# 总结单元测试Mock使用方法

# 更新记录

|  |  |
| --- | --- |
| **版本** | **更新内容** |
| 1.0 | 初始化 |
| 2.0 | 新增1.5、1.6、1.7 |

目录

[总结单元测试Mock使用方法 1](#_Toc15650719)

[更新记录 2](#_Toc15650720)

[1、Mock使用 3](#_Toc15650721)

[1.1、Mock 公共方法/Mapper/外调Service 3](#_Toc15650722)

[1.2、Mock静态方法 4](#_Toc15650723)

[1.3、Mcok私有方法 5](#_Toc15650724)

[1.4、多次调用同一个方法（包括循环），每次返回不同的值 6](#_Toc15650725)

[1.5、Mock本身类中的公共方法 6](#_Toc15650726)

[1.6、Mock方法中new出来的对象 10](#_Toc15650727)

[1.7、Mock无返回值（void）的方法 11](#_Toc15650728)

[2、调用私有方法的两种方式 13](#_Toc15650729)

[2.1、使用反射 14](#_Toc15650730)

[2.2、使用PowerMock 15](#_Toc15650731)

[3、PowerMock与Jacoco冲突（覆盖率扫描不到） 16](#_Toc15650732)

[3.1、对于本地生成覆盖率方式： 17](#_Toc15650733)

[3.2、Jenkins集成Sonar、jacoco覆盖率扫描 19](#_Toc15650734)

[4、TestNG使用PowerMock 20](#_Toc15650735)

引用依赖

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>1.10.19 </version>

</dependency>

<!--PowerMock依赖 -->

<dependency>

<groupId>org.powermock</groupId>

<artifactId>powermock-api-mockito</artifactId>

<version>1.6.6</version>

</dependency>

<dependency>

<groupId>org.powermock</groupId>

<artifactId>powermock-module-junit4</artifactId>

<version>1.6.6</version>

</dependency>

## 1、Mock使用

### 1.1、Mock 公共方法/Mapper/外调Service

被测类：

@RunWith(MockitoJUnitRunner.class)  
public class InnerAccBalSumUpdateServiceImplTest {  
  
 @InjectMocks  
 private InnerAccBalSumUpdateServiceImpl innerAccBalSumUpdateService;  
  
 @Mock  
 private ActInifMapper inifMapper;  
  
 @Mock  
 private ActInblMapper inblMapper;  
  
 @Mock  
 private AccOperSerivceImpl accOperSerivce;  
  
 @Test  
 public void test1() throws Exception{  
 ReflectionTestUtils.*setField*(innerAccBalSumUpdateService,"inifMapper",inifMapper);  
 ReflectionTestUtils.*setField*(innerAccBalSumUpdateService,"inblMapper",inblMapper);  
 ReflectionTestUtils.*setField*(innerAccBalSumUpdateService,"accOperSerivce",accOperSerivce);  
 ActIndtSumDO sumDO = new ActIndtSumDO();  
 sumDO.setAcOrg("test");  
 sumDO.setAcNo("test");  
 sumDO.setCapTyp("test");  
 sumDO.setCcy("test");  
 sumDO.setTotCrAmt(new BigDecimal(1));  
 sumDO.setTotDrAmt(new BigDecimal(1));  
 sumDO.setAcDt("20190101");  
 GDAEntity gda =new GDAEntity();  
 gda.setJrnNo("test");  
 InnerAccBalUpdateRspCO rspCO = new InnerAccBalUpdateRspCO();  
 Mockito.*when*(inifMapper.selectByPrimaryKey(Mockito.*anyString*(),Mockito.*anyString*())).thenReturn(null);  
 InnerAccBalUpdateRspCO rspCO1=innerAccBalSumUpdateService.acBalUpdate(sumDO,gda,rspCO);  
 Assert.*assertTrue*(rspCO1.getGda().getMsgCd().equals("ACTP2038"));  
  
 }

### 1.2、Mock静态方法

**Mock静态方法时，需要使用PowerMock**

被测类：

使用PowerMock

@RunWith(PowerMockRunner.class)  
@PrepareForTest({YGParamMain.class})  
@PowerMockIgnore("javax.management.\*")  
public class GetAccfServiceImplTest {  
  
 @InjectMocks  
 GetAccfServiceImpl getAccfService;  
  
