TestNG教程文档

`作者：Alan/阿风`

`该文档参考了大量的网络资源，如果有引用到您的文章，请告知，为其署名`

`该文档仅用于技术分享`

`文档尚有不足之处，请见谅`

> 注意：本教程使用的开发工具是IDEA与Eclipse的操作稍有不同

简介

TestNG，即Testing, ，是一套根据JUnit和NUnit思想而构建的利用注释来强化测试功能的一个测试框架。

TestNG设计涵盖所有类型的测试：单元，功能，端到端，集成等。

前置知识点

- 套件:一个套件包含多个测试
- 测试:一个测试包含多个类
- 类:一个类包含多个测试方法
- 方法：即测试方法
- 组：可以为测试方法分组

`所以这里有几个概念：套件(suite)、测试(test)、类（class）、方法（method）、组（group）`

入门案例

- 导入依赖

```xml
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>
```

- 建立测试类

```java
package com.hzwq.demo;

import org.testng.annotations.Test;

public class MyTestOfTestNG {

    @Test
    public void test(){
        System.out.println("This is first TestNG test!");
    }
}
```

运行:

运行结果:

```
[TestNG] Running:
 // C:\Users\ThinkPad\.IntelliJIdea2018.1\system\temp-testng-customsuite.xml
This is first TestNG test!

===============================================
Default Suite
Total tests run: 1, Failures: 0, Skips: 0
===============================================
```

> `上面的运行结果我们可以看的到，实际运行的是C:\Users\ThinkPad\.IntelliJIdea2018.1\system\temp-testng-customsuite.xml进指定的。`
>
> 查看temp-testng-customsuite.xml：
>
> ```
> <?xml version="1.0" encoding="UTF-8"?>
> <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
> <suite name="Default Suite">
>   <test name="TestNG_Project">
>     <classes>
>       <class name="com.hzwq.demo.MyTestOfTestNG">
>         <methods>
>           <include name="test"/>
>         </methods>
>       </class> <!-- com.hzwq.demo.MyTestOfTestNG -->
>     </classes>
>   </test> <!-- TestNG_Project -->
> </suite> <!-- Default Suite -->
> 
> ```
>
> **所以这个temp-testng-customsuite.xml才是TestNG的实际运行**

