

[首页 \(/\)](#) / [文章 \(/tutorials\)](#) / [SpringBoot与JUnit+Mockito 单元测试](#)[参考代码 \(/tutorial/JunitForSpringBoot/repo\)](#)

SpringBoot与JUnit+Mockito 单元测试

周鸿博 (/user/zhbzhbzbz) 发布于 1月10日 0评论 3773浏览

[mockito \(/tag/mockito/tutorials\)](#)[springboot \(/tag/springboot/tutorials\)](#)[junit \(/tag/junit/tutorials\)](#)

2 个

2 个

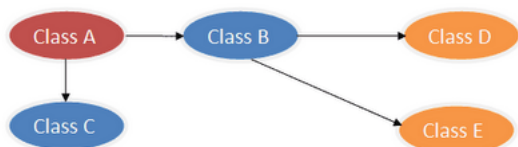
JUnit和Mockito

对于JUnit, 这里就不详细介绍了, 网上的教程有很多, 比如 这个 (<http://www.cnblogs.com/yangxia-test/p/3991572.html>) 和 这个 (<http://huihai.iteye.com/blog/1986568>)。

下面主要介绍一下**Mockito**。

什么是mock测试, 什么是mock对象?

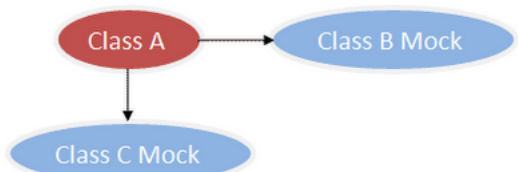
先来看看下面这个示例:



(<http://dl.iteye.com/upload/attachment/0067/5882/02030f95-ba9a-3104-b0f1-d7d8f02029fd.png>)

从上图可以看出如果我们要对A进行测试, 那么就要先把整个依赖树构建出来, 也就是BCDE的实例。

一种替代方案就是使用mocks



(<http://dl.iteye.com/upload/attachment/0067/5884/2fab8997-d489-396e-9365-2ae1fe94b6c2.png>)

从图中可以清晰的看出:

- mock对象就是在调试期间用来作为真实对象的替代品
- mock测试就是在测试过程中, 对那些不容易构建的对象用一个虚拟对象来代替测试的方法就叫mock测试

模拟的好处是什么?

- 提前创建测试; TDD (测试驱动开发)

如果你创建了一个Mock那么你就可以在service接口创建之前写Service Tests了, 这样你就能在开发过程中把测试添加到你的自动化测试环境中了。换句话说, 模拟使你能够使用测试驱动开发。

- 团队可以并行工作

这类似于上面的那点; 为不存在的代码创建测试。但前面讲的是开发人员编写测试程序, 这里说的是测试团队来创建。当还没有任何东西要测的时候测试团队如何来创建测试呢? 模拟并针对模拟测试! 这意味着当service接口需要测试时, 实际上QA团队已经有了一套完整的测试组件; 没有出现一个团队等待另一个团队完成的情况。这使得模拟的效益型尤为突出了。

- 你可以创建一个验证或者演示程序。
- 为无法访问的资源编写测试

这个好处不属于实际效益的一种, 而是作为一个必要时的“救生圈”。有没有遇到这样的情况? 当你想要测试一个service接口, 但service需要经过防火墙访问, 防火墙不能为你打开或者你需要认证才能访问。遇到这样情况时, 你可以在你能访问的地方使用MockService替代, 这就是一个“救生圈”功能。

- Mock 可以分发给用户
- 隔离系统

知道什么是mock测试后，那么我们就来认识一下mock框架---Mockito。

Mockito区别于其他模拟框架的地方主要是允许开发者在没有建立“预期”时验证被测系统的行为。

Mockito相关教程：

- 5分钟了解Mockito (<http://liuzhijun.iteye.com/blog/1512780>)
- Mockito：一个强大的用于Java开发的模拟测试框架 (<http://blog.csdn.net/zhoudaxia/article/details/33056093>)
- 学习Mocktio - 利用ArgumentCaptor（参数捕获器）捕获方法参数进行验证 (<http://hotdog.iteye.com/blog/916364>)
- 使用Mockito进行单元测试【2】—— stub 和 高级特性 (<http://qiuguo0205.iteye.com/blog/1456528>)

