# 概述

TestNG 表示下一代（Next Generation的首字母）。它的设计覆盖所有类别的测试：单元、功能、端到端、集成等。

测试套（suite），用一个XML文件表述，包含一个或多个测试（test），<suite>标签定义。

测试（test），用<test>表述，包含一个或多个TestNG类。

TestNG类，一个包含至少一个TestNG注释Java类，<class>标签表述，包含一个或多个测试方法（test method）。

测试方法（test method），一个@Test注解的Java方法。

# 注解

|  |  |  |
| --- | --- | --- |
| **@BeforeSuite**  **@AfterSuite**  **@BeforeTest**  **@AfterTest**  **@BeforeGroups**  **@AfterGroups**  **@BeforeClass**  **@AfterClass**  **@BeforeMethod**  **@AfterMethod** |  | **TestNG类配置信息:**  **注解在TestNG类的超类中的行为**  存在继承关系的TestNG类，@Before注解的方法将按继承链自上而下的顺序执行，@After注解的方法与之相反。 |
|  | alwaysRun |  |
|  | dependsOnGroups |  |
|  | dependsOnMethods |  |
|  | enabled |  |
|  | groups |  |
|  | inheritGroups |  |
| **@DataProvider** |  | **注解方法必须返回Object[][]** |
|  | name | 不指定则同方法名 |
|  | parallel | 默认false |
| **@Factory** |  | **注解方法必须返回 Object[].** |
| **@Listeners** |  |  |
|  | value | org.testng.ITestNGListener |
| **@Parameters** |  |  |
|  | value |  |
| **@Test** |  |  |
|  | alwaysRun |  |
|  | dataProvider |  |
|  | dataProviderClass |  |
|  | dependsOnGroups |  |
|  | dependsOnMethods |  |
|  | description |  |
|  | enabled |  |
|  | expectedExceptions |  |
|  | groups |  |
|  | invocationCount |  |
|  | invocationTimeOut |  |
|  | priority | 低优先级优先 |
|  | successPercentage |  |
|  | singleThreaded | 仅用于类级别 |
|  | timeOut |  |
|  | threadPoolSize |  |

# 使用配置

## 指定类

|  |
| --- |
| <!DOCTYPE suite SYSTEM "<http://testng.org/testng-1.0.dtd>" >    <suite name="Suite1" verbose="1" >  <test name="Nopackage" >  <classes>  <class name="NoPackageTest" />  </classes>  </test>    <test name="Regression1">  <classes>  <class name="test.sample.ParameterSample"/>  <class name="test.sample.ParameterTest"/>  </classes>  </test>  </suite> |

## 指定包

|  |
| --- |
| <!DOCTYPE suite SYSTEM "<http://testng.org/testng-1.0.dtd>" >  <suite name="测试套件" verbose="1" >  <test name="简单测试">  <packages>  <package name="test.sample" />  <package name="test.sample2" />  </packages>  </test>  </suite> |

## 指定方法

|  |
| --- |
| <suite name="测试套件" verbose="1" >  <test name="简单测试">  <classes>  <class name="test.sample.FirstTest">  <methods>  <include name="testCase" />  </methods>  </class>  </classes>  </test>  </suite> |

## 分组

|  |
| --- |
| <suite name="测试套件" verbose="1" >  <test name="简单测试">  <groups>  <run>  <exclude name="异常" /> <!-- 排除不执行的测试用例 -->  <include name="高" /> <!-- 指定执行的测试用例 -->  </run>  </groups>    <classes>  <class name="test.sample.TagTest"/>  </classes>  </test>  </suite> |

### 正则表达式

|  |
| --- |
| <test name="Test1">  <classes>  <class name="example1.Test1">  <methods>  <include name=".\*enabledTestMethod.\*"/>  <exclude name=".\*brokenTestMethod.\*"/>  </methods>  </class>  </classes>  </test> |

### 方法分组

|  |
| --- |
| <test name="Test1">  <classes>  <class name="example1.Test1">  <methods>  <include name=".\*enabledTestMethod.\*"/>  <exclude name=".\*brokenTestMethod.\*"/>  </methods>  </class>  </classes>  </test> |

### 元分组（MetaGroup）——分组的分组

|  |
| --- |
| <test name="Regression1">  <groups>  <define name="functest">  <include name="windows"/>  <include name="linux"/>  </define>    <define name="all">  <include name="functest"/>  <include name="checkintest"/>  </define>    <run>  <include name="all"/>  </run>  </groups>    <classes>  <class name="test.sample.Test1"/>  </classes>  </test> |

### 部分组（Partial group）

在类级别定义的组，下例中两个方法都属于checkin-test组：

|  |
| --- |
| @Test(groups = { "checkin-test" })  public class All {    @Test(groups = { "func-test" )  public void method1() { ... }    public void method2() { ... }  } |

## 排序

默认按配置序执行，但如果想按不可预期的顺序执行，设置preserve-order="false"：

|  |
| --- |
| <test name="简单测试" preserve-order="false"> |

## 参数

### xml参数

|  |
| --- |
| @Parameters({ "first-name" })  @Test  public void testSingleString(String firstName) {  System.out.println("Invoked testString " + firstName);  assert "Cedric".equals(firstName);  }    <suite name="My suite">  <parameter name="first-name" value="Cedric"/>  <test name="Simple example">  <-- ... --> |

也可应用于@Before、@After、@Factory或至多一个构造器。

指定默认值：

|  |
| --- |
| @Parameters("db")  @Test  public void testNonExistentParameter(@Optional("mysql") String db) { ... } |

