Junit3.8.2设计模式浅谈之命令模式

作者：changming
原文链接：https://ld246.com/article/1471155464492
来源网站：链滴

命令模式(Command)
对于客户端来说，要执行一次请求或者命令，不是直接执行，而是把这次请求或命令封装成一个求命令对象，该命令对象中包含一个execute方法，然后由命令执行端来执行此方法；这种方式可以命令的客户端与命令的执行端有效的解偶，方便后面自由添加请求客户端，而不用关注请求执行端。
举个例子： 我们去超市买东西，一般是自己走进超市自己选好商品后，付款搞定，买东西这个求始终是自己来完成；现在，我们让一个管家去完成买东西这个请求，我们只要和他说去买即可，管是不会关心我们具体要买的东西的，比如，我把买衣服封装成一个命令对象，该对象有个方法叫做execute，方法内容是买衣服，而不是买鞋子；同理我们可以再把买牛奶封装成同样具有execute方法的一命令对象，这些命令对象的execute方法的执行全部由这个管家来执行，这个就叫做Command模式
有什么好处，管家可以对所有买的东西进行日志记录，可以进行排序，谁先谁后，同时还可以定那个不买，这个是高级功能了。
在Junit框架中，执行程序员写的测试用例就是一次请求，整个Junit测试框架就是一个命令执行，也就是那个管家，每次执行测试用例都会被封装成一个命令(测试用例)，然后由测试框架执行，这有效的解偶了请求和请求的执行。
Test就是所有封装的命令对象的接口，代码如下：
<pre class="brush: java">public interface Test {
    /**
     * Counts the number of test cases that will be run by this test.
     */
    public abstract int countTestCases();
    /**
     * Runs a test and collects its result in a TestResult instance.
     */
    public abstract void run(TestResult result);
}</pre>
其中run方法就是execute方法，管家执行的就是这个方法，在Junit中，每个测试用例被封装成命令对象就是TestSuit，它继承了Test接口。
<pre class="brush: java">public class TestSuite implements Test {
        /**
     * Runs the tests and collects their result in a TestResult.
     */
    public void run(TestResult result) {
        for (Enumeration e= tests(); e.hasMoreElements();) {
            if (result.shouldStop())
                break;
            Test test= (Test)e.nextElement();
            runTest(test, result);
        }
    }

public void runTest(Test test, TestResult result) {
    test.run(result);
}

    /**
 * Counts the number of test cases that will be run by this test.
 */
public int countTestCases() {
    int count= 0;
    for (Enumeration e= tests(); e.hasMoreElements();) {
        Test test= (Test)e.nextElement();
        count= count + test.countTestCases();

```
    }
    return count;
}

}</pre>

<p><br />同时Junit中的管家就是各种客户端实现中的TestRunner类，由它来驱动封装好的命令的行。<br /><br /></p>
<pre class="brush: java">public class TestRunner extends BaseTestRunner {
        public TestResult doRun(Test suite, boolean wait) {
        TestResult result= createTestResult();
        result.addListener(fPrinter);
        long startTime= System.currentTimeMillis();
        suite.run(result);
        long endTime= System.currentTimeMillis();
        long runTime= endTime-startTime;
        fPrinter.print(result, runTime);


    pause(wait);
    return result;
}

}</pre>

<p><br /><br />其中suite.run(result);方法不就是我们每个封装好的命令对象中的execute方法吗<br />Command模式的实现，方便了测试人员编写各种不同的测试用例，而无须修改原有的类。下节我们来说说这其中的适配器模式，也很精彩！</p>
```