运行方式

第一种：test.xml

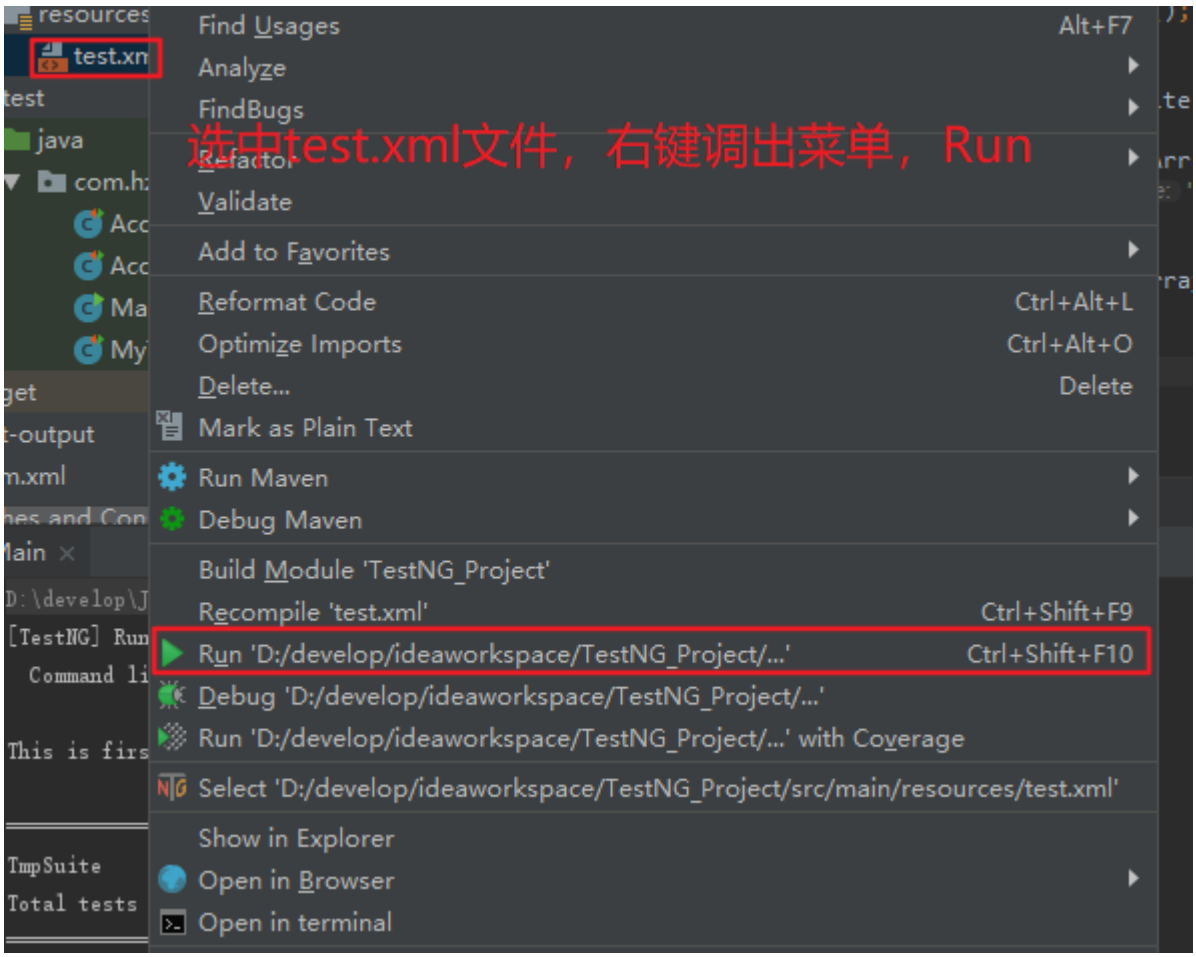
- 详细介绍

```
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name="Suite1" verbose="1" >

    <!--1. 指定类-->
    <test name="Regression1">
        <classes>
            <class name="test.sample.ParameterSample"/>
            <class name="test.sample.ParameterTest"/>
        </classes>
    </test>
```

```
12
13      <!--2. 指定包-->
14      <test name="Regression1"   >
15          <packages>
16              <package name="test.sample" />
17          </packages>
18      </test>
19
20      <!--3. 指定组 和 方法-->
21      <test name="Regression1">
22          <groups>
23              <run>
24                  <exclude name="brokenTests"  />
25                  <include name="checkinTests"  />
26              </run>
27          </groups>
28
29          <classes>
30              <class name="test.IndividualMethodsTest">
31                  <methods>
32                      <include name="testMethod" />
33                  </methods>
34              </class>
35          </classes>
36      </test>
37  </suite>
38
```



第二种：编程式

- 直接指定类的运行

```
package com.hzwq.demo;

import org.testng.TestListenerAdapter;
import org.testng.TestNG;

public class Main {
    public static void main(String[] args) {
        TestListenerAdapter tla = new TestListenerAdapter();
        TestNG testng = new TestNG();
        testng.setTestClasses(new Class[] { MyTestOfTestNG.class
});
        testng.addListener(tla);
        testng.run();
    }
}
```

```
15
```

- 代码配置xml的方式运行

```
    public static void main(String[] args) {
        XmlSuite suite = new XmlSuite();
        suite.setName("TmpSuite");

        XmlTest test = new XmlTest(suite);
        test.setName("TmpTest");
        List<XmlClass> classes = new ArrayList<XmlClass>();
        classes.add(new
XmlClass("com.hzwq.demo.MyTestOfTestNG"));
        test.setXmlClasses(classes) ;

        List<XmlSuite> suites = new ArrayList<XmlSuite>();
        suites.add(suite);
        TestNG tng = new TestNG();
        tng.setXmlSuites(suites);
        tng.run();
    }
```

第三种：傻瓜式

选中类或者方法右键run

注解

@Before类别和@After类别

注解:

