第24章

命令（Command）模式

直接调用是执行方法的一般方式。然而，有时我们无法控制方法执行的时机与上下文。这种情况下，可以将方法封装在对象的内部。通过在对象内部存储调用方法所需要的信息，就可以让客户端或者服务决定何时调用该方法。

命令模式的意图是将请求封装在对象内部。

经典范例：菜单命令

支持菜单的工具包通常都使用了命令模式。每个菜单项关联一个对象，以便在用户单击该菜单项时，可以执行相关的行为。这种设计分离了 GUI 逻辑与应用逻辑。Swing 库就采用了这种模式，允许你为每个 `JMenuItem` 关联一个 `ActionListener`。

当用户单击菜单时，该如何让类调用相关的方法呢？答案是使用多态：固定操作的名称，针对不同的类给出同名方法的不同实现。对 `JMenuItem` 而言，方法名是 `actionPerformed()`，当用户选择任意一个菜单项时，`JMenuItem` 就去调用注册到侦听器上任意对象的 `actionPerformed()`方法。

挑战 24.1

Java 的菜单机制使你可以方便地使用命令模式，而并不要求你将自己的代码组织为命令模式。事实上，在开发过程中，用一个对象监听 GUI 中的所有事件是很常见的。请问这用的是哪种模式?

答案参见第 352 页

当在开发 Swing 应用时，可能会为所有的 GUI 事件注册同一个侦听器对象，尤其是在 GUI 组件交互时。然而，对于菜单而言，这并非好的选择。倘若让一个单独的对象侦听所有的菜单事件，当 GUI 对象发出事件时，就需要对这些事件进行筛选。相反，如果让每个菜单项都用独立的对象来处理事件，那么最好使用命令模式。

当用户选择菜单项时，会调用 actionPerformed()方法。在创建菜单项时，可以使用指定了命令行为的 actionPerformed()方法，为其绑定一个 ActionListener。相对于定义一个新类来实现这个小功能，使用匿名类是更好的选择。

考虑 com.oozinoz.visualization 包中的 Visualization2 类。该类提供了一个菜单栏，包含了文件菜单，使得用户可以保存或者恢复模拟 Oozinoz 工厂的效果。该菜单包含两个菜单项，分别是 Save As...和 Restore From. ...。它们都注册了侦听器对用户事件进行侦听。这两个侦听器分别调用 Visualization2 类的 save()和 load()方法，来实现 actionPerformed()方法：

```
package com.oozinoz.visualization;
import java.awt.event.*;
import javax.swing.*;
import com.oozinoz.ui.*;
public class Visualization2 extends Visualization {
    public static void main(String[] args) {
        Visualization2 panel = new Visualization2(UI.NORMAL);
        JFrame frame =
            SwingFacade.launch(panel, "Operational Model");
        frame.setJMenuBar(panel.menus());
        frame.setVisible(true);
    }

    public Visualization2(UI ui) {
        super(ui);
```

```
}

public JMenuBar menus() {
    JMenuBar menuBar = new JMenuBar();

    JMenu menu = new JMenu("File");
    menuBar.add(menu);

    JMenuItem menuItem = new JMenuItem("Save As...");
    menuItem.addActionListener(new ActionListener() {
        // 挑战!
    });
    menu.add(menuItem);

    menuItem = new JMenuItem("Restore From...");
    menuItem.addActionListener(new ActionListener() {
        // 挑战!
    });
    menu.add(menuItem);

    return menuBar;
}

public void save() { /* 忽略 */ }
public void restore() { /* 忽略 */ }
}
```

挑战 24.2

请写出匿名类 ActionListener 的 actionPerformed()方法。注意，该方法需要一个 ActionEvent 参数。

答案参见第 352 页

当使用命令模式来装配菜单时，应将这些命令应用到其他开发者提供的上下文，即 Java 菜单框架中。在某些命令模式的应用场合，需要自己提供命令执行的上下文。例如，你可能需要提供一个时间服务，它可以记录方法执行的时间。

使用命令模式来提供服务

假设你想让开发者统计某个方法执行的时间。现在，我们有一个 Command 接口，实现如下：

```
public abstract void execute();
```