（与此同时推荐一个东西，SpringOckito (<https://github.com/springockito/springockito>)，不过已经2年没更新了。）

mockito入门实例

Maven依赖：

Xml代码

```
<dependencies>
<dependency>
<groupId>org.mockito</groupId>
<artifactId>mockito-all</artifactId>
<version>1.8.5</version>
<scope>test</scope>
</dependency>
</dependencies>
```

首先，需要在@Before注解的setUp()中进行初始化（下面这个是个测试类的基类）

Java代码

```
public abstract class MockitoBasedTest {
    @Before
    public void setUp() throws Exception {
        // 初始化测试用例类中由Mockito的注解标注的所有模拟对象
        MockitoAnnotations.initMocks(this);
    }
}
```

Java代码

```
import static org.mockito.Mockito.*;
import java.util.List;
import org.junit.Assert;
import org.junit.Test;
public class SimpleTest {

    @Test
    public void simpleTest(){

        //创建mock对象，参数可以是类，也可以是接口
        List<String> list = mock(List.class);

        //设置方法的预期返回值
        when(list.get(0)).thenReturn("helloworld");

        String result = list.get(0);

        //验证方法调用(是否调用了get(0))
        verify(list).get(0);

        //junit测试
        Assert.assertEquals("helloworld", result);
    }
}
```

创建mock对象不能对final，Anonymous，primitive类进行mock。

可对方法设定返回异常

Java代码

```
when(list.get(1)).thenThrow(new RuntimeException("test excpetion"));
```

stubbing另一种语法(设置预期值的方法), 可读性不如前者

Java代码

```
doReturn("secondhello").when(list).get(1);
```

没有返回值的void方法与其设定(支持迭代风格, 第一次调用donothing,第二次dothrow抛出runtime异常)

Java代码

```
doNothing().doThrow(new RuntimeException("void exception")).when(list).clear();  
list.clear();  
list.clear();  
verify(list,times(2)).clear();
```

参数匹配器(Argument Matcher)

Matchers类内加你有很多参数匹配器 anyInt、anyString、anyMap.....Mockito类继承于Matchers,Stubbing时使用内建参数匹配器, 下例:

Java代码

```
@Test  
public void argumentMatcherTest(){  
    List<String> list = mock(List.class);  
  
    when(list.get(anyInt())).thenReturn("hello","world");  
  
    String result = list.get(0)+list.get(1);  
  
    verify(list,times(2)).get(anyInt());  
  
    Assert.assertEquals("helloworld", result);  
}
```

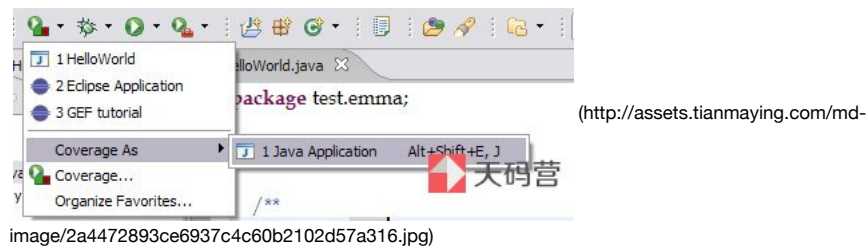
需要注意的是: 如果使用参数匹配器, 那么所有的参数都要使用参数匹配器, 不管是stubbing还是verify的时候都一样。

EclEmma

在众多的Java覆盖率测试工具中, 开源的Emma是最著名的一个, 而EclEmma相当于是它在Eclipse上的图形化界面插件。它使用简单, 结果直观。

安装很简单, 打开Eclipse, 点击Help → Install New Software →输入update.eclEmma.org, 安装软件即可。

首先, 我们可以建立一个HelloWorld类, 然后通过Coverage来运行它。



执行完毕之后, 我们正在编辑 HelloWorld.java 的窗口将会变成如下所示:

