Mockito & PowerMock之如何使用篇


骊骅 关注
0.365 2018.04.09 22:49:58 字数 1,023 阅读 1,705

为什么要mock

Mock 测试就是在测试过程中，对于某些不容易构造（如 HttpServletRequest 必须在Servlet 容器中才能构造出来）或者不容易获取比较复杂的对象（如 JDBC 中的ResultSet 对象），用一个虚拟的对象（Mock 对象）来创建以便测试的测试方法。

如下使用范畴

- 真实对象具有不可确定的行为，产生不可预测的效果，（如：股票行情，天气预报）
- 真实对象很难被创建的
- 真实对象的某些行为很难被触发
- 真实对象实际上还不存在的（和其他开发小组或者和新的硬件打交道）等等

Mockito

- 官方文档
- 大多 Java Mock 库如 EasyMock 或 JMock 都是 expect-run-verify （期望-运行-验证）方式，而 Mockito 则使用更简单，更直观的方法： 在执行后的互动中提问
- 非 `expect-run-verify` 方式 也意味着，Mockito 无需准备昂贵的前期启动。他们的目标是透明的，让开发人员专注于测试选定的行为。

Mockito使用流程

mock--> stub ---> run --> verify（可选）

mock

创建mock对象有如下几种方法

1、mock

```
//mock creation
 List mockedList = mock(List.class);

 //using mock object
 mockedList.add("one");
 mockedList.clear();
```


骊骅 关注
总资产5 (约0.51元)

Once created, a mock will remember all interactions. Then you can selectively verify whatever interactions you are interested in.

2、spy

```
    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" - the first element of a list
    System.out.println(spy.get(0));

    //size() method was stubbed - 100 is printed
    System.out.println(spy.size());

    //optionally, you can verify
    verify(spy).add("one");
    verify(spy).add("two");
```

注意:spy对象并不是真实的对象

```
public class TestSpy {

    @Test
    public void test(){
        List<String> list = new LinkedList<String>();
        List spy = spy(list);

        spy.add("one");

        doReturn(100).when(spy).size();

        // 返回one
        System.out.println(spy.get(0));
        //返回100
        System.out.println(spy.size());
        //返回0
        System.out.println(list.size());
        //抛异常java.lang.IndexOutOfBoundsException: Index: 0, Size: 0
        System.out.println(list.get(0));
    }
}

```

3、mock 和 spy 的区别

- By default, for all methods that return a value, a `mock` will return either null, a primitive/primitive wrapper value, or an empty collection, as appropriate. For example 0 for an int/Integer and false for a boolean/Boolean.(使用mock生成的对象，所有方法都是被

mock的，除非某个方法被stub了，否则返回值都是默认值)

- You can create spies of real objects. When you use the spy then the real methods are called (unless a method was stubbed).（使用spy生产的spy对象，所有方法都是调用的spy对象的真实方法，直到某个方法被stub后）

4、Shorthand for mocks creation - @Mock annotation

```
public class ArticleManagerTest {

       @Mock private ArticleCalculator calculator;
       @Mock private ArticleDatabase database;
       @Mock private UserProvider userProvider;

       private ArticleManager manager;
}

```

注意：为了是的上面的annotation生效，必须调用下面之一

- MockitoAnnotations.initMocks(testClass);

```
MockitoAnnotations.initMocks(testClass)
```

- use built-in runner: MockitoJUnitRunner

```

@RunWith(MockitoJUnitRunner.StrictStubs.class)
public class ExampleTest {

    @Mockprivate List list;

    @Test
    public void shouldDoSomething() {
        list.add(100);
    }
  }
```

- MockitoRule.

```
public class ExampleTest {

    //Creating new rule with recommended Strictness setting
    @Rule public MockitoRule rule = MockitoJUnit.rule().strictness(Strictness.STRICT_STU

    @Mock
    private List list;

    @Test
    public void shouldDoSomething() {
        list.add(100);
    }
}
```

stub

1、简单例子

```
//You can mock concrete classes, not just interfaces
LinkedList mockedList = mock(LinkedList.class);

//stubbing
when(mockedList.get(0)).thenReturn("first");
when(mockedList.get(1)).thenThrow(new RuntimeException());

//following prints "first"
System.out.println(mockedList.get(0));

//following throws runtime exception
System.out.println(mockedList.get(1));

//following prints "null" because get(999) was not stubbed
System.out.println(mockedList.get(999));

//Although it is possible to verify a stubbed invocation, usually it's just redundant
//If your code cares what get(0) returns, then something else breaks (often even before
//If your code doesn't care what get(0) returns, then it should not be stubbed. Not conv
verify(mockedList).get(0);
```

- Stubbing can be overridden: for example common stubbing can go to fixture setup but the test methods can override it. Please note that overridding stubbing is a potential code smell that points out too much stubbing
- Once stubbed, the method will always return a stubbed value, regardless of how many times it is called.
- Last stubbing is more important - when you stubbed the same method with the same arguments many times. Other words: the order of stubbing matters but it is only meaningful rarely, e.g. when stubbing exactly the same method calls or sometimes when argument matchers are used, etc.