注解	简介	
@BeforeSuite	对于套件测试，在此套件中的所有测试运行之前运行。	
@AfterSuite	对于套件测试，在此套件中的所有测试运行之后运行。	
@BeforeTest	对于套件测试，在运行属于`<test>`标签内的类的任何测试方法之前运行	
@AfterTest	于套件测试，在运行属于`<test>`标签内的类的所有测试方法都已运行之后运行。	
@BeforeGroups	在调用属于该组的第一个测试方法之前运行。	
@AfterGroups	在调用属于组的最后一个测试方法之后运行。	
@BeforeClass	在当前类的第一个测试方法之前运行。	
@AfterClass	运行当前类中的所有测试方法之后都运行。	
@BeforeMethod	在每个测试方法之前运行。	
@AfterMethod	在每个测试方法之后运行。	

注解属性：

属性	介绍
alwaysRun	略
dependsOnGroups	先运行依赖的组中所有的方法，后运行改注解指定的测试方法
dependsOnMethods	先运行依赖的方法，后运行改方法
enabled	是否启用该方法（即忽略该方法）
groups	对方法/类进行分组
inheritGroups	继承组，true继承，但必须在class级别上指定组
onlyForGroups	仅在指定的组运行测试，只适用于 @BeforeMethod and @AfterMethod；

生命周期：

- 案例代码

```
public class TestNGAnnotationTest {

    @BeforeSuite
    public void beforeSuite() {
        System.out.println(this.getClass().getName() + "
beforeSuite");
    }

    @AfterSuite
    public void afterSuite() {
        System.out.println(this.getClass().getName() + "
afterSuite");
    }

    @BeforeTest
    public void beforeTest() {
        System.out.println(this.getClass().getName() + "
beforeTest");
    }

    @AfterTest
    public void afterTest() {
        System.out.println(this.getClass().getName() + "
afterTest");
    }

    @BeforeClass
    public void beforeClass() {
        System.out.println(this.getClass().getName() + "
beforeClass");
    }

    @AfterClass
    public void afterClass() {
        System.out.println(this.getClass().getName() + "
afterClass");
    }

    @BeforeMethod
    public void beofreMethod() {
```

```
35          System.out.println(this.getClass().getName() + "
beforeMethod");
36      }
37
38      @AfterMethod
39      public void afterMethod() {
40          System.out.println(this.getClass().getName() + "
afterMethod");
41      }
42
43      @Test
44      public void test1() {
45          System.out.println(this.getClass().getName() + "
test1");
46      }
47
48      @Test
49      public void test2() {
50          System.out.println(this.getClass().getName() + "
test2");
51      }
52
53  }
```

- 案例test.xml

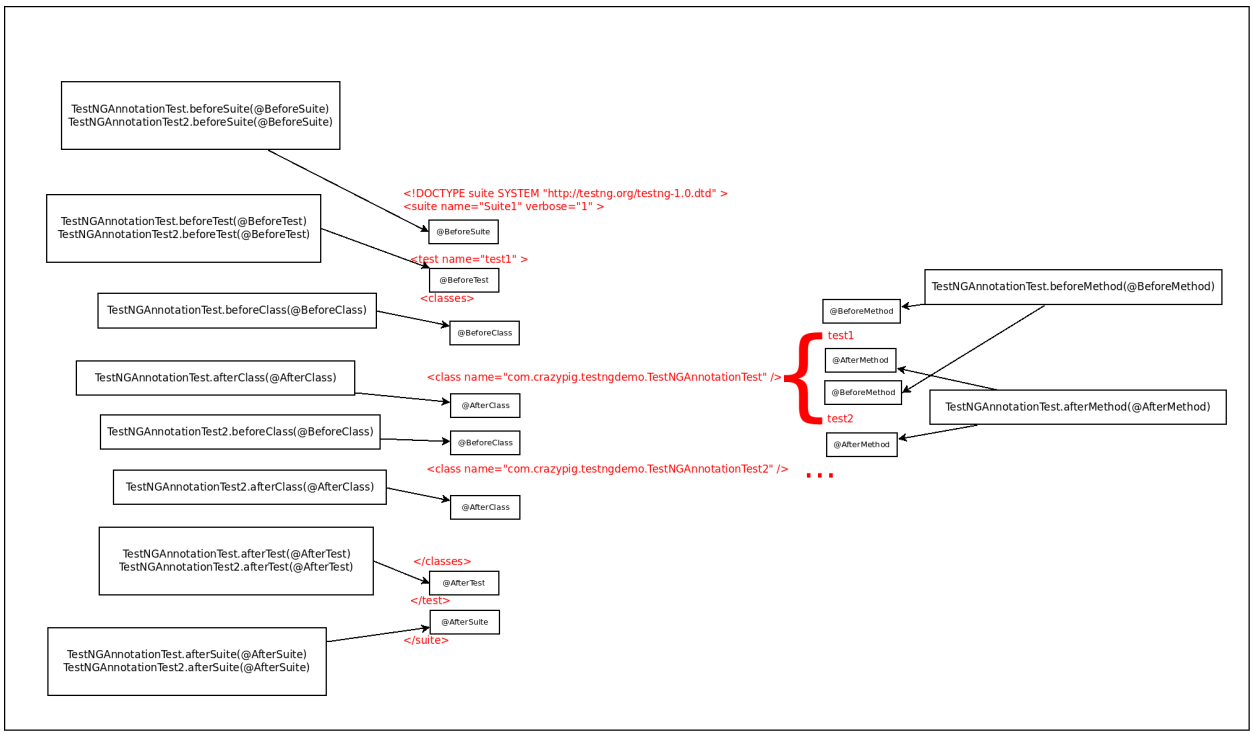
```
1  <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
2  <suite name="Suite1" verbose="1" >
3    <test name="test1" >
4      <classes>
5          <class name="com.crazypig.testngdemo.TestNGAnnotationTest"
   />
6          <class
   name="com.crazypig.testngdemo.TestNGAnnotationTest2" />
7      </classes>
8    </test>
9  </suite>
```

