TestNG学习笔记-测试设计模式-数据驱动测试

一、数据驱动测试(DDT)-概览

主要用来解决以下形式的问题:

- 测试需要针对许多具有类似数据结构的数据来执行
- 实际的测试逻辑是一样的，仅仅改变的是数据结构
- 数据可以被一组不同的人修改

它的重点不是被测试的程序代码逻辑，而是这段代码所想要操作的数据。

TestNG可以让你通过两种方式向测试方法传递参数:

- 利用testng.xml
- 利用data providers

二、 利用testng.xml传递参数

可以在testng.xml中配置需要传递的参数，用标签，可以在suite中声明，也可以在test中声明，如果test中有，则优先使用test中定义的，如果没有则会往上找。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="ddt with Parameters">
        <test thread-count="5" name="Test">
                <parameter name="name" value="nameInTest"></parameter>
                <parameter name="sex" value="sexInTest"></parameter>
                <classes>
                        <class
                                name="com.autotest.java.demo.testng.ddt.TestNGDDTDemoTest">
                                <methods>
                                        <method name="validateUserTest"></method>
                                </methods>
                        </class>
                </classes>
        </test>
</suite>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="DDT test in suite Parameters">
        <parameter name="name" value="nameInSuite"></parameter>
        <parameter name="sex" value="sexInSuite"></parameter>
        <test thread-count="5" name="Test">
                <classes>
                        <class
                                name="com.autotest.java.demo.testng.ddt.TestNGDDTDemoTest">
                                <methods>
                                        <method name="validateUserTest"></method>
                                </methods>
                        </class>
                </classes>
        </test> <!-- Test -->
</suite> <!-- Suite -->
```

在测试代码中使用@Parameters注解来获取，按照如下方式使用:

```
@Test
@Parameters({ "name", "sex" })
public void validateUserTest(String name, String sex) {
    LOGGER.debug("name: {},sex:{}", name, sex);
    boolean result = testNGDemo.validateUser(name, sex);
    LOGGER.debug("验证结果:{}", result);
}
```

如下是分别使用这两个配置文件运行的结果:

```
[2020.04.29 16:26:59] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTest] name：nameInSuite,sex:sexInSuite

===============================================
DDT test in suite Parameters
Total tests run: 1, Passes: 0, Failures: 1, Skips: 0
===============================================
```

```
[2020.04.29 16:26:21] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTest] name：nameInTest,sex:sexInTest

===============================================
ddt with Parameters
Total tests run: 1, Passes: 0, Failures: 1, Skips: 0
===============================================
```

当然了，这种方式也有其缺点，就是不支持非基本类型的传递，或者说你需要的值，不能静态传递而是想要在运行时刻创建，则只能用DataProvider方式

三、 使用DataProvider来传递参数

3.1 通用概念

它可以用@DataProvider注解标注一个方法，它有一个字符串name属性，定义这个提供者的名称。
如果没有提供这个名称，则会默认使用方法的名称。

它通常返回一些Java对象，这些对象将会传递给某个被@Test注解标注的方法，@Test注解中使用dataProvider属性来指定。

从本质上讲，它解决了两个问题:

- 向测试方法传递任意数目的参数,可以是任何的Java类型
- 根据需要允许利用不同的参数集合，对测试方法进行多次测试

一个例子如下:
下面是java代码:

```
/**
 * 带参数的
 */
public boolean validateUser(String name, String sex) {
    if ("admin".equals(name) && "man".equals(sex)) {
        return true;
    }

