者在本地都搭建一套系统并不现实，而且也会增加开发难度。

结论：使用stub或者mock技术进行隔离，模拟AB两种情况。 （但基于spring框架的系统或多或少都依赖了其自动注入的特性，所以本文使用stub来处理与spring框架相关的外部模块）

内部功能模块：完成系统业务工能并实现健壮性，所以需要测试两种情况。A其在正常情况下的处理是否符合预期。B对各种可预期的异常情况，处理是否符合预期。

# 3测试方法示例

## 3.1项目基本情况说明

Demo系统模拟了请求一个用户信息的整体过程，即通过用户id与性别获取用户信息。项目尽量模拟了常用到的Spring技术。项目结构如下图：



服务过程如下：

Step 1： controller接到用户请求，进行参数判断与处理过程

Step 2：交于service处理具体用户请求，这里涉及业务逻辑与DAO层异常处理

Step 3: 交于DAO层获取数据。有两种情况，正常返回，不正常用null返回（为了模拟没有做的太复杂，当然也可以是异常抛出，只是service层处理方式做一些变化）

## 3.2项目测试方法说明

### 3.2.1 与spring框架相关测试

第一步：对外部相关模块写stub进行场景模拟。

按第二章节的划分，如果要测和dao相关的模块需要先创建stub。不论直接相关还是间接相关，都需要把spring的依赖改到stub上。

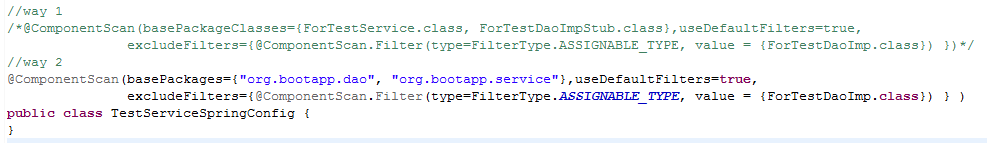
而DAO目前只有两种情况，1.正常返回。2.如果id为0则返回null。Stub上的name与真实的一至（spring 自动注入需要），如下图



第二步：确定测试目标（需要测试的内部功能模块），并写spring配置文件

这用service层做为示例。

这里最重要的是要过滤掉真实的功能模块，如上图所示的ForTestDaoImp.细节如下图所示

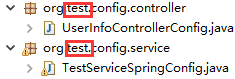


第三步：写测试

按spring要求引入测试配置文件。如下图



**关键点：因为配置是有@Configuration标注，所以是可以被扫描并被spring自动装配，所以一定要避免测试的时候不同的测试扫描到其它的配置文件，而产生相互干扰，请将其放到配置文件中@ComponentScan扫不到的目录中。即每个配置在指明使用后，该配置不能扫描到其它的配置。如下图中，有两个测试配置，有不同的@ComponentScan方案，但最终都不会感知到对方的存在！看代码会发现他们都是基于org.bootapp开始扫描。Org.test这样就永远不会被扫到**



以上过程就是写一个与外部相关模块有耦合关系的模块的测试过程。

### 3.2.2 与spring框架无关测试

这个比较简单，Demo中随意加了一个org.bootapp.utilities.jmockit.test包，里面有两个类CompositeFunctionClass🡪BasedFunctionClass。在CompositeFunctionClassTest中mock了BasedFunctionClass对CompositeFunctionClass的情况做了测试。CompositeFunctionClass不是外部相关的，那么就不需要mock。

至于对BasedFunctionClass简单的Junit test即可。

细节参看代码！

### 3.2.3 与spring框架相关的内部功能模块包命名

建议细化命名，因为这样在定义测试扫描的时候在包内涉及的类会少，一次性引入的测试类相应也会少，因为是自动扫描所有如果范围太大，那么不需要测试的对象也会被spring自动装配。

例如：org.bootapp.service中有两个类为aservice,bservice两分别依赖了adao,bdao那么在测试aservice的时候也需要把b相关的所有stub都准备好

**3.2.4 用mockito进行测试**

这种方式可以理解成完全放弃了spring容器，完全使用mockito来做测试。

Service层见ServiceTestWithMockitoTest

Controller层见UserInfoControllerByMockitTest

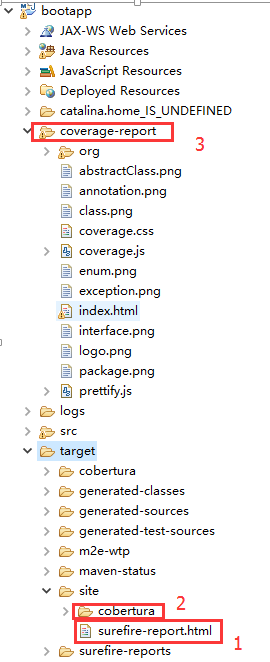
# 4 关于测试覆盖率的兼容性

从第三章可以看出，我们测试中涉及一般的Junit测试（3.2.1），带spring框架的测试。这两部分都混合了Jmockit。如何做覆盖率报告？或者以前的覆盖率报告是否可用呢？