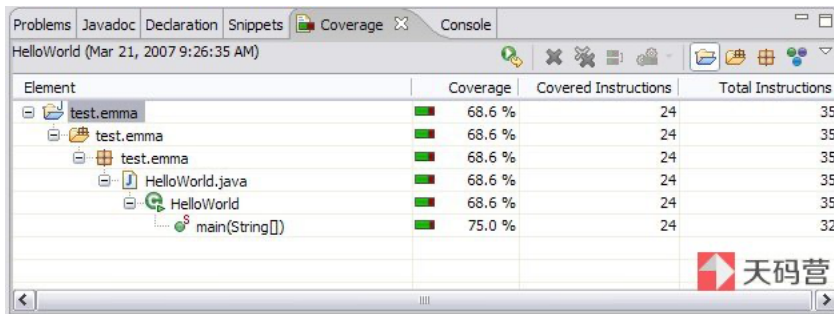


(http://assets.tianmaying.com/md-

image/1c3164c16a49adbe29552606734222ad.jpg)

EclEmma 用不同的色彩标示了源代码的测试情况。其中，绿色的行表示该行代码被完整的执行，红色部分表示该行代码根本没有被执行，而黄色的行表明该行代码部分被执行。黄色的行通常出现在单行代码包含分支的情况。

EclEmma 还提供了单独的视图来统计程序的覆盖测试率。可以选择行覆盖(Line)，分支(Branch)覆盖等多种覆盖率检测标准。



| Element | Coverage | Covered Instructions | Total Instructions |
|-----------------|----------|----------------------|--------------------|
| test.emma | 68.6 % | 24 | 35 |
| test.emma | 68.6 % | 24 | 35 |
| test.emma | 68.6 % | 24 | 35 |
| HelloWorld.java | 68.6 % | 24 | 35 |
| HelloWorld | 68.6 % | 24 | 35 |
| main(String[]) | 75.0 % | 24 | 32 |

(http://assets.tianmaying.com/md-image/f607b39f6c2d868ecda9ab298aaa78ac.jpg)

(更多资料，请参考： 这里 (http://www.cnblogs.com/lpshou/p/3719081.html)， 这里 (http://blog.sina.com.cn/s/blog_4adf62ab0101g68p.html)， 和 这里 (http://www.ibm.com/developerworks/cn/java/j-lo-eclEmma/))。

Spring与单元测试

首先在maven中加载以下的库，尤其是第三个。

[库] junit (http://www.mvnrepository.com/artifact/junit/junit) : 4.12

[库] mockito-core (http://www.mvnrepository.com/artifact/org.mockito/mockito-core) : 1.10.19

[库] spring-boot-starter-test (http://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-test/1.3.1.RELEASE) : 1.3.1

Pom.xml (节选)

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
</dependency>
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
</dependency>
```

接下来，如果你要用到一些Spring自带的注解，比如 @Autowired 的话，最好是在测试类的基类中，加入如下注解，这样会使得测试时先将SpringBoot运行起来。

```
@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration
@SpringApplicationConfiguration(classes = Application.class)
```

接下来需要在@Before注解的setUp()中进行初始化

```
@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration
@SpringApplicationConfiguration(classes = Application.class)
public abstract class MockitoBasedTest {
    @Before
    public void setUp() throws Exception {
        // 初始化测试用例类中由Mockito的注解标注的所有模拟对象
        MockitoAnnotations.initMocks(this);
    }
}
```

由于Eclipse对于 import static 的支持很差，你可能还需要记得加上