 @Spy  
 YGParamMain ygParamMain=new YGParamMain();  
 @Before  
 public void setUp(){  
 MockitoAnnotations.*initMocks*(this);  
 PowerMockito.*mockStatic*(YGParamMain.class);  
 }  
  
 @Test  
 public void Test01() throws Exception{  
 YGJSONObject rules=null;  
 PowerMockito.*when*(YGParamMain.*getInstance*()).thenReturn(ygParamMain);  
 PowerMockito.*when*(ygParamMain.getParamObject(Mockito.*anyString*(),Mockito.*anyString*())).thenReturn(rules);  
 String acOrg="";  
 String busTyp="";  
 String prdNo="";  
 String txTyp="";  
 String acSceCd="";  
 String aeNo="";  
 Logger logger=YGLogger.*getLogger*();  
 ActAccfDO actAccfDO=getAccfService.queryAccfByRam(acOrg, busTyp, prdNo, txTyp, acSceCd, aeNo, logger);  
 Assert.*assertEquals*(actAccfDO,null);  
  
 }

### 1.3、Mcok私有方法

**Mock私有方法时，需要使用PowerMock**

被测类：

@RunWith(PowerMockRunner.class)  
@PrepareForTest({CheckGlServiceImpl.class})  
@PowerMockIgnore("javax.management.\*")  
public class CheckGlServiceImplTest extends YGTestCase03 {  
  
 @InjectMocks  
 CheckGlServiceImpl checkGlService;  
  
 @Test  
 public void updMainAcBalTest03() throws Exception{  
 ActBatchRspBO rspBO=new ActBatchRspBO();  
 rspBO.setSucc();  
 CheckGlServiceImpl spy = PowerMockito.*spy*(checkGlService);  
 PowerMockito.*doReturn*(rspBO).when(spy, "updMainAcBal",*any*(), *any*());  
 PowerMockito.*doReturn*(rspBO).when(spy, "updInnerAcBal",*any*(), *any*());  
 PowerMockito.*doReturn*(rspBO).when(spy, "checkGltp",*any*(), *any*());  
 PowerMockito.*doReturn*(rspBO).when(spy, "checkBal",*any*(), *any*());  
 ActBatchReqBO actBatchReqBO=new ActBatchReqBO();  
 ActBatchRspBO actBatchRspBO=new ActBatchRspBO();  
 ActBatchRspBO result=spy.checkGl(actBatchReqBO,actBatchRspBO);  
 System.*out*.println(result.getGda().getMsgCd());  
  
 }

### 1.4、多次调用同一个方法（包括循环），每次返回不同的值

Mockito.when(methodCall).thenReturn(1x).thenReturn(2x).thenReturn(nx)

### 1.5、Mock本身类中的公共方法

#### 1.5.0、部分Mock

在这之前先了解一下部分Mock。部分Mock：部分方法实际调用，部分方法模拟。

有两种不同的情况：

Spy：当需要整体执行真正部分，只有少部分方法mock，选用这种方式

AddNumber spy= Mockito.*spy*(addNumber);

doCallRealMethod()：当需要整体Mock，只有少部分方法执行真正部分时，选用这种方式

AddNumber mock= Mockito.*mock*(AddNumber.class);  
Mockito.*doCallRealMethod*().when(mock).add(Mockito.*any*());

**对于想测试类的某个方法（addNum()），但是在该方法中调用了此类中的其他公共方法(add())，此时想将add() Mock掉，有以下几种方式。**

被测类：

public class AddNumber {  
  
 public boolean addNum() throws Exception{  
 AddCO addCO=new AddCO();  
 String a=add(addCO);  
 if(addCO.isSuccess()){  
 return true;  
 }  
 System.*out*.println("add返回："a);  
 return false;  
 }  
 public String add(AddCO co) throws Exception{  
 //throw new RuntimeException();  
 System.*out*.println("调用到了");  
 return "1";  
 }  
}

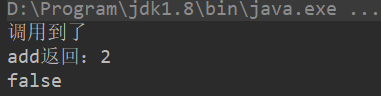
#### 1.5.1、使用spy

##### 方式一：Mockito.when

测试方法：

@InjectMocks  
AddNumber addNumber;  
  