    return false;
}
```

下面是java测试代码:

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Ignore;
import org.testng.annotations.Test;
public class TestNGDDTDemoTestWithDataProvider {
	private static final Logger LOGGER = LoggerFactory
		.getLogger(TestNGDDTDemoTestWithDataProvider.class);

	private TestNGDemo testNGDemo;

	@BeforeMethod
	private void initInstance() {
		testNGDemo = new TestNGDemo();
	}

	@Test
	@Ignore
	public void validateUserTest(String name, String sex) {
		LOGGER.debug("name: {},sex:{}", name, sex);
		testNGDemo.validateUser(name, sex);
	}

	@Test(dataProvider = "initTestUsersClearly")
	public void validateUserTestWithProviderClearly(String name, String sex) {
		LOGGER.debug("name: {},sex:{}", name, sex);
		testNGDemo.validateUser(name, sex);
	}

	@Test(dataProvider = "initTestUsersUnClearly")
	public void validateUserTestWithProviderUnClearly(String name, String sex) {
		LOGGER.debug("name: {},sex:{}", name, sex);
		LOGGER.debug("testNGDemo:{}", testNGDemo);
		testNGDemo.validateUser(name, sex);
	}

	@DataProvider(name = "initTestUsersClearly")
	public Object[][] initTestUsersClearly() {
		LOGGER.debug("initTestUsersClearly方法被调用");
		return new Object[][] { { "admin", "man" }, { "admin2", "women" },
			{ "admin", "women" } };

	}

	@DataProvider
	public Object[][] initTestUsersUnClearly() {
		LOGGER.debug("initTestUsersUnClearly方法被调用");
		return new Object[][] { { "admin", "man" }, { "admin2", "women" },
			{ "admin", "women" } };
	}

}
```

测试运行结果如下:

```
[2020.04.29 17:01:48] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProvider] initTestUsersClearly方法被调用
[2020.04.29 17:01:48] [INFO ] [c.a.j.d.t.TestNGDemo] com.autotest.java.demo.testng.TestNGDemo构造方法被初始化
[2020.04.29 17:01:48] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProvider] name: admin,sex:man
[2020.04.29 17:01:48] [INFO ] [c.a.j.d.t.TestNGDemo] com.autotest.java.demo.testng.TestNGDemo构造方法被初始化
[2020.04.29 17:01:48] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProvider] name: admin2,sex:women
[2020.04.29 17:01:48] [INFO ] [c.a.j.d.t.TestNGDemo] com.autotest.java.demo.testng.TestNGDemo构造方法被初始化
[2020.04.29 17:01:48] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProvider] name: admin,sex:women
[2020.04.29 17:01:48] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProvider] initTestUsersUnClearly方法被调用
[2020.04.29 17:01:48] [INFO ] [c.a.j.d.t.TestNGDemo] com.autotest.java.demo.testng.TestNGDemo构造方法被初始化
[2020.04.29 17:01:48] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProvider] name: admin,sex:man
[2020.04.29 17:01:48] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProvider] testNGDemo:com.autotest.java.demo.testng.TestNGDemo@166fa74d
[2020.04.29 17:01:48] [INFO ] [c.a.j.d.t.TestNGDemo] com.autotest.java.demo.testng.TestNGDemo构造方法被初始化
[2020.04.29 17:01:48] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProvider] name: admin2,sex:women
[2020.04.29 17:01:48] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProvider] testNGDemo:com.autotest.java.demo.testng.TestNGDemo@588df31b
[2020.04.29 17:01:48] [INFO ] [c.a.j.d.t.TestNGDemo] com.autotest.java.demo.testng.TestNGDemo构造方法被初始化
[2020.04.29 17:01:48] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProvider] name: admin,sex:women
[2020.04.29 17:01:48] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProvider] testNGDemo:com.autotest.java.demo.testng.TestNGDemo@7181ae3f
PASSED: validateUserTestWithProviderClearly("admin", "man")
PASSED: validateUserTestWithProviderClearly("admin2", "women")
PASSED: validateUserTestWithProviderClearly("admin", "women")
PASSED: validateUserTestWithProviderUnClearly("admin", "man")
PASSED: validateUserTestWithProviderUnClearly("admin2", "women")
PASSED: validateUserTestWithProviderUnClearly("admin", "women")

===============================================
    Default test
    Tests run: 6, Failures: 0, Skips: 0
===============================================


===============================================
Default suite
Total tests run: 6, Passes: 6, Failures: 0, Skips: 0
===============================================
```

以上例子证明了之前的结论。

总结如下:

- DataProvider的name属性是可选的，如果不指定则用方法名称。一般来说不建议默认。
  因为有时候你可能重命名方法(重构的时候，或者复制多个的时候等等)，则会导致测试出错，因为这个方法名被Test注解引用过，除非有特殊的工具支持或者TestNg本身支持，重构方法不会重构Test注解中的这个名称（当然你也可以全局搜索替换）。
- 由于数据提供者是测试类的一个方法，可以让它归属于一个超类，然后被一些测试方法复用。月可以有多个数据提供者方法，使用不同的名称，只要它们定义在测试类或者其子类上，效果也是一样的。