```
import static org.junit.Assert.*;
import static org.mockito.Mockito.*;
```

接下来我们为每个类创建测试用例，在比如Service的一个类上面右键-新建-JUnit Test Case，注意要把测试类的目录改到 src/test/java。





至于其他的部分，与上文中提到的mockito的测试步骤基本相同。至于涉及到@Autowired这种，涉及Spring框架的注解而导致测试无法顺利进行的问题，请看下一节的讲解。

相关链接：

- SpringBoot官方文档#测试 (<http://docs.spring.io/spring-boot/docs/1.3.2.BUILD-SNAPSHOT/reference/htmlsingle/#boot-features-testing>)
- eclipse下SpringBoot开发和测试 (<http://somefuture.iteye.com/blog/2247207>)
- 使用 Spring 进行单元测试 (<http://blog.jobbole.com/40740/>)
- 使用SpringJUnit4ClassRunner进行单元测试 (<http://my.oschina.net/anomynous/blog/168999>)
- 如何将mock的类自动注入到待测类 (<http://www.cnblogs.com/mabaishui/p/4029237.html>)
- spring中使用mockito (<http://www.cnblogs.com/syxchina/p/4150879.html>)
- Spring中如何使用Mockito做单元测试 (<http://blog.csdn.net/fireofjava/article/details/8687128>)

测试中遇到的问题和解决办法

1) 被测试类、测试类的覆盖率不同

| | | | | | |
|---|-----------------------|---|---------|---|--|
|  | WechatServiceImpl |  | 4.2 % | 1 | (http://assets.tianmaying.com/md-image/ff294516348d869c3092b3333ab0dee1.png) |
|  | WechatServiceImplTest |  | 100.0 % | 2 | |

我们以 WeChatServiceImpl 类和它的测试类为例，WeChatServiceImpl的代码不变，为了达到上图的效果，我们把测试函数中内容删掉：

```
@Test
public void testIsOutOfTime() {
    ;
}
```

因此可以看出，

- 100%是代表测试函数的每一行都成功执行了，比如我只输入一个分号；
- 但是，4.2%才代表的是被测函数的实际覆盖率。

所以不要被测试类覆盖率100%骗了~



(http://tmy-assets.oss-cn-beijing.aliyuncs.com/img/emoji/stuck_out_tongue.png)

2) 被测类中@Autowired注解，如何控制其中Repository返回值

```
public class GameHelper {
    @Autowired
    private PointRepository pointRepository;

    public boolean checkLineItem(final Line line) {
        Point fromPoint = pointRepository.findById(line.getFromPointId()); //如何控制这个repository的返回值
        Point toPoint = pointRepository.findById(line.getToPointId());
        return fromPoint.getId().equals(toPoint.getId()); //简化了原函数
    }
    ...
}
```

因为在目前的单元测试中，Spring一个很特殊的注解是 @Autowired。（@Autowired 可以对成员变量、方法和构造函数进行标注，来完成自动装配的工作。）

如果我们要写个testCheckLineItem()函数的话，我们怎么控制fromPoint和toPoint呢？

因为不能改变被测类，因此我曾经尝试在测试类中使用过以下方法：

- 在测试函数中，使用new PointRepository().test()
- 对pointRepository使用@Mock, @Spy, @InjectMocks等
- 对pointRepository加@Autowired注解，然后发现注解无效。于是在所有测试类的基类中，增加如下注解以启用@Autowired，但是依然有问题。（不过如果想使用@Autowired一类的注解，下面这个代码是必须加的。）

```
@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration
@SpringApplicationConfiguration(classes = Application.class)
```

- 后来与@Autowired一起加了@Spy也不行！！
- 以及对@Mock, @Spy, @InjectMocks的各种使用组合...

最后，经过朋友的提示以及查找，我查到了这个问题的英文解释 (<http://stackoverflow.com/questions/25893247/difference-between-injectmocks-and-autowired-usage-in-mockito>) 和中文的解答 (<http://www.cnblogs.com/mabaishui/p/4029237.html>)。

正确答案是：对被测类中 @Autowired 的对象，用 @Mocks 标注；对被测类自己，用 @InjectMocks 标注。代码如下：

```
public class GameHelperTest {
    @Mock
    private PointRepository pointRepository;

