TestNg 介绍(二)

数据驱动特点

数据驱动是指在脚本固定的情况下，根据数据的条数来决定脚本的运行次数，即有几组数据，脚本就会运行几遍。

数据驱动概述

数据驱动(Data Driven)，这里强调的是数据，驱动即执行，那么数据驱动就是根据数据来执行测试脚本。

- 场景：测试登录，分别用刘能和赵四的帐号去测试。

先写一个公共方法来描述登录的过程：(伪代码实现)

```
public boolean login(String username, String password){
    //do login
}
```

再到测试方法里面去调用这个 login 方法/函数：

```
public void test1(){
    login("liuneng","123456");
}

public void test2(){
    login("zhaosi","654321");
}
```

这样测试用例就写完了，执行 test1 与 test2 两个方法即可。但细心的你可能会发现 test1 与 test2 这两个测试方法里的方法体除了数据，其它完全一样，这就存在重构的空间了：

```
public boolean login(String[][] accounts){
    for(int i = 0; i<accounts.length; i++){
        //do login
    }
}

public void test(){
    String[][] accounts = [["liuneng","123456"],["zhaosi","654321"]];
    login(accounts);
}
```

经过重构后的代码，就有点数据驱动的意思了，根据 accounts 的 length 来决定 login 方法/函数体运行几次，这样维护起来就方便了，假如又有一个老王的帐号想用来测试，就不需要再加一个测试方法了，只需要：

```
String[][] accounts = [["liuneng","123456"],["zhaosi","654321"],["laowang","000000"]];
```

重构后的代码，是不是令你很激动？原来这就是数据驱动！别急，淡定，这还不是真的数据驱动，因为上面只有一个测试方法，最后执行完后，报告中记录的也是只有一个测试方法，而场景中：分别用刘能和赵四的帐号去测试，是希望在测试报告中有两个测试方法出现，显然上面的代码还不能满足我们的需求，于是进一步优化：

```
public boolean login(String username, String password){
    //do login
}

public void test(String username, String password){
    login(username,password);
}
```

```
public void executor(){
    String[][] accounts = [["liuneng","123456"],["zhaosi","654321"]];
    for(int i = 0; i<accounts.length; i++){
        test(accounts[i][0],accounts[i][1]);
    }
}
```

是的，离数据驱动原理真相越来越近了，上面多了个 executor 方法，这是啥？这就是车子的发动机引擎啊，就是测试脚本的执行引擎，让测试方法能够被执行起来，以及根据你所提供的测试数据的条数，决定测试方法的执行次数，并且报告中会显示是两个测试方法，这就是数据驱动。测试框架就是一个执行引擎，并且测试框架都会支持数据驱动这一基本诉求，比如 TestNg 里的 dataProvider，junit 里的 Parameters 等。下面将为大家介绍 TestNg 里的 dataProvider 的用法。

TestNg 数据驱动

TestNg 的数据驱动也是以注解的形式来表达的：

```
public class TestData {

    @DataProvider(name="dataDemo")
    public Object[][] dataProvider(){
        return new Object[][]{{1,2},{3,4}};
    }

    @Test(dataProvider="dataDemo")
    public void testDemo(int a, int b){
        int sum = a + b;
        System.out.println("this is sum: "+sum);
    }

}
```

说明：

1. TestNg 的数据驱动的提供数据的方法用@DataProvider 注解
2. @DataProvider 中的 name 属性表示该数据源的名称
3. @DataProvider 的方法要返回一个 Object[][]的二维数组，当然也可以是 Iterator<Object[]>的数据结构。
4. 在测试方法中要用到数据源，则要在@Test 中加上 dataProvider 属性，其值为数据源的名称。
5. Object[][]二维数组{{1,2},{3,4}}可以理解为有两组数据，分别为：{1,2}，{3,4}，所以测试方法会运行两次，每一次运行时，测试方法的第一个参数 a 对应这一组数据中的第一个，也就是 1 或者 3，第二个参数 b 则对应这一组数据中的第二个，也就是 2 或者 4，如果还有数据，则依次类推。当然数据源的每一组数据的个数要大于或等于测试方法的参数个数，否则会报错，且数据类型要对应上，否则也会报错。

上面的示例中我们指定了@DataProvider 数据源的名称为 dataDemo，其实也可以不指定其名称，如果不指定其名称，则数据源的名称为该数据源方法的方法名，比如：

```
public class TestData {

    @DataProvider
    public Object[][] dataProvider(){
        return new Object[][]{{1,2},{3,4}};
    }
```

```
    @Test(dataProvider="dataProvider")
    public void testDemo(int a, int b){
        int sum = a + b;
        System.out.println("this is sum: "+sum);
    }

}
```

