# 验证某个函数是否被执行

验证某个函数A执行的时候，该函数调用的子函数是否会被执行，其中子函数的宿主是一个mock对象，该mock对象传入到待验证的函数A：

**public** **class** **Person** {

**private** String name;

**public** String getName() {

**return** name;

}

**public** void setName(String name) {

**this**.name = name;

}

}

**public** **class** **PersonService** {

**private** Person person;

**public** PersonService(){

**this**(**null**);

}

**public** PersonService(Person person){

**this**.person = person;

}

**public** void changeName(String name){

person.setName(name);

}

}

**public** **class** **TestMockito** {

**@Test**

**public** **final** void testCreatePerson(){

Person p = mock(Person.class);

when(p.getName()).thenReturn("haha");

PersonService ps = **new** PersonService(p);

ps.changeName("haha");

verify(p).setName("haha");

assertEquals(p.getName(), "haha");

}

}

# 设置期望值

**public** **class** **TestStubbing** {

**@Test**

**public** **final** void testNormal(){

List<String> list = mock(ArrayList.class);

String expectedElement = "first";

when(list.get(0)).thenReturn(expectedElement);

assertEquals(list.get(0), expectedElement);

*//没有为999这个元素stubbing, 故list.get(999)返回null*

assertNull(list.get(999));

*//重复stubb,则以第二次为准*

when(list.get(0)).thenReturn("first");

when(list.get(0)).thenReturn("second");

assertEquals(list.get(0), "second");

*//调用同一函数多次，返回不同的值*

when(list.get(0)).thenReturn("first").thenReturn("second");

assertEquals(list.get(0), "first");

assertEquals(list.get(0), "second");

assertEquals(list.get(0), "second");

}

**@Test**(expected=Exception.class)

**public** **final** void testException(){

List<String> list = mock(ArrayList.class);

*//stubb,当调用list.get(1)时，抛出异常*

when(list.get(1)).thenThrow(IndexOutOfBoundsException.class);

list.get(1);

}

}

# 参数匹配

**public** **class** **TestArgumentMatcher** {

**@Test**

**public** **final** void testNormal(){

List<String> list = mock(LinkedList.class);

when(list.get(anyInt())).thenReturn("element");

assertEquals(list.get(10), "element");

when(list.get(any(Integer.class))).thenReturn("element1");

assertEquals(list.get(4), "element1");

}

}

# ****不执行[ 返回值为Void的函数]（doNothing()）****

**public** **class** **TestDoNothing** {

**@Test**

**public** **final** void testNormal(){

Person p = mock(Person.class);

doNothing().when(p).setName("ted");

p.setName("ted");

assertNull(p.getName());

}

}

# ****验证调用的具体次数/最少次数/从未调用****

**public** **class** **TestMethodRunTimes** {

**@Test**

**public** **final** void testNormal(){

List list = mock(ArrayList.class);

when(list.add(any(String.class))).thenReturn(**true**);

list.add("first");

list.add("second");

list.add("second");

list.add("third");

list.add("third");

list.add("third");

*//校验是否执行了一次，若执行了两次，则测试不通过*

*//verify(list).add("second"); // 测试不通过*

verify(list).add("first");

verify(list, times(1)).add("first");

*//指定具体次数*

verify(list, times(2)).add("second");

verify(list, times(3)).add("third");

*//从来没有执行*

verify(list, never()).add("fourth");

*//使用了最多 多少次，最少 多少次*

verify(list, atLeastOnce()).add("second");

verify(list, atLeast(2)).add("third");

verify(list, atLeast(3)).add("third");

}

}

# 验证mock对象的所有方法都没被执行

**public** **class** **TestAllMethodsHasNotRan** {

**@Test**

**public** **final** void testNormal(){

List list = mock(ArrayList.class);

List list1 = mock(ArrayList.class);

verifyZeroInteractions(list);

verifyZeroInteractions(list, list1);

list.add("abc");

*//验证失败*

*//verifyZeroInteractions(list);*

}

}

# 有时我们可以需要调用同一个方法多次，每次需要有不同的返回值（例如迭代），这时我们需要对同一个方法设置不同的返回值/期望值，这个根据调用次数来返回不同的值的特性是Mockito后来添加的。这个其实跟上一篇的第一点提到的多次覆盖期望值一样。

# Void方法抛出异常校验

1. @Test
2. **public** **void** testMockito12() {
3. LinkedList mockedList = mock(LinkedList.**class**);
4. doThrow(**new** RuntimeException()).when(mockedList).clear();
6. // 下面调用会抛异常
7. mockedList.clear();
8. }