3.2 针对数据提供者的参数

数据提供者方法本身可以接受两个类型的参数Method和ITestContext,在调用数据提供者方法的时候会设置这两个参数,它们为代码提供了某种上下文，然后在此上下文的基础上决定再做什么。

可以根据需要任意设置这两个参数的组合。

3.2.1 关于Method

先来看个例子:

```
@Test(dataProvider = "initTestUsersWithDataProviderParam")
public void validateUserTestWithProviderParam(String name, String sex) {
    LOGGER.debug("name: {},sex:{}", name, sex);
    testNGDemo.validateUser(name, sex);
}

@DataProvider(name = "initTestUsersWithDataProviderParam")
public Object[][] initTestUsersWithDataProviderParam(Method method,
        ITestContext iTestContext) {
    LOGGER.debug("initTestUsersWithDataProviderParam方法被调用,传入参数:{},传入上下文:{}",
            method, iTestContext);
    return new Object[][] { { "admin", "man" }, { "admin2", "women" },
            { "admin", "women" } };
}
```

下面是执行结果：

```
[2020.04.29 17:17:27] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProvider] initTestUsersWithDataProviderParam方法被调用,传入参数:public void com.autotest.java.
[2020.04.29 17:17:27] [INFO ] [c.a.j.d.t.TestNGDemo] com.autotest.java.demo.testng.TestNGDemo构造方法被初始化
[2020.04.29 17:17:27] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProvider] name: admin,sex:man
[2020.04.29 17:17:27] [INFO ] [c.a.j.d.t.TestNGDemo] com.autotest.java.demo.testng.TestNGDemo构造方法被初始化
[2020.04.29 17:17:27] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProvider] name: admin2,sex:women
[2020.04.29 17:17:27] [INFO ] [c.a.j.d.t.TestNGDemo] com.autotest.java.demo.testng.TestNGDemo构造方法被初始化
[2020.04.29 17:17:27] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProvider] name: admin,sex:women
PASSED: validateUserTestWithProviderParam("admin", "man")
PASSED: validateUserTestWithProviderParam("admin2", "women")
PASSED: validateUserTestWithProviderParam("admin", "women")

===============================================
    Default test
    Tests run: 3, Failures: 0, Skips: 0
===============================================


===============================================
Default suite
Total tests run: 3, Passes: 3, Failures: 0, Skips: 0
===============================================
```

从上面可以看出来，传进来的参数Method是调用了这个dataProvider的方法，也就是哪个方法使用这个数据提供者，则**可以根据不同的测试方法来决定是否需要区别对待测试方法。**

更有甚者，可以针对不同的测试方法来返回不同的测试数据。

如下代码，是一个根据不同测试方法返回不同测试数据的，按需改造即可，实际场景中不一定经常有这种需求。

```
@Test(dataProvider = "initTestUsersWithDataProviderParamOfDiffTestMethod")
public void validateUserTestWithProviderParamOne(String name, String sex) {
    LOGGER.debug("name: {},sex:{}", name, sex);
    testNGDemo.validateUser(name, sex);
}

@Test(dataProvider = "initTestUsersWithDataProviderParamOfDiffTestMethod")
public void validateUserTestWithProviderParamTwo(String name, String sex) {
    LOGGER.debug("name: {},sex:{}", name, sex);
    testNGDemo.validateUser(name, sex);
}

@Test(dataProvider = "initTestUsersWithDataProviderParamOfDiffTestMethod")
public void validateUserTestWithProviderParamDefault(String name,
        String sex) {
    LOGGER.debug("name: {},sex:{}", name, sex);
    testNGDemo.validateUser(name, sex);
}

@DataProvider(name = "initTestUsersWithDataProviderParamOfDiffTestMethod")
public Object[][] initTestUsersWithDataProviderParamOfDiffTestMethod(
        Method method, ITestContext iTestContext) {
    LOGGER.debug(
            "initTestUsersWithDataProviderParamOfDiffTestMethod方法被调用,传入参数:{},传入上下文:{}",
            method, iTestContext);

    Object[][] resultObjects = new Object[][] { { "admin", "man" },
            { "admin2", "women" } };

    if (method.getName().equals("validateUserTestWithProviderParamOne")) {
        resultObjects = new Object[][] { { "admin", "man" },
                { "admin3", "women" }, { "admin4", "women" } };
    } else if (method.getName()
            .equals("validateUserTestWithProviderParamTwo")) {
        resultObjects = new Object[][] { { "admin", "man" },
                { "admin5", "women" }, { "admin6", "women" } };
    }
    return resultObjects;
}
```