上面的例子中，@DataProvider 数据源的方法与测试方法是在同一个测试类里，或者@DataProvider 数据源的方法放在测试类的父类中。但其实还有一种方式，@DataProvider 数据源的方法可以单独的放在一个类里，比如：

```
public class DataSource {

    @DataProvider
    public static Object[][] dataProvider(){
        return new Object[][]{{1,2},{3,4}};
    }

}

public class TestData {

    @Test(dataProvider="dataProvider",dataProviderClass=DataSou
rce.class)
    public void testDemo(int a, int b){
        int sum = a + b;
        System.out.println("this is sum: "+sum);
    }

}
```

说明：

1. 在测试方法中可以指定一个数据源的类，用 dataProviderClass 来指定
2. 如果用了 dataProviderClass 指定数据源的类，则@DataProvider 数据源的方法必须是 static 的。

其实@DataProvider 数据源的方法，还可以提供一个参数 Method，这个 Method 是指要使用该数据源的测试方法的 Method 对象，比如：

```
public class TestData {

    @DataProvider
    public Object[][] dataProvider(Method method){
        //method对象指使用该数据源的测试方法的Method对象，这是java中的一
种反射
        System.out.println(method.getName());//输出testDemo
        return new Object[][]{{1,2},{3,4}};
    }

    @Test(dataProvider="dataProvider")
    public void testDemo(int a, int b){
        int sum = a + b;
        System.out.println("this is sum: "+sum);
    }

}
```

很显然，有了这个 Method 对象后，就很方便我们扩展了，请看下面的例子：
先写一个类，将所有的数据都放在里面：

```
public class DataSource {
    /**
     * dataMap里有两个数据源，分别是testDemo与testDemo1
     * 根据测试方法的名称，来使用不同的数据源。
     * 比如testDemo方法就用数据源{{1,2},{3,4}}
     * testDemo1方法就用数据源{{5,6},{3,4}}
     * @return
     */
    public Map<String, Object[][]> dataSource(){
        Map<String, Object[][]> dataMap = new HashMap<String,
Object[][]>();
        Object[][] o1 = new Object[][]{{1,2},{3,4}};
        dataMap.put("testDemo", o1);
        Object[][] o2 = new Object[][]{{5,6},{7,8}};
        dataMap.put("testDemo1", o2);
        return dataMap;
    }

}
```

再结合到@DataProvider 数据源方法与测试方法中去：

```
public class TestData {

    @DataProvider
    public Object[][] dataProvider(Method method){
        DataSource data = new DataSource();
        Object[][] obj = data.dataSource().get(method.getName());
        return obj;
    }

    @Test(dataProvider="dataProvider")
    public void testDemo(int a, int b){
        int sum = a + b;
        System.out.println("this is sum: "+sum);
    }

    @Test(dataProvider="dataProvider")
    public void testDemo1(int a, int b){
        int sum = a + b;
        System.out.println("this is sum: "+sum);
    }

}
```

以上两个测试方法用到了同一个数据源@DataProvider，这样为我们以后写测试框架定下了基调。

关于@Parameters

这里所说的@Parameters 就 TestNg 的@Parameters，在前面我们介绍过用 TestNg 的 xml 配置文件来制定策略执行测试类，那 xml 配置文件与测试方法间如何进行参数传递？@Parameters 就是干这个事的，所以，@Parameters 只是参数的传递，并不是真正意义上的数据驱动，我个人对这个应用的不多，但还是有必要给大家介绍一下用法。

```
public class TestDemo {

    @Parameters({"a","b"})
    @Test
    public void testDemo(int a, int b){
        int sum = a+b;
        System.out.println("this is sum: "+sum); //输出3
    }

}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<suite name="Suite" verbose="1" parallel="false" thread-count="1">
    <parameter name="a"  value="1"/>
    <parameter name="b"  value="2"/>
    <test name="Test1">
        <classes>
            <class name="com.test.demo.TestDemo" />
        </classes>
    </test>
</suite>
```

说明：

1. @Parameters 注解能用在测试方法或者@Before/@After/@Factory 上面。
2. @Parameters 中的字符串是在 xml 中的 parameter 结点的 name。
3. 在 xml 中的 parameter 结点的 value 就是该 name 对应的具体的参数值。
4. @Parameters 注解的{}中的参数的顺序，对应其标注的方法上的参数的顺序。