*/\*\*  
 \* 使用此方式进行mock，整个流程（addNum方法）实际执行，也会调用实际方法（add），依赖运行环境  
 \** ***@throws*** *Exception  
 \*/*@Test  
public void test02() throws Exception{  
 AddNumber spy= Mockito.*spy*(addNumber);  
 Mockito.*when*(spy.add(Mockito.*any*())).thenReturn("2");  
 System.*out*.println(spy.addNum());  
}

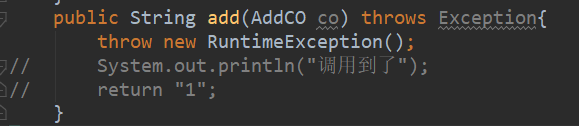
输出结果：



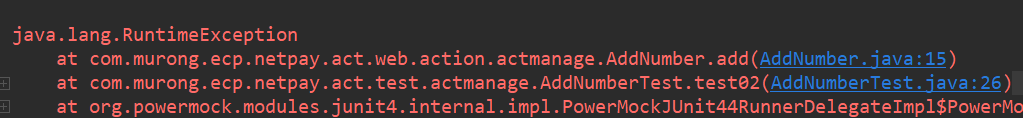
**注意：如图所示，使用spy when方式模拟方法，还是会调用实际的的方法（打印了“调用到了”），只是模拟了返回值。也就是说还是会依赖运行环境。**

**也就有了如下情况：**

**修改add方法为如下图，使其抛出异常，则使用上面的spy方式就会报错。**



**报错：**



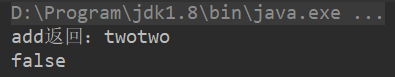
##### 方式二：Mockito.doReturn

**推荐使用此方式mock add方法，之后再单独调用测试add方法**

测试方法：

*/\*\*  
 \* 使用此方式进行mock，整个流程（addNum方法）实际执行，不会调用实际方法（add）  
 \** ***@throws*** *Exception  
 \*/*@Test  
public void testDo() throws Exception{  
 AddNumber spy= Mockito.*spy*(addNumber);  
 Mockito.*doReturn*("twotwo").when(spy).add(Mockito.*any*());  
 System.*out*.println(spy.addNum());  
}

结果：

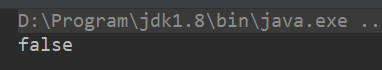


#### 1.5.2、使用Mock

**测试方法：**

*/\*\*  
 \* 使用此方式进行mock，是对整个流程（addNum方法）mock，也不会调用实际方法（add）  
 \** ***@throws*** *Exception  
 \*/*@Test  
public void test04() throws Exception{  
 AddNumber mock= Mockito.*mock*(AddNumber.class);  
 Mockito.*when*(mock.add(Mockito.*any*())).thenReturn("2");  
 System.*out*.println(mock.addNum());  
}

**结果：整个流程被mock（未打印a的值），也未调用实际方法add**

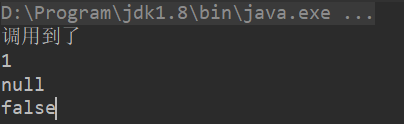


#### 1.5.3、使用doCallRealMethod()方式

**测试方法：**

*/\*\*  
 \* doCallRealMethod() 使用此方式进行mock，整个流程（addNum方法）被mock，根据插桩参数执行实际方法  
 \** ***@throws*** *Exception  
 \*/*@Test  
public void test06() throws Exception{  
 AddNumber mock= Mockito.*mock*(AddNumber.class);  
 AddCO addCO1=new AddCO();  
 AddCO addCO2=new AddCO();  
 Mockito.*doCallRealMethod*().when(mock).add(addCO1);  
 System.*out*.println(mock.add(addCO1)); //步骤①  
 System.*out*.println(mock.add(addCO2)); //步骤②  
 System.*out*.println(mock.addNum()); //步骤③  
}

**对于上面测试方法，有如下结果：**



**解析：**

**“调用到了” 是步骤①实际调用了add方法，执行了实际方法（在add方法中打印出来的）**

**“1” 是步骤①实际调用了add方法返回了1，是步骤①的打印**

**“null” 是步骤②未实际调用add方法，其返回为null，是步骤②的打印**

**“false” 是整个流程都被mock，是步骤③的打印**

### 1.6、Mock方法中new出来的对象

**对于使用@Autowired注入的对象，可使用@Mock方式进行模拟返回，而在方法中直接new的对象，无法使用此方式，可使用PowerMock进行模拟。**

被测试类：

public class AddNumber {  
  
 public boolean addNum(){  
 AddCO addCO=new AddCO();  
 add(addCO);  
 if(addCO.isSuccess()){  
 return true;  
 }  
 return false;  
 }  
 public void add(AddCO co){  
 System.*out*.println("调用到了");  
 }  
}