    @InjectMocks
    private GameHelper gameHelper; //pointRepository作为mock对象被注入到gameHelper中，gameHelper其他成员
    public void testCheckLineItem() {
        Line line = new Line(***);
        when(pointRepository.findById(123L)).thenReturn(new Point(***));
        when(pointRepository.findById(456L)).thenReturn(new Point(***));
        assertTrue(gameHelper.checkLineItem(line));
    }
    ...
}
```

至于原因，我们回到mockito的 官方文档

(http://site.mockito.org/mockito/docs/current/org/mockito/Mockito.html#injectmocks_annotation) 中去看关于@InjectMocks的解释。

@InjectMocks - injects mock or spy fields into tested object automatically.

换言之，被 `@Mock` 标注的对象会自动注入到被 `@InjectMocks` 标注的对象中。比如在本例中，`GameHelper` 中的成员变量 `pointRepository`（的函数），就会被我们用在测试用例中改写过返回值的 `pointRepository` 对象替换掉。

另外，经测试，`thenReturn` 返回的是对象引用而不是深复制了对象本身（所以可以减少写 `thenReturn()` 的次数）。

3) 被测函数调用被测类其他函数，怎么控制返回值？

比如在 `CreateGameServiceImpl` 这个类中，有这样一段函数

```
public class CreateGameServiceImpl implements CreateGameService {

    ...//省略成员变量

    public FullGame createGame(String name, Long creatorId, List<Point> points, List<Selection> sel
        Game gameItem = createBlankGame(name, creatorId);    //createBlankGame()为CreateGameService
```

那么，如果我还没实现 `createBlankGame()`，我在测试函数里应该怎么控制它呢？这次用2) 中的方法 `@Mock + @InjectMocks` 就不行了，因为他们属于同一个类。

（这个问题@Xander 觉得应该实现了被调用的函数才好，但是既然mock的存在很多时候是为了在函数都没实现的情况下编写测试，因此我觉得继续研究。）

后来自己通过查阅**官方的文档 (<http://site.mockito.org/mockito/docs/current/org/mockito/Mockito.html#spy>)，解决办法**是使用 `spy()` 命令，结合 `doReturn()`：

```
public class CreateGameServiceImplTest {
    //这部分不需要改。省略其他成员变量
    @Mock
    private GameHelper gameHelper;

    @InjectMocks
    CreateGameServiceImpl serviceimpl;

    @Test
    public void testCreateGameStringLongList0fPointList0fSelectionList0fLine() {
        serviceimpl = spy(serviceimpl); //将serviceimpl部分mock化
        doReturn(***).when(serviceimpl).createBlankGame(a, b); //这里必须用doReturn()而不能是when().t
        ...
    }
}
```

原因我们在最后解释。

首先我们来看文档中对于 `Spy()` 的解释：

You can create spies of real objects. When you use the spy then the methods are called (unless a method was stubbed).

Spying on real objects can be associated with "partial mocking" concept. (重点是，spy与"部分mock"相关。)

对于 `Spy`，官方有个Sample：

```
List list = new LinkedList();
List spy = spy(list);

//optionally, you can stub out some methods:
when(spy.size()).thenReturn(100);

//using the spy calls real methods
spy.add("one");
spy.add("two");

//prints "one" - 这个函数还是真实的
System.out.println(spy.get(0));

