用户操作

[留言] [发消息] [加为好友]

订阅我的博客

0 位读者 POWERED BY FEEDSKY 订阅

订阅到 鲜果

订阅到 Google

订阅到 抓虾

zyl623的公告

文章分类

aop
eclipse
jdk
orm
osgi
ror
soa
spring
webservies
单元测试
个人感想

关于TestNG

收藏

TestNG是一个不错的测试框架，尤其是用于模块测试，以及大范围的测试。相对于JUnit来说，更为灵活。随着JUnit4的推出，很多功能都与TestNG相似，但相对于JUnit4，TestNG还是有很多部分是有区别的。

TestNG的IDE支持也不错，对于Eclipse,Idea,Ant都有很好的支持。

先来看一看怎么使用TestNG，当然首先需要下载TestNG包。目前的版本为5.1，下载地址如下：http://testng.org/doc/download.html ，也可以下载相应的Eclipse插件。

运行TestNG，可以从命令行或者IDE，或者Ant中运行。

命令行：

java org.testng.TestNG -groups windows,linux -testclass org.test.MyTest

对于大型的测试，需要定义一个xml文件，一般为testng.xml。

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

<suitename="Suite1"    verbose="1" >
<testname="Nopackage" >
<classes>
      <classname="NoPackageTest"  />
</classes>
</test>

<testname="Regression1"   >
<classes>
<classname="test.sample.ParameterSample"  />
<classname="test.sample.ParameterTest" />
</classes>
</test>
</suite>
```

java org.testng.TestNG testng.xml
当然如果使用Eclipse插件，就简单多了。

下面来看一下，如何来实现测试的，与JUnit4 差不多（怀疑，JUnit4是不是有抄袭TestNG的成分）。
声明测试方法如下：

```
  @Test
    public void testMethod1() {
        System.out.println("in testMethod1");
    }

@Test
    public void testMethod2() {
        System.out.println("in testMethod2");
    }
```

基本上都是采用java5的注释实现的。
与JUnit4 不同在于，测试方法可以分组，它可以根据诸如运行时间这样的特征来对测试分类。

```
  @Test(groups={"fun1","fun2"})
    public void testMethod1() {
        System.out.println("in testMethod1");
    }

@Test(groups={"fun1"})
    public void testMethod2() {
        System.out.println("in testMethod2");
    }
```

同JUnit4 一样，同样支持Before,After方法，如同setUp 和tearDown,不过TestNG更为灵活，支持各种签名方式，如private,protected。

```
  @BeforeMethod
  protected void beforeMethod() {
```

```
    System.out.println("in beforeMethod");
}

@AfterMethod
protected void afterMethod() {
    System.out.println("in afterMethod");
}
```

同样也支持BeforeClass 和AfterClass，只执行一次的方法，但是可以不需要使用static签名

```
@BeforeClass
protected void beforeClassMethod() {
    System.out.println("in beforeClassMethod");
}

@AfterClass
protected void afterClassMethod() {
    System.out.println("in afterClassMethod");
}
```

不同于JUnit4，TestNG提供了以下的特性：

依赖性测试

JUnit 框架想达到的一个目标就是测试隔离。它的缺点是：人们很难确定测试用例执行的顺序，而这对于任何类型的依赖性测试都非常重要。开发者们使用了多种技术来解决这个问题，例如，按字母顺序指定测试用例，或是更多地依靠 fixture 来适当地解决问题。

与 JUnit 不同，TestNG 利用 `Test` 注释的 `dependsOnMethods` 属性来应对测试的依赖性问题。有了这个便利的特性，就可以轻松指定依赖方法。如以下定义：testMethod2依赖于testMethod1。

```
@Test
public void testMethod1() {
    System.out.println("in testMethod1");
}

@Test(dependsOnMethods="testMethod1")
public void testMethod2() {
    System.out.println("in testMethod2");
}
```

当然如果testMethod1失败的话，默认testMethod2也不会执行，不过只需要设置alwaysRun = true，则可以跳过testMethod1

```
@Test
public void testMethod1() {
   System.out.println("in testMethod1");
   throw new RuntimeException("failed");
}

@Test(dependsOnMethods="testMethod1",alwaysRun = true)
public void testMethod2() {
   System.out.println("in testMethod2");
}
```

失败和重运行

在大型测试套件中，这种重新运行失败测试的能力显得尤为方便。这是 TestNG 独有的一个特性。在 JUnit 4 中，如果测试套件包括 1000 项测试，其中 3 项失败，很可能就会迫使您重新运行整个测试套件（修改错误以后）。不用说，这样的工作可能会耗费几个小时。

一旦 TestNG 中出现失败，它就会创建一个 XML 配置文件，对失败的测试加以说明。如果利用这个文件执行 TestNG 运行程序，TestNG 就只 运行失败的测试。所以，在前面的例子里，您只需重新运行那三个失败的测试，而不是整个测试套件。可以看到以下的失败文件，一般命名为testng-failed.xml，以后只需要运行此文件就可以了。