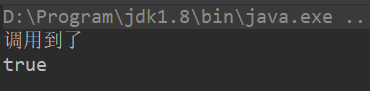
被测试类中new的对象：

public class AddCO {  
 public boolean isSuccess(){  
 return false;  
 }  
}

测试类：

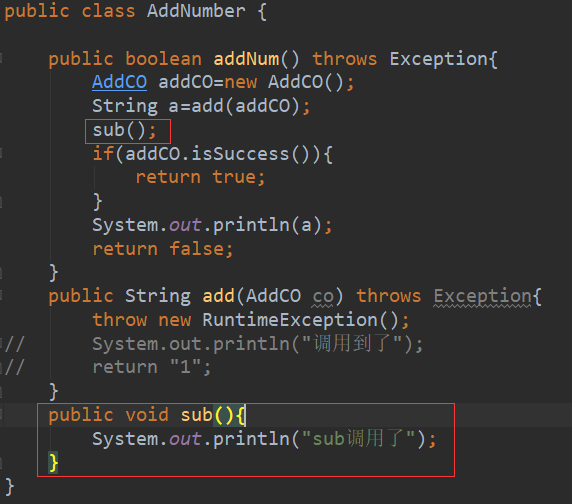
@RunWith(PowerMockRunner.class)  
@PrepareForTest({AddNumber.class})  
@PowerMockIgnore("javax.management.\*")  
public class AddNumberTest {  
  
 @InjectMocks  
 AddNumber addNumber;  
  
 @Test  
 public void test01() throws Exception{  
 AddCO addCO= PowerMockito.*mock*(AddCO.class);  
 PowerMockito.*whenNew*(AddCO.class).withAnyArguments().thenReturn(addCO);  
 PowerMockito.*when*(addCO.isSuccess()).thenReturn(true);  
 System.*out*.println(addNumber.addNum());  
 }  
  
}

输出结果：



### 1.7、Mock无返回值（void）的方法

**此时被测类中新增一个void方法sub，需要将其mock掉， 有两种方式。**

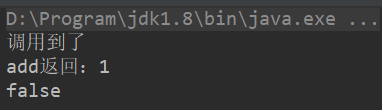


#### 方式一：PowerMock/Mock 的spy方式

**测试方法（两种均可 推荐mock方式）：**

*/\*\*  
 \* PowerMockito spy mock无返回值的公共方法  
 \** ***@throws*** *Exception  
 \*/*@Test  
public void test07() throws Exception{  
 AddNumber spy=PowerMockito.*spy*(addNumber);  
 PowerMockito.*doNothing*().when(spy).sub();  
 System.*out*.println(spy.addNum());  
}  
*/\*\*  
 \* Mockito spy mock无返回值的公共方法  
 \** ***@throws*** *Exception  
 \*/*@Test  
public void test09() throws Exception{  
 AddNumber spy=Mockito.*spy*(addNumber);  
 Mockito.*doNothing*().when(spy).sub();  
 System.*out*.println(spy.addNum());  
}

**结果：未调用到sub方法**

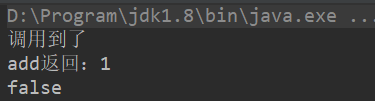


#### 方式二：使用doAnswer

**测试方法：**

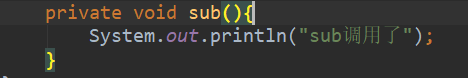
*/\*\*  
 \* Mockito doAnswer mock无返回值的公共方法  
 \** ***@throws*** *Exception  
 \*/*@Test  
public void test10() throws Exception{  
 AddNumber spy = Mockito.*spy*(addNumber);  
 Mockito.*doAnswer*(new Answer<Object>() {  
 public Object answer(InvocationOnMock invocation) {  
 Object[] args = invocation.getArguments();  
 return "called with arguments: " + args;  
 }  
 }).when(spy).sub();  
 System.*out*.println(spy.addNum());  
}

**结果：**



#### 当void方法为private时

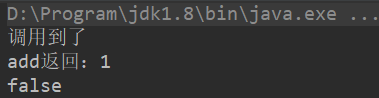
**将sub方法改为私有方法，前面的mock方式都不能用，此时需要使用PowerMock**