看看下面的输出结果：

```
[2020.04.29 17:41:16] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProvider] initTestUsersWithDataProviderParamOfDiffTestMethod方法被调用,
传入参数:public void com.autotest.java.demo.testng.ddt.TestNGDDTDemoTestWithDataProvider.validateUserTestWithProviderParamDefault(java.lang.String,java.lang.Stri
传入上下文:org.testng.TestRunner@80169cf
[2020.04.29 17:41:16] [INFO ] [c.a.j.d.t.TestNGDemo] com.autotest.java.demo.testng.TestNGDemo构造方法被初始化
[2020.04.29 17:41:16] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProvider] name: admin,sex:man
[2020.04.29 17:41:16] [INFO ] [c.a.j.d.t.TestNGDemo] com.autotest.java.demo.testng.TestNGDemo构造方法被初始化
[2020.04.29 17:41:16] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProvider] name: admin2,sex:women
[2020.04.29 17:41:16] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProvider] initTestUsersWithDataProviderParamOfDiffTestMethod方法被调用,
传入参数:public void com.autotest.java.demo.testng.ddt.TestNGDDTDemoTestWithDataProvider.validateUserTestWithProviderParamOne(java.lang.String,java.lang.String),
传入上下文:org.testng.TestRunner@80169cf
[2020.04.29 17:41:16] [INFO ] [c.a.j.d.t.TestNGDemo] com.autotest.java.demo.testng.TestNGDemo构造方法被初始化
[2020.04.29 17:41:16] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProvider] name: admin,sex:man
[2020.04.29 17:41:16] [INFO ] [c.a.j.d.t.TestNGDemo] com.autotest.java.demo.testng.TestNGDemo构造方法被初始化
[2020.04.29 17:41:16] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProvider] name: admin3,sex:women
[2020.04.29 17:41:16] [INFO ] [c.a.j.d.t.TestNGDemo] com.autotest.java.demo.testng.TestNGDemo构造方法被初始化
[2020.04.29 17:41:16] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProvider] name: admin4,sex:women
[2020.04.29 17:41:16] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProvider] initTestUsersWithDataProviderParamOfDiffTestMethod方法被调用,
传入参数:public void com.autotest.java.demo.testng.ddt.TestNGDDTDemoTestWithDataProvider.validateUserTestWithProviderParamTwo(java.lang.String,java.lang.String),
传入上下文:org.testng.TestRunner@80169cf
[2020.04.29 17:41:16] [INFO ] [c.a.j.d.t.TestNGDemo] com.autotest.java.demo.testng.TestNGDemo构造方法被初始化
[2020.04.29 17:41:16] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProvider] name: admin,sex:man
[2020.04.29 17:41:16] [INFO ] [c.a.j.d.t.TestNGDemo] com.autotest.java.demo.testng.TestNGDemo构造方法被初始化
[2020.04.29 17:41:16] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProvider] name: admin5,sex:women
[2020.04.29 17:41:16] [INFO ] [c.a.j.d.t.TestNGDemo] com.autotest.java.demo.testng.TestNGDemo构造方法被初始化
[2020.04.29 17:41:16] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProvider] name: admin6,sex:women
PASSED: validateUserTestWithProviderParamDefault("admin", "man")
PASSED: validateUserTestWithProviderParamDefault("admin2", "women")
PASSED: validateUserTestWithProviderParamOne("admin", "man")
PASSED: validateUserTestWithProviderParamOne("admin3", "women")
PASSED: validateUserTestWithProviderParamOne("admin4", "women")
PASSED: validateUserTestWithProviderParamTwo("admin", "man")
PASSED: validateUserTestWithProviderParamTwo("admin5", "women")
PASSED: validateUserTestWithProviderParamTwo("admin6", "women")

===============================================
    Default test
    Tests run: 8, Failures: 0, Skips: 0
===============================================


===============================================
Default suite
Total tests run: 8, Passes: 8, Failures: 0, Skips: 0
===============================================
```