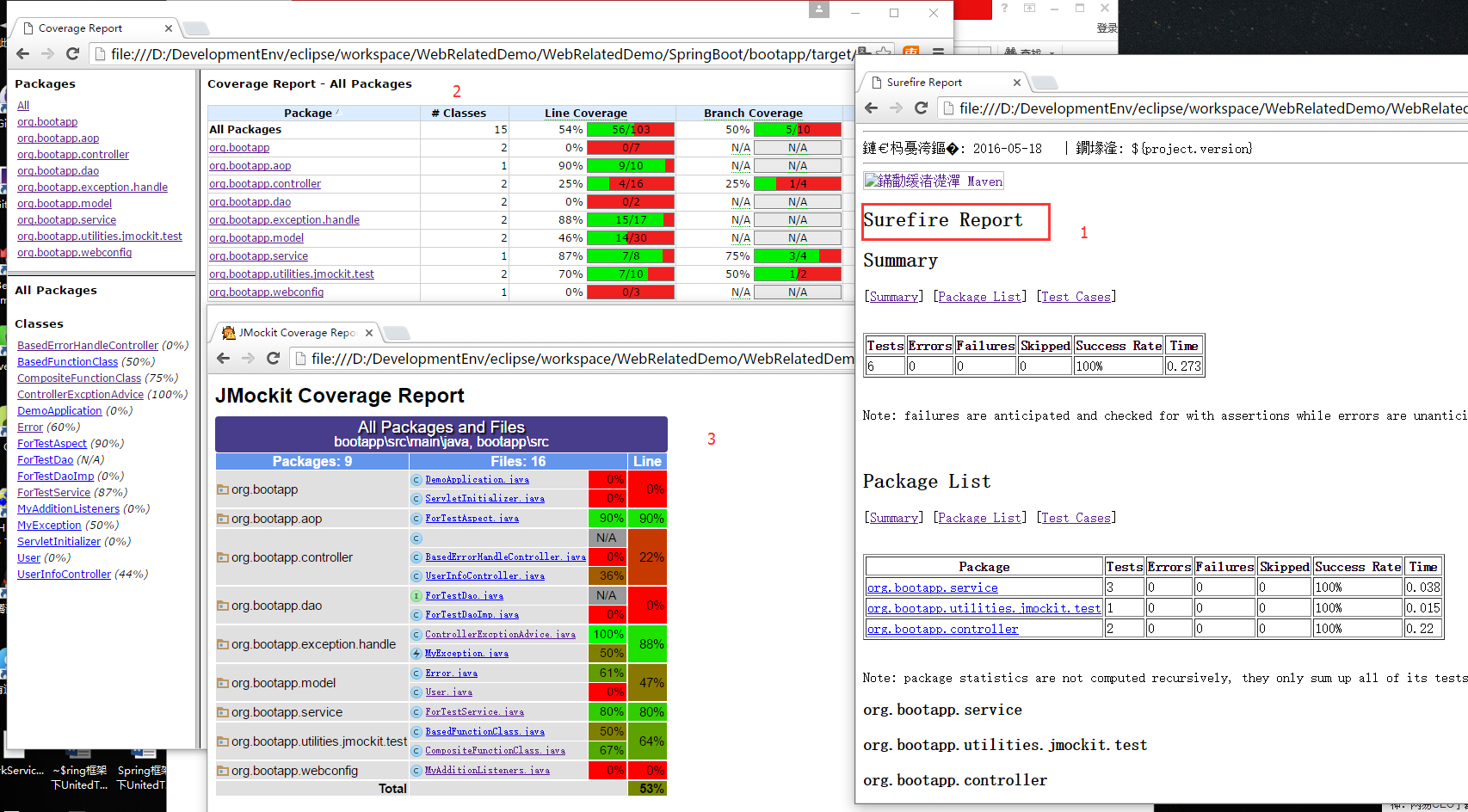
经过测试答案是肯定的！但他们的结果会有些出入，但不影响对测试覆盖的判读。只是统计方式不一样，细节请自看

Demo中使用了三个测试报告：1.测试通过整体报告（surefire report）;2.codehaus覆盖率测试报告;3.jmockit-coverage覆盖率测试报告。上面说的出入是2 3两者数据上的小出入。

项目运行mvn cobertura:cobertura surefire-report:report然后会自动在site下生成报告（**所有测试类都需要以Test结尾**，否则可能出现不执行测试，mockit的测试类就是这样）。如下图所示，标号与上面数字对应



三者报告如下：



# Spring boot 1.4测试方式

https://spring.io/blog/2016/04/15/testing-improvements-in-spring-boot-1-4