//100 is printed - size()函数被替换了
System.out.println(spy.size());
```

通俗来讲，在我个人理解，`Spy()` 可以使一个对象的一部分方法被用户替换。

在我们的例子中，`CreateGameServiceImpl` 中的函数 `createGame()` 调用了 `createBlankGame()`，而后者可能是未实现的。

但是此时 CreateGameServiceImpl 类的注解是 @InjectMocks 而不是 @Mock，只能接收@Mock对象的注入，而自己的方法无法被mock(stub)。

因此我们通过 spy()，将CreateGameServiceImpl部分mock化，从而将createBlankGame()函数替换掉。

不过这里如果遇到private的被调函数就没办法了。

覆盖率

对于单元测试，一个重要的衡量指标就是覆盖率。

覆盖率分为：

行覆盖(Line Coverage，又叫段覆盖/语句覆盖(Statement Coverage)等)

分支覆盖(Branch Coverage，又叫判定覆盖Decision Coverage等)

条件覆盖(Condition Coverage)

路径覆盖(Path Coverage)等等

据了解 (<http://blog.csdn.net/quicknet/article/details/5549902>)，所有这些覆盖中行覆盖 (Line coverage) 是最简单的，也是最常用的、最有效的覆盖率。

在EclEmma中可以选择任意一种覆盖率，以下是我的项目中行覆盖率的截图。

3. 指令覆盖(Instruction Coverage)，方法覆盖(Method Coverage)：均为100%，就不截图了。

*额外工作：实现任意两个对象比较

由于单元测试常常需要用到 assertEquals() 方法，而对于很多自定义的数据结构（比如Point.java）要重写equals()方法，否则调用equals()只会比较两个对象的引用，这带来非常多的麻烦事。

鉴于Web应用中使用的自定义的数据结构(Model)通常是JavaBean规范的（这些类的成员属性通常是Java的基本数据类型或String.Collection等常见类型），因此我希望通过只对比两个对象的成员变量的变量类型、变量名、变量的值（如果是集合和数组就深入进去判断），来判断两个对象是否“相等”。

我查了好久，除了发现貌似还真没人写，只有 apache.commons.beanutils 和 apache.commons.collections.comparator 有类似的方法，但是看了他们的源码觉得跟我要的不是一个东西。

这个工具类，因为有些细节上写起来很困难，大概写了好几个小时吧，打算以后如果真的没人做这个，就把它做成一个开源的小工具。

在这里我同时提供了两种方式，第一种比较取巧，直接比较两个对象对应的JSON字符串，这种方法很方便，不容易出错，但是可能适用范围上略小一点。

我使用的是JackJson的库（其他也可以），代码如下：

```
static public boolean compareByJson(Object a,Object b){
    try {
        ObjectMapper objectMapper = new ObjectMapper();
        String jsona =objectMapper.writeValueAsString(a);
        String jsonb =objectMapper.writeValueAsString(b);
        System.out.println(jsona);
        System.out.println(jsonb);
        return jsona.equals(jsonb);
    } catch (IOException e) {
        e.printStackTrace();
        return false;
    }
}
```

第二种，则是通过Java的反射机制，通过 getClass(),getDeclaredFields(),setAccessible(true) 等方法来取得任意对象的成员变量，按顺序分析两者中的两个变量的变量名、变量类型、变量值是否相等，是否是重写了equals()方法的常见类型、是否是集合等方面来比较和判断，也借用了LeetCode上一道难度很低的题 (<https://leetcode.com/problems/same-tree/>) 的算法，代码如下（某处仍有bug）：