3.2.2 关于ITestContext

如果数据提供者方法中传入了这个参数，那么TestNG可以把运行时刻的一些信息设置给它，这就可以让数据提供者知道当前测试执行的运行时参数。

还是看个例子:

如下是一个配置了Test和groups的配置文件:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="DDT DATAPROIVIDER SUITE">
        <test thread-count="5" name="DDT DATAPROVIDER TEST">
                <classes>
                        <class
                                name="com.autotest.java.demo.testng.ddt.TestNGDDTDemoTestWithDataProviderITestContext">
                        </class>
                </classes>
        </test> <!-- Test -->

        <groups>
                <run>
                        <include name="testGroup1"></include>
                        <include name="testGroup2"></include>
                </run>
        </groups>
</suite> <!-- Suite -->
```

运行结果如下:

```
[2020.04.29 18:03:20] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProviderITestContext] initTestUsersWithDataProviderParamOfDiffTestMethod方法被调用,
传入上下文信息:org.testng.TestRunner@b9afc07
[2020.04.29 18:03:20] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProviderITestContext]
测试组:[testGroup2, testGroup1],suite:DDT DATAPROVIDER TEST
[2020.04.29 18:03:20] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProviderITestContext] name: admin,sex:man
[2020.04.29 18:03:20] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProviderITestContext] name: admin2,sex:women
[2020.04.29 18:03:20] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProviderITestContext] initTestUsersWithDataProviderParamOfDiffTestMethod方法被调用,
传入上下文信息:org.testng.TestRunner@b9afc07
[2020.04.29 18:03:20] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProviderITestContext]
测试组:[testGroup2, testGroup1],suite:DDT DATAPROVIDER TEST
[2020.04.29 18:03:20] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProviderITestContext] name: admin,sex:man
[2020.04.29 18:03:20] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProviderITestContext] name: admin3,sex:women
[2020.04.29 18:03:20] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProviderITestContext] name: admin4,sex:women
[2020.04.29 18:03:20] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProviderITestContext] initTestUsersWithDataProviderParamOfDiffTestMethod方法被调用
,传入上下文信息:org.testng.TestRunner@b9afc07
[2020.04.29 18:03:20] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProviderITestContext]
测试组:[testGroup2, testGroup1],suite:DDT DATAPROVIDER TEST
[2020.04.29 18:03:20] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProviderITestContext] name: admin,sex:man
[2020.04.29 18:03:20] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProviderITestContext] name: admin5,sex:women
[2020.04.29 18:03:20] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProviderITestContext] name: admin6,sex:women

===============================================
DDT DATAPROIVIDER SUITE
Total tests run: 8, Passes: 0, Failures: 8, Skips: 0
===============================================
```

3.3 延迟数据提供者

数据提供者本身返回一个二维数组，这是通常的用法。但有的时候，我们未必都会直接以明确的方式定义返回的数据，而是从数据库中查出来，那么就有可能有这么一种场景，你定义了很多数据，放到了二维数组中或者查出来上千万条数据放到二维数组中。

由于保存的是对象，那么这些个对象都会存在于内存中。你可以尝试一下，会导致内存溢出，不过现在的内存普遍比较大，这个测试比较苛刻（有兴趣的不妨试一下）。

显然这是不理想的，那就有一种方案可以解决这个问题，就是延迟初始化，这个概念并不新颖，软件工程中有许多领域都有这个应用。它的核心思想就是当你真正需要一个对象的时候，再初始化它。

为了满足这个场景，TestNG允许我们从数据提供者返回一个Iterator，而不是一个二维数组。

它的好处就在于，当TestNG需要从数据提供者取得下一组数据的时候，会调用iterator的next方法，这样就可以在最后一刻初始化这个对象，刚好在需要这些参数的测试方法运行之前。

如下是代码示例:

```java
import java.lang.reflect.Method;
import java.util.Iterator;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.testng.ITestContext;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

import com.autotest.java.demo.testng.TestNGDemo;

/**
 * @author : Administrator
 * @since : 2020年4月29日 下午2:37:04
 * @see :
 */
public class TestNGDDTDemoTestWithDataProviderWithLazyInit {
	private static final Logger LOGGER = LoggerFactory
		.getLogger(TestNGDDTDemoTestWithDataProviderWithLazyInit.class);

	private TestNGDemo testNGDemo;