2、Argument matchers

```
//stubbing using built-in anyInt() argument matcher
when(mockedList.get(anyInt())).thenReturn("element");

//stubbing using custom matcher (let's say isValid() returns your own matcher implementa
when(mockedList.contains(argThat(isValid()))).thenReturn("element");

//following prints "element"
System.out.println(mockedList.get(999));

//you can also verify using an argument matcher
verify(mockedList).get(anyInt());

//argument matchers can also be written as Java 8 Lambdas
verify(mockedList).add(argThat(someString -> someString.length() > 5));

```

3、Stubbing void methods with exceptions

```
    doThrow(new RuntimeException()).when(mockedList).clear();

    //following throws RuntimeException:
    mockedList.clear();
```

4、Stubbing consecutive calls

```
when(mock.someMethod("some arg"))
   .thenThrow(new RuntimeException())
   .thenReturn("foo");

 //First call: throws runtime exception:
 mock.someMethod("some arg");

 //Second call: prints "foo"
 System.out.println(mock.someMethod("some arg"));

 //Any consecutive call: prints "foo" as well (last stubbing wins).
 System.out.println(mock.someMethod("some arg"));
```

```
 when(mock.someMethod("some arg"))
   .thenReturn("one", "two", "three");
```

5、Stubbing with callbacks

```
when(mock.someMethod(anyString())).thenAnswer(new Answer() {
     Object answer(InvocationOnMock invocation) {
         Object[] args = invocation.getArguments();
         Object mock = invocation.getMock();
         return "called with arguments: " + args;
     }
 });

 //the following prints "called with arguments: foo"
 System.out.println(mock.someMethod("foo"));

```

verify

1、Verifying exact number of invocations / at least x / never

```
//using mock
 mockedList.add("once");

 mockedList.add("twice");
 mockedList.add("twice");

 mockedList.add("three times");
 mockedList.add("three times");
 mockedList.add("three times");
```

```
//following two verifications work exactly the same - times(1) is used by default
verify(mockedList).add("once");
verify(mockedList, times(1)).add("once");

//exact number of invocations verification
verify(mockedList, times(2)).add("twice");
verify(mockedList, times(3)).add("three times");

//verification using never(). never() is an alias to times(0)
verify(mockedList, never()).add("never happened");

//verification using atLeast()/atMost()
verify(mockedList, atLeastOnce()).add("three times");
verify(mockedList, atLeast(2)).add("three times");
verify(mockedList, atMost(5)).add("three times");
```

2、Verification in order

```
// A. Single mock whose methods must be invoked in a particular order
List singleMock = mock(List.class);

//using a single mock
singleMock.add("was added first");
singleMock.add("was added second");

//create an inOrder verifier for a single mock
InOrder inOrder = inOrder(singleMock);

//following will make sure that add is first called with "was added first, then with "w
inOrder.verify(singleMock).add("was added first");
inOrder.verify(singleMock).add("was added second");
```

Mockito & PowerMock之如何使用篇


```
List secondMock = mock(List.class);

//using mocks
firstMock.add("was called first");
secondMock.add("was called second");

//create inOrder object passing any mocks that need to be verified in order
InOrder inOrder = inOrder(firstMock, secondMock);

//following will make sure that firstMock was called before secondM
inOrder.verify(firstMock).add("was called first");
inOrder.verify(secondMock).add("was called second");

// Oh, and A + B can be mixed together at will
```


3、Making sure interaction(s) never happened on mock

```
//using mocks - only mockOne is interacted
mockOne.add("one");

//ordinary verification
verify(mockOne).add("one");

//verify that method was never called on a mock
verify(mockOne, never()).add("two");
```

```
11  //verify that other mocks were not interacted
12  verifyZeroInteractions(mockTwo, mockThree);
13
```

4、Finding redundant invocations

```
1  //using mocks
2  mockedList.add("one");
3  mockedList.add("two");
4
5  verify(mockedList).add("one");
6
7  //following verification will fail
8  verifyNoMoreInteractions(mockedList);
9
```

partial mocking

spy对象是调用真实方法，mock的doCallRealMethod也是调用真实方法。当我们想partial mocking的时候，选择依据是根据要stub的方法的多少：

- spy（全部真实除了stub）
- mock.doCallRealMethod（全部不真实 除了stub）

doReturn when && when thenReturn

- `when thenReturn` 会真实调用函数,再把结果改变
- `doReturn when` 不会真的去调用函数，直接把结果改变
- 举例子

```
1  public class Jerry {
2
3      public boolean go() {
4          System.out.println("I say go go go!!");
5          return true;
6      }
7  }
8
```

```
1  public class SpyTest {
2
3      @Test
4      public void test(){
5          Jerry spy = spy(new Jerry());
6
7          when(spy.go()).thenReturn(false);
8
9          //I say go go go!!
10         //false
11         System.out.println(spy.go());
12
13         doReturn(false).when(spy).go();
```

```
14
15          //false
16          System.out.println(spy.go());
17
18      }
19  }
20
```