**测试方法：**

*/\*\*  
 \* PowerMockito spy mock无返回值的私有方法  
 \** ***@throws*** *Exception  
 \*/*@Test  
public void test09() throws Exception{  
 AddNumber spy=PowerMockito.*spy*(addNumber);  
 PowerMockito.*doNothing*().when(spy,"sub"); //若sub方法有参数，在括号中加逗号和参数即可  
 System.*out*.println(spy.addNum());  
}

**结果：**



## 2、调用私有方法的两种方式

**将私有方法Mcok之后，覆盖率无法扫描到，此时想到直接调用私有方法进行测试。**

### 2.1、使用反射

被测类：同Mock私有方法的被测类

@RunWith(PowerMockRunner.class)  
@PrepareForTest({CheckGlServiceImpl.class})  
@PowerMockIgnore("javax.management.\*")  
public class CheckGlServiceImplTest extends YGTestCase03 {  
  
 @InjectMocks  
 CheckGlServiceImpl checkGlService;  
 @Mock  
 ActInblMapper inblMapper;  
  
 */\*\*  
 \* 调用私有方法 并mock方法中Mapper  
 \** ***@throws*** *Exception  
 \*/* @Test  
 public void sumAcBal02Test() throws Exception{  
 Class<CheckGlServiceImpl> clazz=CheckGlServiceImpl.class;  
 try{  
 Method method = clazz.getDeclaredMethod("sumAcBal", new Class[]{ActBatchReqBO.class,ActBatchRspBO.class,String.class});  
 method.setAccessible(true);// 抑制访问修饰符，使得私有方法变为可以访问的  
 ActBatchReqBO actBatchReqBO=new ActBatchReqBO();  
 ActBatchRspBO actBatchRspBO=new ActBatchRspBO();  
 Mockito.*when*(inblMapper.sumAcBal(Mockito.*anyString*())).thenReturn(null);  
 ActBatchRspBO result = (ActBatchRspBO)method.invoke(checkGlService,actBatchReqBO,actBatchRspBO,"0");  
 System.*out*.println(result.getGda().getMsgCd());  
 }catch (Exception e){  
 e.printStackTrace();  
 Assert.*fail*();  
 }  
 }

### 2.2、使用PowerMock

被测类：同Mock私有方法的被测类

#### 2.2.1、Mock私有方法中的Mapper

@RunWith(PowerMockRunner.class)  
@PrepareForTest({CheckGlServiceImpl.class})  
@PowerMockIgnore("javax.management.\*")  
public class CheckGlServiceImplTest extends YGTestCase03 {  
  
 @InjectMocks  
 CheckGlServiceImpl checkGlService;  
 @Mock  
 ActInblMapper inblMapper;  
  
 */\*\*  
 \* 调用私有方法 并mock方法中Mapper  
 \** ***@throws*** *Exception  
 \*/* @Test  
 public void sumAcBalTest() throws Exception{  
 ActBatchReqBO actBatchReqBO=new ActBatchReqBO();  
 ActBatchRspBO actBatchRspBO=new ActBatchRspBO();  
 Mockito.*when*(inblMapper.sumAcBal(Mockito.*anyString*())).thenReturn(null);  
 Method method = PowerMockito.*method*(CheckGlServiceImpl.class, "sumAcBal", ActBatchReqBO.class,ActBatchRspBO.class,String.class);  
 ActBatchRspBO result = (ActBatchRspBO)method.invoke(checkGlService, actBatchReqBO,actBatchRspBO,"0");  
 //验证是否和预期结果一样  
 System.*out*.println(result.getGda().getMsgCd());  
 }

#### 2.2.2、Mock私有方法中的私有方法

@RunWith(PowerMockRunner.class)  
@PrepareForTest({CheckGlServiceImpl.class})  
@PowerMockIgnore("javax.management.\*")  
public class CheckGlServiceImplTest extends YGTestCase03 {  
  
 @InjectMocks  
 CheckGlServiceImpl checkGlService;  
   