- 结果

```
1  com.crazypig.testngdemo.TestNGAnnotationTest beforeSuite
2  com.crazypig.testngdemo.TestNGAnnotationTest2 beforeSuite
```

```
com.crazypig.testngdemo.TestNGAnnotationTest beforeTest
com.crazypig.testngdemo.TestNGAnnotationTest2 beforeTest
com.crazypig.testngdemo.TestNGAnnotationTest beforeClass
com.crazypig.testngdemo.TestNGAnnotationTest beforeMethod
com.crazypig.testngdemo.TestNGAnnotationTest test1
com.crazypig.testngdemo.TestNGAnnotationTest afterMethod
com.crazypig.testngdemo.TestNGAnnotationTest beforeMethod
com.crazypig.testngdemo.TestNGAnnotationTest test2
com.crazypig.testngdemo.TestNGAnnotationTest afterMethod
com.crazypig.testngdemo.TestNGAnnotationTest afterClass
com.crazypig.testngdemo.TestNGAnnotationTest2 beforeClass
com.crazypig.testngdemo.TestNGAnnotationTest2 beforeMethod
com.crazypig.testngdemo.TestNGAnnotationTest2 test1
com.crazypig.testngdemo.TestNGAnnotationTest2 afterMethod
com.crazypig.testngdemo.TestNGAnnotationTest2 beforeMethod
com.crazypig.testngdemo.TestNGAnnotationTest2 test2
com.crazypig.testngdemo.TestNGAnnotationTest2 afterMethod
com.crazypig.testngdemo.TestNGAnnotationTest2 afterClass
com.crazypig.testngdemo.TestNGAnnotationTest afterTest
com.crazypig.testngdemo.TestNGAnnotationTest2 afterTest
com.crazypig.testngdemo.TestNGAnnotationTest afterSuite
com.crazypig.testngdemo.TestNGAnnotationTest2 afterSuite
```

- 结果图解



应用范围：

> before类别的注解方法里面可以做一些初始化动作，如实例化数据库连接、新建数据库连接池、创建线程池、打开文件流等等。
>
> After类别的注解方法里面可以做一些销毁动作，如释放数据库连接、销毁数据库连接池、销毁线程池或者关闭文件流等等 。

@Test 注解

使用方式

> 加在class上面，则class中的所有方法都作为测试方法
>
> 加在method上面，则该方法作为测试方法

```
package com.hzwq.demo;

import org.testng.annotations.Test;
@Test
public class MyTestOfTestNG {

    public void test(){

        System.out.println("This is first TestNG test!");
    }

    public void test01(){
        System.out.println("test");
    }

    @Test
    public void test02(){

    }
}