	@BeforeMethod
	private void initInstance() {
		testNGDemo = new TestNGDemo();
	}

	@Test(dataProvider = "initTestUsersWithoutDataProviderOfLazyInit")
	public void validateUserTestWithProviderParamTwo(String name, String sex) {
		LOGGER.debug("name: {},sex:{}", name, sex);
		testNGDemo.validateUser(name, sex);
	}

	@Test(dataProvider = "initTestUsersWithDataProviderOfLazyInit")
	public void validateUserTestWithProviderParamDefault(String name,
		String sex) {
		LOGGER.debug("name: {},sex:{}", name, sex);
		testNGDemo.validateUser(name, sex);
	}

	@DataProvider(name = "initTestUsersWithDataProviderOfLazyInit")
	public Iterator initTestUsersWithDataProviderOfLazyInit(Method method,
		ITestContext iTestContext) {
		LOGGER.debug("initTestUsersWithDataProviderOfLazyInit方法被调用,传入上下文信息:{}",
			iTestContext);
		Iterator iterator = new DataInitIterator();

		LOGGER.debug("Iterator:{}", iterator);
		return iterator;
	}

	@DataProvider(name = "initTestUsersWithoutDataProviderOfLazyInit")
	public Object[][] initTestUsersWithoutDataProviderOfLazyInit(Method method,
		ITestContext iTestContext) {
		LOGGER.debug(
			"initTestUsersWithoutDataProviderOfLazyInit方法被调用,传入上下文信息:{}",
			iTestContext);

		Object[][] resultObjects = new Object[][] { { "admin", "man" },
			{ "admin2", "women" } };

		return resultObjects;
	}

	class DataInitIterator implements Iterator {
		static private final int MAX = 4;
		private int index = 0;

		@Override
		public boolean hasNext() {
			return index < MAX;
```

```
                }

                @Override
                public Object next() {
                        int thisIndex = index++;
                        // 此处需要返回一个数组，因为每个位置是数组，才能放在一起组成一个跟二维数组等效的iterator
                        return new Object[] { "admin" + thisIndex, "sex" + thisIndex };
                }

                public void remote() {
                        throw new UnsupportedOperationException();
                }

        }
}
```

运行结果如下：

```
[2020.04.30 01:18:55] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProviderWithLazyInit] initTestUsersWithDataProviderOfLazyInit方法被调用,传入上下文信息:org.tes
[2020.04.30 01:18:55] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProviderWithLazyInit] Iterator:com.autotest.java.demo.testng.ddt.TestNGDDTDemoTestWithDataPr
[2020.04.30 01:18:55] [INFO ] [c.a.j.d.t.TestNGDemo] com.autotest.java.demo.testng.TestNGDemo构造方法被初始化
[2020.04.30 01:18:55] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProviderWithLazyInit] name: admin0,sex:sex0
[2020.04.30 01:18:55] [INFO ] [c.a.j.d.t.TestNGDemo] com.autotest.java.demo.testng.TestNGDemo构造方法被初始化
[2020.04.30 01:18:55] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProviderWithLazyInit] name: admin1,sex:sex1
[2020.04.30 01:18:55] [INFO ] [c.a.j.d.t.TestNGDemo] com.autotest.java.demo.testng.TestNGDemo构造方法被初始化
[2020.04.30 01:18:55] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProviderWithLazyInit] name: admin2,sex:sex2
[2020.04.30 01:18:55] [INFO ] [c.a.j.d.t.TestNGDemo] com.autotest.java.demo.testng.TestNGDemo构造方法被初始化
[2020.04.30 01:18:55] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProviderWithLazyInit] name: admin3,sex:sex3
[2020.04.30 01:18:55] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProviderWithLazyInit] initTestUsersWithoutDataProviderOfLazyInit方法被调用,传入上下文信息:org.
[2020.04.30 01:18:55] [INFO ] [c.a.j.d.t.TestNGDemo] com.autotest.java.demo.testng.TestNGDemo构造方法被初始化
[2020.04.30 01:18:55] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProviderWithLazyInit] name: admin,sex:man
[2020.04.30 01:18:55] [INFO ] [c.a.j.d.t.TestNGDemo] com.autotest.java.demo.testng.TestNGDemo构造方法被初始化
[2020.04.30 01:18:55] [DEBUG] [c.a.j.d.t.d.TestNGDDTDemoTestWithDataProviderWithLazyInit] name: admin2,sex:women
PASSED: validateUserTestWithProviderParamDefault("admin0", "sex0")
PASSED: validateUserTestWithProviderParamDefault("admin1", "sex1")
PASSED: validateUserTestWithProviderParamDefault("admin2", "sex2")
PASSED: validateUserTestWithProviderParamDefault("admin3", "sex3")
PASSED: validateUserTestWithProviderParamTwo("admin", "man")
PASSED: validateUserTestWithProviderParamTwo("admin2", "women")