### 数据提供器（DataProvider）

|  |
| --- |
| @DataProvider(name = "test1")  public Object[][] createData1() {  return new Object[][] {  { "Cedric", new Integer(36) },  { "Anne", new Integer(37)},  };  }    @Test(dataProvider = "test1")  public void verifyData1(String n1, Integer n2) {  System.out.println(n1 + " " + n2);  } |

默认情况下，将在当前类或其某个基类中查找数据提供器。如果想将数据放在一个不同的类中，需要一个静态方法或一个带无参构造器的类：

|  |
| --- |
| public class StaticProvider {  @DataProvider(name = "create")  public static Object[][] createData() {  return new Object[][] {  new Object[] { new Integer(42) }  };  }  }    public class MyTest {  @Test(dataProvider = "create", dataProviderClass = StaticProvider.class)  public void test(Integer n) {  // ...  }  } |

注入的数据可以是 Object[][] 或 Iterator<Object[]>。

**获取被调用方法**

TestNG会将当前调用方法(Method)传给数据提供器作为第一参数：

|  |
| --- |
| @DataProvider(name = "dp")  public Object[][] createData(Method m) {  System.out.println(m.getName()); // print test method name  return new Object[][] { new Object[] { "Cedric" }};  } |

这使得多个测试方法可以使用同一数据提供器，并根据方法不同生成不同的数据。

**并行**

|  |
| --- |
| @DataProvider(parallel = true) |

从xml启动测试线程池大小默认为10，可配置：

|  |
| --- |
| <suite name="Suite1" data-provider-thread-count="20" > |

## 依赖

### 注解依赖

硬依赖（hard dependency） 依赖项之一失败则跳过不执行。

软依赖（soft dependency） 依赖项失败仍然执行，仅保证执行顺序，@Test(alwaysRun=true)。

**依赖方法**

|  |
| --- |
| @Test  public void serverStartedOk() {}    @Test(dependsOnMethods = { "serverStartedOk" })  public void method1() {} |

**依赖组**

|  |
| --- |
| @Test(groups = { "init" })  public void serverStartedOk() {}    @Test(groups = { "init" })  public void initEnvironment() {}    @Test(dependsOnGroups = { "init.\*" })  public void method1() {} |

注：支持正则表达式。

**按实例分组依赖**

|  |
| --- |
| <suite name="Factory" group-by-instances="true">  <test name="Factory" group-by-instances="true"> |

### XML依赖

|  |
| --- |
| <test name="My suite">  <groups>  <dependencies>  <group name="c" depends-on="a b" />  <group name="z" depends-on="c" />  </dependencies>  </groups>  </test> |

## 工厂

|  |
| --- |
| public class WebTestFactory {  @Factory  public Object[] createInstances() {  Object[] result = new Object[10];  for (int i = 0; i < 10; i++) {  result[i] = new WebTest(i \* 10);  }  return result;  }  }    public class WebTest {  private int m\_numberOfTimes;  public WebTest(int numberOfTimes) {  m\_numberOfTimes = numberOfTimes;  }    @Test  public void testServer() {  for (int i = 0; i < m\_numberOfTimes; i++) {  // access the web page  }  }  }    <class name="WebTestFactory" /> |

工厂方法可以接收参数，并且必须返回 Object[]。

**构造器工厂**

|  |
| --- |
| @Factory(dataProvider = "dp")  public FactoryDataProviderSampleTest(int n) {  super(n);  }    @DataProvider  static public Object[][] dp() {  return new Object[][] {  new Object[] { 41 },  new Object[] { 42 },  };  } |

## 类级注解

@Test可应用于Class，使类中所有public方法成为测试方法。

## 并行测试

### 并行测试套

|  |
| --- |
| java org.testng.TestNG -suitethreadpoolsize 3 testng1.xml testng2.xml testng3.xml |

### 并行级别

|  |
| --- |
| <suite name="测试套件" parallel="classes" thread-count="2" > |

**parallel**

|  |  |
| --- | --- |
| methods | TestNG将在不同的线程中运行所有的测试方法。依赖方法也将在单独的线程中运行，但它们将尊重你指定的顺序。 |
| tests | TestNG 将在同一个线程中运行相同的标记的所有方法，但是每个标记将在一个单独的线程中。这允许你将所有非线程安全的类分组在同一个中，并保证它们将在同一个线程中运行，同时利用尽可能多的线程来运行测试。 |
| classes | TestNG将在同一个线程中运行同一个类中的所有方法，但是每个类都将在一个单独的线程中运行。 |
| instances | TestNG将在同一个线程中运行相同实例中的所有方法，但是在两个不同实例上的两个方法将在不同的线程中运行。 |

**thread-count**

用于指定线程的个数。

## 执行JUnit

|  |
| --- |
| <suite name="Converted JUnit suite" >  <test name="JUnitTests" junit="true">  <classes>  <class name="TestJunit" />  </classes>  </test>  </suite> |

JUnit 4中，TestNG将使用 org.junit.runner.JUnitCore 运行测试。

## 带返回值方法

默认情况下，如果一个方法有返回值，即使被@Test注解也会被忽略，不会执行，除非设置allow-return-values="true"：

|  |
| --- |
| <suite allow-return-values="true">  <test allow-return-values="true"> <!-- (?) --> |

# 执行

## 命令行

|  |
| --- |
| java org.testng.TestNG testng1.xml [testng2.xml testng3.xml ...] |

### 参数

参数可以放在一个文本文件中：

|  |
| --- |
| C:> more c:\command.txt  -d test-output testng.xml  C:> java org.testng.TestNG @c:\command.txt |

### JVM命令行属性

|  |
| --- |
| java -Dtestng.test.classpath="c:/build;c:/java/classes;" org.testng.TestNG testng.xml |

# 附录

Test Fixture 是指一个测试运行所需的固定环境，准确的定义：

The test fixture is everything we need to have in place to exercise the SUT.