```
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite thread-count="5" verbose="1" name="Failed suite [testng]" parallel="false" annotations="JDK5">
  <test name="demo.testng.Test2(failed)" junit="false" parallel="false" annotations="JDK5">
    <classes>
      <class name="demo.testng.Test2">
        <methods>
          <include name="testMethod1"/>
          <include name="testMethod2"/>
          <include name="beforeClassMethod"/>
          <include name="afterClassMethod"/>
          <include name="beforeMethod"/>
```

```
      <include name="afterMethod"/>
     </methods>
    </class>
   </classes>
  </test>
</suite>
```

参数化测试

TestNG 中另一个有趣的特性是参数化测试。在 JUnit 中，如果您想改变某个受测方法的参数组，就只能给每个 不同的参数组编写一个测试用例。多数情况下，这不会带来太多麻烦。然而，我们有时会碰到一些情况，对其中的业务逻辑，需要运行的测试数目变化范围很大。

在这样的情况下，使用 JUnit 的测试人员往往会转而使用 FIT 这样的框架，因为这样就可以用表格数据驱动测试。但是 TestNG 提供了开箱即用的类似特性。通过在 TestNG 的 XML 配置文件中放入参数化数据，就可以对不同的数据集重用同一个测试用例，甚至有可能会得到不同的结果。这种技术完美地避免了只能 假定一切正常的测试，或是没有对边界进行有效验证的情况。

```
 @Parameters( { "first-name"
 })
 @Test(groups = { "param"
 })
 public void testParm(String firstName) {
    System.out.println("invoked testString:" + firstName);
    assertEquals(firstName, "Test");
 }
```

在xml中设置相应的参数值，可以放入suite下面或者test下面，如果同名，一般test下面的定义覆盖suite定义。

```
<parameter name="first-name" value="Test"/>
```

高级参数化测试

尽管从一个 XML 文件中抽取数据会很方便，但偶尔会有些测试需要有复杂类型，这些类型无法用 `String` 或原语值来表示。TestNG 可以通过它的 `@DataProvider` 注释处理这样的情况。`@DataProvider` 注释可以方便地把复杂参数类型映射到某个测试方法。例如，清单 7 中的 `verifyHierarchy` 测试中，我采用了重载的 `buildHierarchy` 方法，它可接收一个 `Class` 类型的数据，它断言（asserting）`Hierarchy` 的 `getHierarchyCl`

assNames() 方法应该返回一个适当的字符串数组:

```
package test.com.acme.da.ng;

import java.util.Vector;

import static org.testng.Assert.assertEquals;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

import com.acme.da.hierarchy.Hierarchy;
import com.acme.da.hierarchy.HierarchyBuilder;

public class HierarchyTest {

@DataProvider(name = "class-hierarchies")
 public Object[][] dataValues(){
  return new Object[][]{
   {Vector.class, new String[] {"java.util.AbstractList",
     "java.util.AbstractCollection"}},
   {String.class, new String[] {}}
  };
 }

@Test(dataProvider = "class-hierarchies")
 public void verifyHierarchy(Class clzz, String[] names)
  throws Exception{
    Hierarchy hier = HierarchyBuilder.buildHierarchy(clzz);
    assertEquals(hier.getHierarchyClassNames(), names,
          "values were not equal");
 }
}
```

当然还有一些其他的特性，就不一一详细说明了，有兴趣可以参考相应的testNG文档。

JUnit 4 和 TestNG 在表面上是相似的。然而，设计 JUnit 的目的是为了分析代码单元，而 TestNG 的预期用途则针对高级测试。对于大型测试套件，我们不希望在某一项测试失败时就得重新运行数千项测试，TestNG 的

灵活性在这里尤为有用。这两个框架都有自己的优势，您可以随意同时使用它们。

发表于 @ 2006年10月05日 15:46:00 | 评论(0) | 举报 | 收藏

旧一篇:Junit 4新的特性 | 新一篇: 关于AspectJ 中的pointcut 语法

发表评论 “评论王争夺赛”第3期活动开始啦！

表 情:

评论内容:

用 户 名: huapuyu6 匿名评论

发表评论

Copyright © zyl623

Powered by CSDN Blog