===============================================
    Default test
    Tests run: 6, Failures: 0, Skips: 0
===============================================


===============================================
Default suite
Total tests run: 6, Passes: 6, Failures: 0, Skips: 0
===============================================
```

如此可以看出，返回iterator和返回二维数组可以等效。
又因为可以延迟到next的时候初始化，所以可以做到延迟初始化对象，方便达到可以让他们使用完就可以被回收(只是被而已，不见得什么时候回收)。

四、两种方式的比较

方式	优点	缺点
DataProvider	可以向方法传递任何有效的java类型;非常灵活; 可以通过java代码动态算出或从存储机制里取出来	实现需要一些逻辑，才能 返回正确的对象
testng	值在xml文件中确定，易于修改，不需要重新编译; 值通过testNg自动传递给测试方法，不需要整理或转换	需要一个xml文件，不能动态算值; 文件中的值只能代表基本类型，不能是复杂的对象

总之，简单传递就利用xml是很好的，当我们需要更多的灵活性并知道参数的数目和值将随时间增加的时候，应该倾向于DataProvider

五、 数据的提供

DP提供数据的方式有许多种，以下列举了一些,简单说下优点和不足:

数据位置	优点	缺点
硬编码在Java源码	开发者很易于修改和解析	修改触发重新编译; 除了开发者之外其他人接触、理解、 修改比较难
文本中 (如逗号分隔的值)	1. 开发者想对容易解析; 2. 不需要重新编译(以下几种都有这种有点)易出错， 文本文件容易用任何编辑器修改， 每个人都易于修改点结果就是容易冲突和可控性较差，很容易破坏格式; 3. 需要让该文件的持有者共享该文件 （下面的几种除了数据库之外都有这种限制） 实现方式有以下几种: 共享的网络文件系统、git或者svn等版本控制系统， 但会产生一定的风险导致数据丢失或者被盗取	
属性文件中	开发者可方便将其转换成java的Properties对象	同上; 结构有局限：只能保存键值对；无序
Excel中	开发者容易使用三方库解析；非开发者容易操作; 容易确保数据的正确性(宏、或者其他机制来验证)	同上;需要安装excel才能读取， 因为它是私有的二进制格式， 当然也可以转为csv绕过这个问题， 但相对就丢失了excel的一些元数据
DB中	高度结构化而且灵活; 可以保存任何类型并可以通过复杂的sql来读取; SQL和JDBC是JDK本身自带的，文档比较丰富; 可以将数据库放到网络上	对于程序员来说， 需要更加复杂的程序来读取和解析， 但可以用框架来减轻负担如hibernate和mybatis、jpa等等; 向数据库插入新数据的时候， 需要有个前端界面， 特别是非开发者需要录入的时候; 测试需要额外的开销， 需要填充数据到数据库才能测试
网络中	抽象化数据存储的真正方式; 可以来源于隔壁的计算机或者横跨半个国家和地球而来自某一个数据中心l; 也可以使用许多协议比如tcp/ip、jms等等	需要复杂的编程逻辑以及额外的设置和配置

六、数据提供者还是工厂，如何选择?

在之前的一篇文章中，我们提到了工厂，本篇中着重提到了数据驱动，那它们都可以向测试提供参数：数据提供者向测试方法传递参数，而工厂向构造方法传递参数。
显然，可以使用数据提供者的地方，都可以使用工厂，反之亦然。

如果不确定怎么使用，那就仔细看一下分析一下测试方法的参数。是不是有几个测试方法需要同样的参数？如果是，那就最好保存到一个字段中然后在几个方法中复用字段，这意味着选择工厂比较合适。反之，如果所有的测试方法都需要不同的参数，那么DP显然是好一些。

当然，也可以两者同时使用。这会根据具体的测试场景决定，最终还是需要使用者本人来决定。