```

注解属性

属性	介绍
alwaysRun	如果=`true`,表示即使该测试方法所依赖的前置测试有失败的情况，也要执行
dataProvider	选定传入参数的构造器。
dataProviderClass	确定参数构造器的Class类。(参数构造器首先会在当前测试类里面查找，如果参数构造器不在当前测试类定义，那么必须使用该属性来执行它所在的Class类)
dependsOnGroups	确定依赖的前置测试组别。
dependsOnMethods	确定依赖的前置测试方法。
description	测试方法描述信息。(建议为每个测试方法添加有意义的描述信息，这将会在最后的报告中展示出来)
enabled	默认为`true`，如果指定为`false`，表示不执行该测试方法。
expectedExceptions	指定期待测试方法抛出的异常，多个异常以逗号(,)隔开。
groups	指定该测试方法所属的组，可以指定多个组，以逗号隔开。组测试的用法将在后面文章单独介绍。
invocationCount	指定测试方法需要被调用的次数
invocationTimeOut	每一次调用的超时时间，如果`invocationCount`没有指定，该参数会被忽略。应用场景可以为测试获取数据库连接，超时就认定为失败。单位是毫秒。
priority	指定测试方法的优先级，数值越低，优先级越高，将会优先与其他数值高的测试方法被调用。(注意是针对一个测试类的优先级)
successPercentage	期望成功的百分比
singleThreaded	测试方法为单个线程
timeOut	指定测试方法的优先级，数值越低，优先级越高，将会优先与其他数值高的测试方法被调用。(注意是针对一个测试类的优先级)
threadPoolSize	指定方法的线程池数量

@Parameters

@Parameters 注解用于为测试方法传递参数， 用法如下所示:

```
package com.crazypig.testngdemo;

import org.testng.annotations.Parameters;
import org.testng.annotations.Test;

public class AnnotationParametersTest {

    @Parameters(value = {"param1", "param2"})
    @Test
    public void test(String arg1, String arg2) {
        System.out.println("use @Parameters to fill method arguments : arg 1 = " + arg1 + ", arg2 = " + arg2);
    }

}1234567891011121314
```

testng.xml配置

```
<test name="testAnnotationParameters">
    <parameter name="param1" value="value1"></parameter>
    <parameter name="param2" value="value2"></parameter>
    <classes>
        <class name="com.crazypig.testngdemo.AnnotationParametersTest" />
    </classes>
</test>1234567
```

运行结果:

```
use @Parameters to fill method arguments : arg 1 = value1, arg2 = value2
```

@DataProvider

上面的小结提到`@Parameters`注解可以为测试方法传递参数，但是这种方式参数值需要配置在`testng.xml`里面，灵活性不高。而`@DataProvider`注解同样可以为测试方法传递参数值，并且，它是真正意义上的参数构造器，可以传入多组测试数据对测试方法进行测试。被`@DataProvider`注解的方法，方法返回值必须为`Object[][]`或者`Iterator<Object[]>`。例子如下所示:

```
package com.crazypig.testngdemo;

import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

public class AnnotationDataProviderTest {

    @DataProvider(name="testMethodDataProvider")
    public Object[][] testMethodDataProvider() {

        return new Object[][]{{"value1-1", "value2-1"}, {"value1-2", "value2-2"}, {"value1-3", "value2-3"}};

    }

    @Test(dataProvider="testMethodDataProvider")
    public void test(String arg1, String arg2) {
        System.out.println("use @DataProvider to fill method argument : arg1 = " + arg1 + " , arg2 = " + arg2);
    }

}12345678910111213141516171819 20
```

testng.xml配置:

```
<test name="testDataProvider">
  <classes>
    <class name="com.crazypig.testngdemo.AnnotationDataProviderTest" />
  </classes>
</test>12345
```

运行结果:

```
use @DataProvider to fill method argument : arg1 = value1-1 , arg2 = value2-1
use @DataProvider to fill method argument : arg1 = value1-2 , arg2 = value2-2
use @DataProvider to fill method argument : arg1 = value1-3 , arg2 = value2-3
```