然后，定义一个 CommandTimer 类：

```
package com.oozinoz.utility;

import com.oozinoz.robotInterpreter.Command;

public class CommandTimer {
    public static long time(Command command) {
        long t1 = System.currentTimeMillis();
        command.execute();
        long t2 = System.currentTimeMillis();
        return t2 - t1;
    }
}
```

可以使用 JUnit 测试 time()方法，如下所示。注意，这并非一个精确的测试，如果计时器是“jittery”，测试就会失败。

```
package app.command;

import com.oozinoz.robotInterpreter.Command;
import com.oozinoz.utility.CommandTimer;

import junit.framework.TestCase;
public class TestCommandTimer extends TestCase {
    public void testSleep() {
        Command doze = new Command() {
            public void execute() {
                try {
                    Thread.sleep(
                        2000 + Math.round(10 * Math.random()));
                } catch (InterruptedException ignored) {
```

```
                }
            }
        };

        long actual = // 挑战!

        long expected = 2000;
        long delta = 5;
        assertTrue(
            "Should be " + expected + " +/- " + delta + " ms",
            expected - delta <= actual
        && actual<= expected + delta);
    }
}
```

挑战 24.3

请写出上面用于统计 doze 命令执行时间的赋值语句。

答案参见第 354 页

命令钩子

第 21 章的模板方法模式中介绍了 Aster 火药填压机：一个应用模板方法的智能机器。如果机器关闭时，当前模具还未处理完，则填压机中的代码会自动将当前模具设置为未完成。

AsterStarPress 类是一个抽象类，需要定义一个子类来重写其 markMoldIncomplete() 方法。其中，shutDown()方法依赖于该方法，用于确保当机器关闭时，让领域对象知道哪些模具未完成。

```
public void shutdown() {
    if (inProcess()) {
        stopProcessing();
        markMoldIncomplete(currentMoldID);
    }
```

```
        usherInputMolds();
        dischargePaste(
        flush();
    }
```

当需要将某些类移植到火药填压机的板载计算机上时，创建 AsterStarPress 类的子类会显得很不方便。假定你要求 Aster 公司的程序员使用命令模式来重新提供一个钩子。图 24.1 展示了 AsterStarPress 类可以使用的 Hook 命令，使你可以在运行时对火药填压机的代码进行参数化。