```
static public boolean compare(Object obj_a, Object obj_b) {
    if(obj_a == null && obj_b == null)
        return true;
    else if (obj_a == null || obj_b == null)
        return false;
    else {
        Field[] fields_a = obj_a.getClass().getDeclaredFields();
        Field[] fields_b = obj_b.getClass().getDeclaredFields();
        if (fields_a.length != fields_b.length)
            return false;
        else for (int i = 0; i < fields_a.length; i++) {
            fields_a[i].setAccessible(true);
            fields_b[i].setAccessible(true);
            Object obj_a_innerobj_i = null, obj_b_innerobj_i = null;
            try {
                obj_a_innerobj_i = fields_a[i].get(obj_a);
                obj_b_innerobj_i = fields_b[i].get(obj_b);
            } catch (IllegalArgumentException e) {
                e.printStackTrace();
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            }
            if (fields_a[i].getName() != fields_b[i].getName())
                return false;
            else if (!fields_a[i].getGenericType().equals(fields_b[i].getGenericType()))
                return false;
            else if (!SupportedClassesList.contains(obj_a_innerobj_i.getClass()))
                if (compare(obj_a_innerobj_i, obj_b_innerobj_i) == false)
                    return false;
            else if (obj_a_innerobj_i instanceof Collection){
                //TODO 仍有bug
                if(!(((Collection) obj_a_innerobj_i).containsAll((Collection) obj_b_innerobj_i))&&
                    return false;
                else if (!obj_a_innerobj_i.equals(obj_b_innerobj_i))
                    return false;
            }
        }
        return true;
    }
}
```

相关链接:

- apache commons collections CollectionUtils工具类 (<http://my.oschina.net/u/1995545/blog/363810>)
- java类中获取属性的名称 (<http://blog.csdn.net/yufaw/article/details/7409602>)
- Java中对象的深复制（深克隆）和浅复制（浅克隆）介绍 (<http://www.jb51.net/article/62909.htm>)
- Java 反射获取private属性 (http://blog.sina.com.cn/s/blog_628d4dd10100xvly.html)
- 在JAVA中如何取得一个变量的类型 (<http://bbs.csdn.net/topics/120096245>)
- Jackson的Json转换 (<http://www.example.com>)
- 比较两个bean对象的值是否相等（源码） (http://www.oschina.net/code/snippet_121156_9543)

版权声明

本文由周鸿博 (user/zhbzhbzbzbz)创作，转载需署名作者且注明文章出处

参考代码

要获取本文的参考代码，请访问：<https://www.tianmaying.com/tutorial/JunitForSpringBoot/repo> (/tutorial/JunitForSpringBoot/repo)

相关文章

在Docker容器中运行Spring Boot应用 (/tutorial/spring-b...
基于Spring的QQ第三方登录实现 (/tutorial/OAuth-login-...
基于Spring的微信第三方登录实现 (/tutorial/OAuth-login-...

其他文章

java入门之表达式、语句、块 (/tutorial/java-expression-...
JDBC使用的经典示例 (/tutorial/jdbc-operation)
centos 7 搭建sentry (/tutorial/sentry)

EclEmma

Spring与单元测试

测试中遇到的问题和解决办法

- 1) 被测试类、测试类的覆盖率不同
- 2) 被测类中@Autowired注解，如何...
- 3) 被测函数调用被测类其他函数，怎...

覆盖率

*额外工作：实现任意两个对象比较

Python入门基础课程 (<https://course.tianmaying.com/basic>)

将Python的入门基础知识贯穿在简单易懂的实例中，代码闯关，名师指导，同学帮助，逐步深入，帮助你快...

Java入门基础教程 (<https://course.tianmaying.com/basic>)

将Java的入门基础知识贯穿在简单易懂的实例中，写代码闯关，名师指导，逐步深入，帮助你快速进入Jav...

MyBatis和Spring MVC整合开发 问答网站 (<https://course.tianmaying.com/mvc-mybatis-qa>)

练习基于MyBatis和Spring MVC搭建问答网站，学习Spring/Spring MVC/S...

一起来写网易云音乐Java爬虫 (<https://course.tianmaying.com/163-crawler>)

爬虫是一个非常适合Java实战练手的项目，而且具有实用性。这个训练中，我们将练习如何爬取、解析网易云...

反馈意见 ^

Spring MVC异常处理 (tutorial/exception-handling-in-s...
 搜狐快站快巴士开发快速入门 (tutorial/kuaizhan-startup)
 基于Spring提供支持不同设备的页面 (tutorial/content-fo...
 Group联合创始人又在拍云架构与运维大会的分享 (tutori...
 Spring MVC@RequestMapping 方法所支持的参数类型...
 如何打造一款互联网软件产品 (tutorial/how-to-develop-...
 LeetCode题解 #30 Substring with Concatenation of All ...

向作者提问 (/qas/create?respondent=zhbzbzbzbz&tutorial=d381976d-baca-4e4c-adfa-e4f0cc0185cb)

注册 (/account/register?next=%2Ftutorial%2FJUnitForSpringBoot)

Copyright Tianmaying © 2016 | 京ICP备15006133号-1 | TMY-EDU | 231216939 (<http://shang.qq.com/wpa/qunwpa?idkey=6cbe9f8835aa1aa6c62f4bceb585d415b9a7f710c4e7f0b20e424905cf3f7e27>) | 天码营 (<http://weibo.com/u/5623427244>) | [服务条款 \(/terms-of-service\)](#) | [官方博客 \(/blog\)](#) | [分享 \(/shares\)](#) | [TCode \(/coders\)](#) | [代码 \(/snippets\)](#) | [友情链接 \(/links\)](#)