可以注入的参数类型：

Annotation	ITestContext	XmlTest	Method	Object[]	ITestResult
BeforeSuite	Yes	No	No	No	No
BeforeTest	Yes	Yes	No	No	No
BeforeGroups	Yes	Yes	No	No	No
BeforeClass	Yes	Yes	No	No	No
BeforeMethod	Yes	Yes	Yes	Yes	Yes
Test	Yes	No	No	No	No
DataProvider	Yes	No	Yes	No	No
AfterMethod	Yes	Yes	Yes	Yes	Yes
AfterClass	Yes	Yes	No	No	No
AfterGroups	Yes	Yes	No	No	No
AfterTest	Yes	Yes	No	No	No
AfterSuite	Yes	No	No	No	No

参数说明：

1. **ITestContext： 如果一个数据提供者在方法签名中声名了一个ITestContext类型的参数，TestNG就会将当前的测试上下文设置给它，这使得数据提供者能够知道当前测试执行的运行时刻参数。**

```
package roger.testng;

import java.util.Random;

import org.testng.ITestContext;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

/*
```

```
 *  数据提供者在方法签名中声明了一个 ITestContext 类型的参数
 *  testng 会将当前的测试上下文设置给它
 *
 */
public class TestDataProviderITestContext {
    @DataProvider
    public Object[][] randomIntegers(ITestContext context) {
        String[] groups= context.getIncludedGroups();
        int size = 2;
        for (String group : groups) {
            System.out.println("--------------" + group);
            if (group.equals("function-test")) {
                size = 10;
                break;
            }
        }

        Object[][] result = new Object[size][];
        Random r = new Random();
        for (int i = 0; i < size; i++) {
            result[i] = new Object[] {new Integer(r.nextInt())};
        }

        return result;
    }

    // 如果在 unite-test 组中执行，将返回2个随机整数构成数组;
    // 如果在 function-test 组中执行，将返回 10 个随机整数构成数组
    @Test(dataProvider = "randomIntegers", groups = {"unit-
test", "function-test"})
    public void random(Integer n) {
        System.out.println(n);
    }

}
```

2. XmlTest
3. Method
4. Object[]

5. ITestResult

@Factory

在一个方法上面打上`@Factory`注解，表示该方法将返回能够被TestNG测试的测试类。利用了设计模式中的工厂模式。例子如下所示:

```
package com.crazypig.testngdemo;

import org.testng.annotations.Factory;

public class AnnotationFactoryTest {

    @Factory
    public Object[] getSimpleTest() {
        return new Object[]{ new SimpleTest("one"), new
SimpleTest("two")};
    }

}123456789101112
```

```
package com.crazypig.testngdemo;

import org.testng.annotations.Test;

public class SimpleTest {

    private String param;

    public SimpleTest(String param) {
        this.param = param;
    }

    @Test
    public void test() {
        System.out.println("SimpleTest.param = " + param);
    }
}12345678910111213141516171
```

testng.xml配置:

```
<test name="testFactory">
    <classes>
        <class name="com.crazypig.testngdemo.AnnotationFactoryTest"
/>
    </classes>
</test>12345
```

运行结果:

```
SimpleTest.param = one
SimpleTest.param = two
```

@Listeners

不做研究

功能讲解

1. **测试**（Partial tests）
2. **忽略测试/排除测试**（Ignoring tests）

   `@Test(enabled = false)`注释在测试方法上，则会绕过这个未准备好测试的测试用例
3. **依赖测试**（Dependencies）

   > 有时，我们可能需要以特定顺序调用测试用例中的方法，或者可能希望在方法之间共享一些数据和状态。 TestNG支持这种依赖关系，因为它支持在测试方法之间显式依赖的声明。

   TestNG允许指定依赖关系：

   - 在`@Test`注释中使用属性`dependsOnMethods`，或者
   - 在`@Test`注释中使用属性`dependsOnGroups`。

   > 使用`<groups>`标记在`testng.xml`文件中指定分组。 它可以在`<test>`或`<suite>`标签下找到。 `<suite>`标签中指定分组适用于其下的所有`<test>`标签。
4. **分组测试**（Test groups ）

   分组测试是TestNG中的一个新的创新功能，它在JUnit框架中是不存在的。 它允许您将方法调度到适当的部分，并执行复杂的测试方法分组。
5. **套件测试**（Test suites ）

测试套件是用于测试软件程序的行为或一组行为的测试用例的集合。 在TestNG中，我们无法在测试源代码中定义一个套件，但它可以由一个XML文件表示，因为套件是执行的功能。 它还允许灵活配置要运行的测试。 套件可以包含一个或多个测试，并由`<suite>`标记定义。

