08 TestNG高级应用

本章大纲

8.1 Java代码执行testNG配置文件

8.2 使用ITestContext共享数据

8.3 ITestResults接口

8.4 Listeners监听

Java代码执行testNG配置文件

```
TestNG testNG = new TestNG();
List<String> suites = new ArrayList<String>();
suites.add(".\\testng.xml");
testNG.setTestSuites(suites);
testNG.run();
// 等待执行结束，然后去执行失败用例
TestNG testNG1 = new TestNG();
List<String> suites1 = new ArrayList<String>();
Thread.sleep(5000);
suites1.add(".\\test-output\\testng-failed.xml");
testNG1.setTestSuites(suites1);
testNG1.run();
```

使用ITestContext共享数据

```java
public class Demo {
    @Test
    public void keyword(ITestContext context) {
        context.setAttribute("name", "tom");
    }

    @Test
    public void keywordPass(ITestContext context) {
        System.out.println(context.getAttribute("name"));
    }
}
```

ITestResults

- ITestResult.getName() 是测试用例名
- ITestResult.getStatus() 测试执行结果：

  ITestResult.SUCCESS，ITestResult.FAILURE，

  ITestResult.SKIP
- ITestResult.getInstanceName()是类名
- ITestResult.getThrowable() 测试的报错信息

Listeners监听

TestNG的监听器（listener）是一系列的接口，用于在运行时设置动态参数，以改变TestNG的执行过程。

TestNG会监听每个测试case的运行结果，有时候我们需要定制一些其他功能，如自动截图，发送邮件给相关人员等。

- IAnnotationTransformer (doc, javadoc)
- IAnnotationTransformer2 (doc, javadoc)
- IHookable (doc, javadoc)
- IInvokedMethodListener (doc, javadoc)
- IMethodInterceptor (doc, javadoc)
- IReporter (doc, javadoc)
- ISuiteListener (doc, javadoc)
- ITestListener (doc, javadoc)

Listeners监听

1. IAnnotationTransformer，操作@Test标注
2. IAnnotationTransformer2，操作@Configuration标注、@DataProvider标注和@Factory标注
3. IAnnotationTransformer3，操作@Listeners标注
4. IMethodInterceptor监听器，可以改变这些测试方法的执行顺序
5. IInvokedMethodListener监听器，能够在调用某个测试方法之前或者之后发出通知
6. IHookable监听器，就是根据当前执行的情况决定是否执行某个测试方法。即测试方法的执行有先决条件，满足条件则执行，否则就跳过执行。

ITestListener

ITestListener监听器如果要在测试方法执行成功、失败或者跳过时指定不同后续行为，可以利用ITestListener 监听器。也可以直接继承TestListenerAdapter类来实现。该类是一个基于ITestListener 的简单适配器，存储了被运行的所有测试用例。

ITestListener

实现监听器的步骤：

1、创建监听类，实现接口ITestListener ，或者继承类TestListenerAdapter

重写三个方法

onTestStart, onTestSuccess(), onTestFailure()

2、配置testng.xml文件，并设置Lienter范围，或者为测试类设置@Listeners注解

```
<listeners>
<listener class-name="Listener-name" />
</listeners>
```

IReporter 接口

TestNG 提供了默认的测试报表。但如果用户希望有不同格式的测试报表，就需要使用 IReporter 监听器。

IReporter 监听器只有一个方法需要实现。

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

该方法在所有测试方法执行结束后被调用，通过遍历 xmlSuites 和 suites 能够获取所有测试方法的信息以及测试结果。outputDirectory 是默认的测试报表生成路径，可以指定其他路径生成报表。

IReporter 接口

这个方法有三个参数：

第一个是xmlSuite，这是TestNG的测试XML正在执行中提到的列表套件。

第二个是套件，其中包含一套测试执行后信息，该对象包含了所有的信息包，类，测试方法和测试执行结果。

第三的outputDirectory，报告将产生的输出文件夹路径，其中包含的信息。

用例失败重跑方法第二种方法

接口IRetryAnalyzer

该接口的作用是提供机会去实现能够让失败用例重跑。实现该接口必须要实现retry(ITestResult result)这个方法。返回值类型是布尔型，如果返回是True，那么就执行失败重跑，返回是false，就不重跑。参数result是当前运行的测试用例的结果状态。

用例失败重跑第二种方法

接口IAnnotationTransformer

该接口的作用是在TestNG执行过程中动态改变测试类中Annotation的参数，当前这个接口主要是针对@Test注释。

IAnnotationTransformer监听器接口只有一个方法

transform(ITestAnnotation annotation, Class testClass, Constructor testConstructor,Method testMethod)

用例失败重跑第二种方法

```
public class MyRetry implements IRetryAnalyzer {

// 设置当前失败执行的次数

    private int retryCount = 1;
// 设置最大失败执行次数
    private static int maxRetryCount = 3;
    @Override
    public boolean retry(ITestResult iTestResult) {

        if (retryCount < maxRetryCount) {
            retryCount++;
            return true;
        }
        return false;

    }
}
```

用例失败重跑方法第二种方法

```
public class MyRetryListener implements IAnnotationTransformer {

    @Override
    public void transform(ITestAnnotation iTestAnnotation, Class aClass, Construc

        IRetryAnalyzer myRetry = iTestAnnotation.getRetryAnalyzer();

        if (myRetry == null) {
            iTestAnnotation.setRetryAnalyzer(MyRetry.class);
        }

    }



}
```