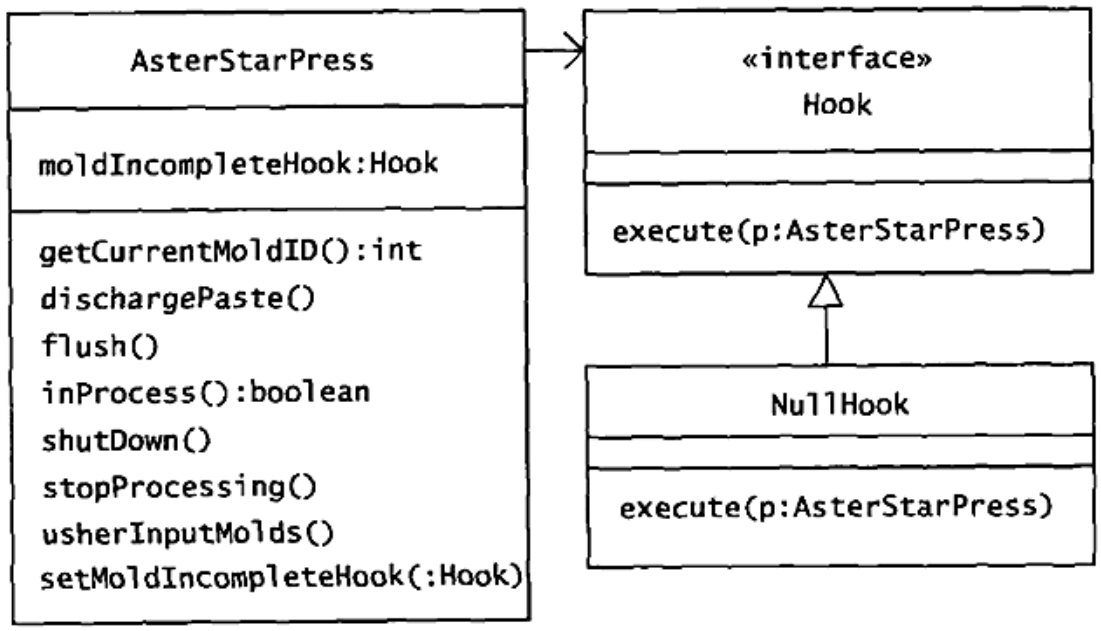

图 24.1　该类可以提供一个钩子——一种插入自定义代码的方式，通过程序执行时在一个指定点来调用支持的命令

在原始的 AsterStarPress 类中，shutDown()方法依赖其子类提供的某个步骤。在新的设计中，shutDown()方法在将停未停之时，会使用钩子来执行客户代码。

```
public void shutDown() {
    if (inProcess()) {
        stopProcessing();
        // 挑战!
    }
    usherInputMolds();
    dischargePaste();
    flush();
}
```

挑战 24.4

请写出新 shutDown()方法的实现。

答案参见第 355 页

这个例子展示了另一种模式——Null Object 模式[译注1]，该模式没有 *Design Patterns* 一书中提到的模式那么知名。它通过引入一个默认的对象来避免空指针的检查。请参考 *Refactoring*（由 Fowler 等人在 1999 年编写）一书，书中介绍了如何在代码中引入该模式。

命令模式提供了除模板方法模式外的另一种设计钩子的方法，它在意图和结构上也和其他几种模式相似。

命令模式与其他模式的关系

命令模式类似于解释器模式。下一章我们会比较这两个模式。命令模式还和另一个模式很相似，两者都是知道创建对象的时间，但都不知道应该创建哪个对象。

挑战 24.5

哪个模式描述了这种情况：知道何时去创建对象，但却不知道需要实例化哪个类？

答案参见第 356 页

命令模式除了与某些模式具有异曲同工之妙，它也常常会与其他模式协作。例如，在 MVC 设计中可以将命令模式与调停者模式混合使用。第 19 章的备忘录模式就给出了这个例子。Visualization 类处理 GUI 控制逻辑，任何与模型相关的逻辑都由调停者来处理。例如，Visualization 类用如下代码来延迟初始化 Undo 按钮：

译注1：即空对象模式，由 Bobby Woolf 提出。该模式用于处理判断对象为 null 的情况。它为指定对象提供了一个特定对象，用以替换指定对象为 null 的情况。引入 Null Object 模式，可以消除判断对象是否为 null 的语句。

```
protected JButton undoButton() {
    if (undoButton == null) {
        undoButton = ui.createButtonCancel();
        undoButton.setText("Undo");
        undoButton.setEnabled(false);
        undoButton.addActionListener(mediator.undoAction());
    }
    return undoButton;
}
```

上面的代码使用了命令模式，将 undo()方法封装在 ActionListener 类的实例中。该代码也使用了调停者模式，让中心对象去协调与底层模型相关的事件。为了让 undo()方法生效，调停者对象必须恢复模拟工厂的前一个版本，这正是将命令模式与其他模式混合使用的最好时机。

挑战 24.6

哪个模式可以实现对象状态的存储与恢复？

答案参见第 356 页

小结

命令模式可以将请求封装在一个对象中，允许你可以像管理对象一样去管理方法，传递并且在合适的时机去调用它们。菜单是应用命令模式的一个经典案例。菜单项知道何时执行一个请求，但却不知道应该去执行什么请求。命令模式可以让你使用与菜单标签相关的方法调用作为参数，传递给一个菜单。

命令模式的另一个用处是，允许在服务执行的上下文中执行客户类代码。服务经常在调用客户代码的前后执行。最后，除了可以控制方法的执行时间和上下文外，命令模式还可以提供一个钩子机制，允许客户代码作为算法执行的一部分。

命令模式将请求封装在了对象中，因此你可以像操作对象那样来操作请求。或许是因为这种想法如此普遍，通常命令模式会和其他模式一起使用。例如，命令模式可以作为模板方法模式的一种替代模式，它也经常和调停者模式与备忘录模式配合使用。