`<suite>`是`testng.xml`的根标记。 它描述了一个测试套件，它又由几个`<test>`部分组成。

下表列出了`<suite>`接受的所有定义的合法属性。

属性	描述
name	套件的名称，这是一个强制属性。
verbose	运行的级别或详细程度。
parallel	TestNG是否运行不同的线程来运行这个套件。
thread-count	如果启用并行模式(忽略其他方式)，则要使用的线程数。
annotations	在测试中使用的注释类型。
time-out	在本测试中的所有测试方法上使用的默认超时。

6. **期望异常**（Expected Exceptions)

   @Test(expectedExceptions = Exception.class)

7. **超时测试**(*Timeout Test*)

   “超时”表示如果单元测试花费的时间超过指定的毫秒数，那么**TestNG**将会中止它并将其标记为失败。

8. **并行测试**（ Parallelism tests)

> 在TestNG的执行中，测试的级别由上至下可以分为**suite -> test -> class -> method**，箭头的左边元素跟右边元素的关系是一对多的包含关系。
>
> ## uite
>
> 一般情况下，一个testng.xml只包含一个suite。如果想起多个线程执行不同的suite，官方给出的方法是：通过命令行的方式来指定线程池的容量。
>
> ```
> java org.testng.TestNG -suitethreadpoolsize 3 testng1.xml
> testng2.xml testng3.xml
> ```

> 即可通过三个线程来分别执行testng1.xml、testng2.xml、testng3.xml。 实际上这种情况在实际中应用地并不多见，我们的测试用例往往放在一个suite中，如果真需要执行不同的suite，往往也是在不同的环境中去执行，届时也自然而然会做一些其他的配置（如环境变量）更改，会有不同的进程去执行。因此这种方式不多赘述。
>
> ### test, class, method
>
> test，class，method级别的并发，可以通过在testng.xml中的suite tag下设置，如：
>
> ```
> 1 <suite name="Testng Parallel Test" parallel="tests" thread-count="5">
> 2 <suite name="Testng Parallel Test" parallel="classes" thread-count="5">
> 3 <suite name="Testng Parallel Test" parallel="methods" thread-count="5">
> ```
>
> 它们的共同点都是最多起5个线程去同时执行不同的用例。 它们的区别如下：
>
> - tests级别：不同test tag下的用例可以在不同的线程执行，相同test tag下的用例只能在同一个线程中执行。
> - classs级别：不同class tag下的用例可以在不同的线程执行，相同class tag下的用例只能在同一个线程中执行。
> - methods级别：所有用例都可以在不同的线程去执行。
>
> 搞清楚并发的级别非常重要，可以帮我们合理地组织用例，比如将非线程安全的测试类或group统一放到一个test中，这样在并发的同时又可以保证这些类里的用例是单线程执行。也可以根据需要设定class级别的并发，让同一个测试类里的用例在同一个线程中执行。
>
> ## 同一个方法的并发
>
> @Test(enabled=true, dataProvider="testdp", threadPoolSize=5, invocationCount=10)

9. **参数供应**（Parameters with DataProviders）
10. **方法拦截器**（Method Interceptors）
11. **监听**（TestNG Listeners）
    - `IAnnotationTransformer` (doc, javadoc)
    - `IAnnotationTransformer2` (doc, javadoc)

- `IHookable` (doc, javadoc)
- `IInvokedMethodListener` (doc, javadoc)
- `IMethodInterceptor` (doc, javadoc)
- `IReporter` (doc, javadoc)
- `ISuiteListener` (doc, javadoc)
- `ITestListener` (doc, javadoc)

> 添加监听：
>
> - 方式一
>
> ```
> <suite>
>
>   <listeners>
>     <listener class-name="com.example.MyListener" />
>     <listener class-name="com.example.MyMethodInterceptor"
> />
>   </listeners>
>
> ...
>
> ```
>
> - 方式二
>
> ```
> @Listeners({ com.example.MyListener.class,
> com.example.MyMethodInterceptor.class })
> public class MyTest {
>   // ...
> }
>
> ```

12. **日志记录**（Logging Reporters）

```
generateReport(List<XmlSuite> xmlSuites, List<ISuite> suites,
String outputDirectory)
```