PowerMock

Mockito 因为可以极大地简化单元测试的书写过程而被许多人应用在自己的工作中，但是Mock工具不可以实现对静态函数、构造函数、私有函数、Final 函数以及系统函数的模拟，但是这些方法往往是我们在大型系统中需要的功能。PowerMock 是在 EasyMock 以及 Mockito 基础上的扩展，通过定制类加载器等技术，PowerMock 实现了之前提到的所有模拟功能，使其成为大型系统上单元测试中的必备工具。

- 类的静态方法

mockStatic

```
1   public class IdGenerator {
2
3       public static long generateNewId() {
4          return 100L;
5       }
6   }
7
```

```
1   public class ClassUnderTest {
2
3       public void methodToTest() {
4
5           final long id = IdGenerator.generateNewId();
6       }
7   }
```

```
1   @RunWith(PowerMockRunner.class)
2   @PrepareForTest(IdGenerator.class)
3   public class MyTestClass {
4
5       @Test
6       public  void demoStaticMethodMocking(){
7           mockStatic(IdGenerator.class);
8           when(IdGenerator.generateNewId()).thenReturn(2L);
9           new ClassUnderTest().methodToTest();
10  //        verifyStatic();
11
12          System.out.println(IdGenerator.generateNewId());
13
14
15
16      }
17  }
18
```

- 构造函数

```
whenNew(File.class).withArguments(direcPath).thenReturn(mockDirectory);
```

```
public class DirectoryStructure {

    public boolean create(String directoryPath) {
        File directory = new File(directoryPath);

        if (directory.exists()) {
            throw new IllegalArgumentException(
                "\"" + directoryPath + "\" already exists.");
        }

        return directory.mkdirs();
    }
}
```

```
@RunWith(PowerMockRunner.class)
@PrepareForTest(DirectoryStructure.class)
public class DirectoryStructureTest {

    @Test
    public void createDirectoryStructureWhenPathDoesntExist() throws Exception {

        final String direcPath = "mocked path";
        File mockDirectory = mock(File.class);

        when(mockDirectory.exists()).thenReturn(false);
        when(mockDirectory.mkdirs()).thenReturn(true);
        whenNew(File.class).withArguments(direcPath).thenReturn(mockDirectory);

        Assert.assertTrue(new DirectoryStructure().create(direcPath));

        verifyNew(File.class).withArguments(direcPath);
    }
}
```

- private & final方法

```
when(underTest, nameOfMethodToMock, input).thenReturn(expected);
```

```
public class PrivatePartialMockingExample {

    public String methodToTest() {
        return methodToMock("input");
    }

    private String methodToMock(String input) {
        return "REAL VALUE = " + input;
    }
}
```

```

@RunWith(PowerMockRunner.class)
@PrepareForTest(PrivatePartialMockingExample.class)
```

```
4  public class PrivatePartialMockingExampleTest {
5
6      @Test
7      public void demoPrivateMethodMocking() throws Exception {
8          final String expected = "TEST VALUE";
9          final String nameOfMethodToMock = "methodToMock";
10         final String input = "input";
11
12         PrivatePartialMockingExample underTest = spy(new PrivatePartialMockingExample());
13
14         when(underTest, nameOfMethodToMock, input).thenReturn(expected);
15         assertEquals(expected,underTest.methodToTest());
16
17         verifyPrivate(underTest).invoke(nameOfMethodToMock,input);
18     }
19 }
20
```

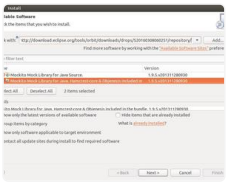
参考文档

http://www.importnew.com/21540.html

https://www.ibm.com/developerworks/cn/java/j-lo-powermock/index.html

[转] 使用强大的 Mockito 测试框架来测试你的代码

1. 预备知识 如果需要往下学习，你需要先理解 Junit 框架中的单元测试。 如果你不熟悉 JUnit，请查看下...



Android单元测试（四）：Mock以及Mockito的使用

几点说明：代码中的 //<== 表示跟上面的相比，这是新增的，或者是修改的代码，不知道怎么样在代码块里面再强调几行...

 邹小创 阅读 8,320 评论 18 赞 38

Android单元测试之Mockito

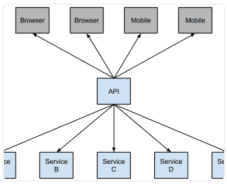
在博客Android单元测试之JUnit4中，我们简单地介绍了：什么是单元测试，为什么要用单元测试，并展示了一个简...

 水木飞雪 阅读 4,251 评论 4 赞 11

Spring Cloud

Spring Cloud为开发人员提供了快速构建分布式系统中一些常见模式的工具（例如配置管理，服务发现，断路器，智...

 卡卡罗2017 阅读 75,252 评论 12 赞 116



Android单元测试（一）

本文介绍了Android单元测试入门所需了解的内容，包括JUnit、Mockito和PowerMock的使用，怎样...

 于卫国 阅读 1,888 评论 0 赞 4