 */\*\*  
 \* 调用私有方法 并Mock其中的私有方法  
 \** ***@throws*** *Exception  
 \*/* @Test  
 public void updMainAcBalTest02() throws Exception{  
 ActBatchRspBO rspBO=new ActBatchRspBO();  
 rspBO.setMsgCd("mock私有方法中的私有方法");  
 CheckGlServiceImpl spy = PowerMockito.*spy*(checkGlService);  
 PowerMockito.*doReturn*(rspBO).when(spy, "sumAcBal",*any*(), *any*(),Mockito.*anyString*());  
 Method method = PowerMockito.*method*(spy.getClass(), "updMainAcBal", ActBatchReqBO.class,ActBatchRspBO.class);  
 ActBatchReqBO actBatchReqBO=new ActBatchReqBO();  
 ActBatchRspBO actBatchRspBO=new ActBatchRspBO();  
 ActBatchRspBO result = (ActBatchRspBO)method.invoke(spy, actBatchReqBO,actBatchRspBO);  
 //验证是否和预期结果一样  
 System.*out*.println(result.getGda().getMsgCd());  
 }

## 3、PowerMock与Jacoco冲突（覆盖率扫描不到）

问题：使用jacoco统计覆盖率时，使用PowerMock时使用如下注解，则CheckGlServiceImpl类的覆盖率为0

@PrepareForTest({CheckGlServiceImpl.class})

解决：使用PowerMock agent方式

依赖：

<!--PowerMock依赖 -->  
<dependency>  
 <groupId>org.powermock</groupId>  
 <artifactId>powermock-api-mockito</artifactId>  
 <version>${powermock.version}</version>  
</dependency>  
<dependency>  
 <groupId>org.powermock</groupId>  
 <artifactId>powermock-module-junit4-rule-agent</artifactId>  
 <version>${powermock.version}</version>  
 <scope>test</scope>  
</dependency>  
<dependency>  
 <groupId>org.powermock</groupId>  
 <artifactId>powermock-module-junit4</artifactId>  
 <version>${powermock.version}</version>  
</dependency>  
<dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.12</version>  
</dependency>

<dependency>  
 <groupId>org.jacoco</groupId>  
 <artifactId>jacoco-maven-plugin</artifactId>  
 <version>${jacoco.version}</version>  
</dependency>

测试方法：去掉@RunWith注解，使用@Rule注解

//@RunWith(PowerMockRunner.class)  
@PrepareForTest({CheckGlServiceImpl.class})  
@PowerMockIgnore("javax.management.\*")  
public class CheckGlServiceImplTest extends YGTestCase03 {  
   
 @Rule  
 public PowerMockRule rule = new PowerMockRule();  
  
 static {  
 PowerMockAgent.*initializeIfNeeded*();  
 }  
 @InjectMocks  
 CheckGlServiceImpl checkGlService;  
 @Mock  
 ActInblMapper inblMapper;

### 3.1、对于本地生成覆盖率方式：

Pom插件配置：

<plugin>  
 <groupId>org.jacoco</groupId>  
 <artifactId>jacoco-maven-plugin</artifactId>  
 <version>${jacoco.version}</version>  
 <executions>  
 <execution>  
 <id>post-unit-test</id>  
 <phase>test</phase>  
 <goals>  
 <goal>report</goal>  
 </goals>  
 <configuration>  
 <dataFile>${project.build.directory}/jacoco.exec</dataFile>  
 </configuration>  
 </execution>  
 </executions>  
</plugin>  
  
<plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-surefire-plugin</artifactId>  
 <version>2.19.1</version>  
 <configuration>  
 <!--java8则需要添加-noverify配置-->  
 <argLine>-noverify ${surefireArgLine}  
 -javaagent:${settings.localRepository}/org/powermock/powermock-module-javaagent/${powermock.version}/powermock-module-javaagent-${powermock.version}.jar  
 </argLine>  
 <systemPropertyVariables>  
 <jacoco-agent.destfile>${project.build.directory}/jacoco.exec</jacoco-agent.destfile>  
 </systemPropertyVariables>  
 <includes>

<!-- 执行的测试类-->  
 <include>\*\*/CheckGlServiceImplTest.java</include>  
 </includes>  
 </configuration>  
</plugin>

<properties>  
 <powermock.version>1.6.6</powermock.version>  
 <jacoco.version>0.8.3</jacoco.version>  
 <surefireArgLine>  
 -javaagent:${settings.localRepository}/org/jacoco/org.jacoco.agent/${jacoco.version}/org.jacoco.agent-${jacoco.version}-runtime.jar=destfile=${project.build.directory}/jacoco.exec  
 </surefireArgLine>  
</properties>

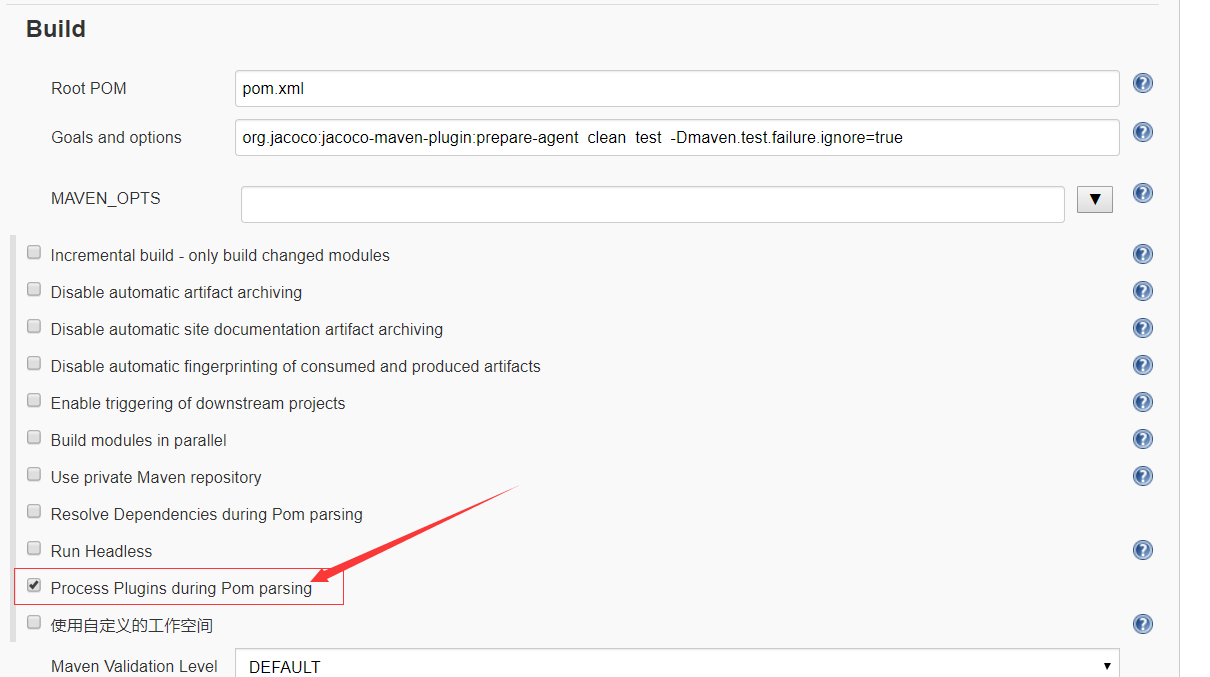
### 3.2、Jenkins集成Sonar、jacoco覆盖率扫描

删除jacoco-maven-plugin配置

<plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-surefire-plugin</artifactId>  
 <version>2.19.1</version>  
 <configuration>  
 <!--java8则需要添加-noverify配置-->  
 <argLine>-noverify ${surefireArgLine}  
 -javaagent:${settings.localRepository}/org/powermock/powermock-module-javaagent/${powermock.version}/powermock-module-javaagent-${powermock.version}.jar  
 </argLine>  
 <systemPropertyVariables>  
 <jacoco-agent.destfile>${project.build.directory}/jacoco.exec</jacoco-agent.destfile>  
 </systemPropertyVariables>  
 <includes>  
 <!-- 执行的测试类-->  
 <include>\*\*/CheckGlServiceImplTest.java</include>  
 </includes>  
 </configuration>  
</plugin>

<properties>  
 <powermock.version>1.6.6</powermock.version>  
 <jacoco.version>0.8.3</jacoco.version>  
 <surefireArgLine>  
 -javaagent:${settings.localRepository}/org/jacoco/org.jacoco.agent/${jacoco.version}/org.jacoco.agent-${jacoco.version}-runtime.jar=destfile=${project.build.directory}/jacoco.exec  
 </surefireArgLine>  
</properties>

并在jenkins任务配置中勾选该选项



## 4、TestNG使用PowerMock

上面讲到的是Junit使用PowerMock的方式。

使用TestNG时，需要继承PowerMockTestCase类并在@PrepareForTest中把这个类本身也加上

