|22.1 command模式

【命令也是类】

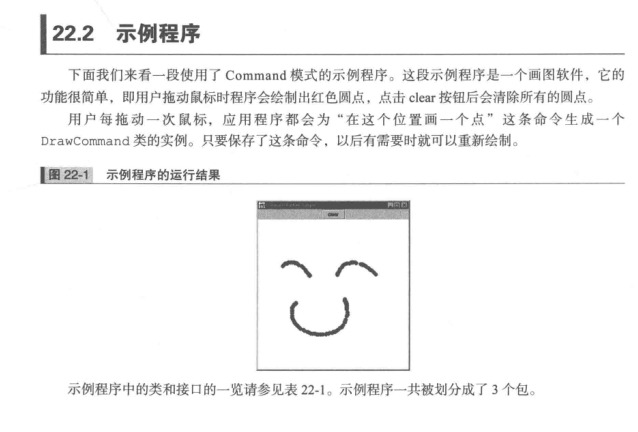
一个类在进行工作时会调用自己或是其他类的方法，虽然调用结果会反映在对象的状态中，但并不会留下工作的历史记录。

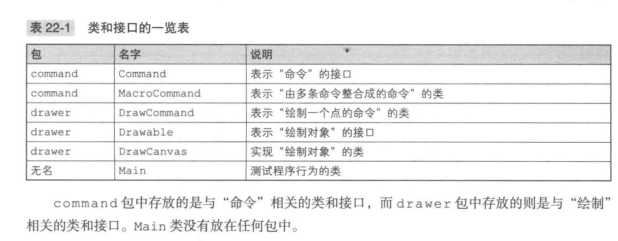
这时，如果我们有一个类，用来表示“请进行这项工作”的“命令”就会方便很多。每一项想做的工作就不再是“方法的调用”这种动态处理了，而是一个表示命令的类的实例，即可以用“物”来表示。要想管理工作的历史记录，只需管理这些实例的集合即可，而且还可以随时再次执行过去的命令，或是将多个过去的命令整合为一个新命令并执行。

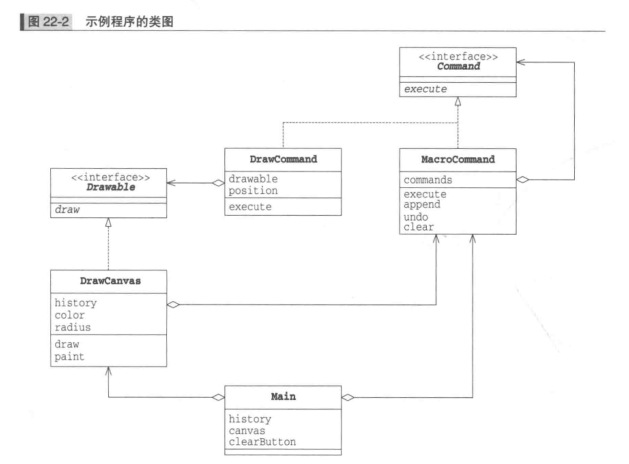
在设计模式中，我们称这样的“命令”为Command模式（command有“命令”的意思）。

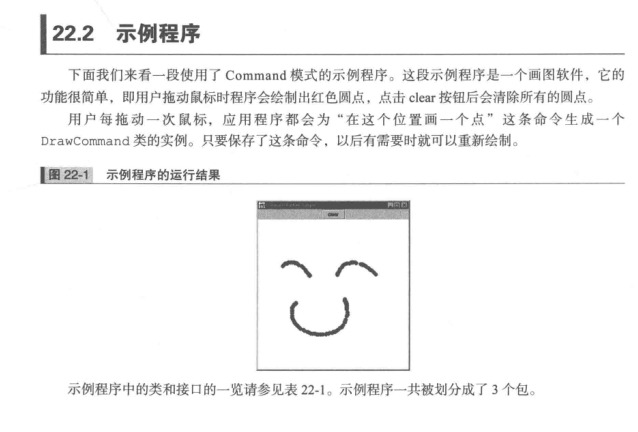
Command有时也被称为事件（event）。它与“事件驱动编程”中的“事件”是一样的意思。当发生点击鼠标、按下键盘按键等事件时，我们可以先将这些事件作成实例，然后按照发生顺序放入队列中。接着，再依次去处理它们。在GUI（graphical user interface）编程中，经常需要与“事件”打交道。

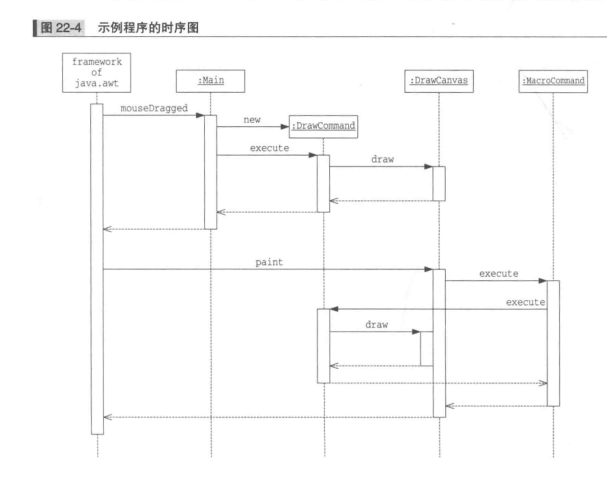
在本章中，我们将学习与“命令”打交道的Command模式。











|22.3 Command模式中的登场角色

在Command模式中有以下登场角色。

·Command（命令）Command角色负责定义命令的接口（API）。在示例程序中，由Command接口扮演此角色。

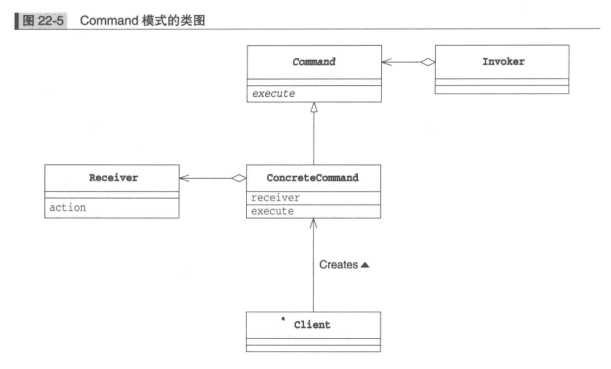
◆ConcreteCommand（具体的命令）ConcreteCommand角色负责实现在Command角色中定义的接口（API）。在示例程序中，由MacroCommand类和DrawCommand类扮演此角色。

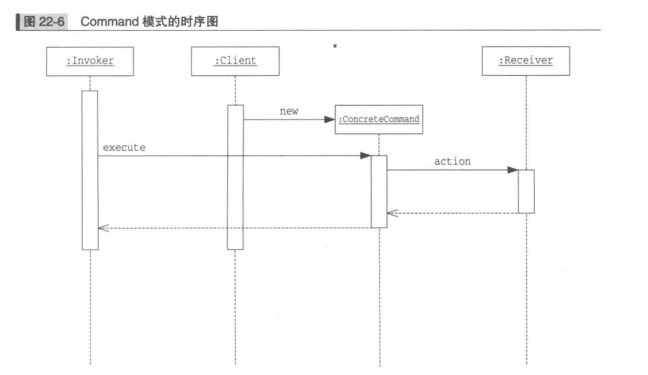
◆Receiver（接收者）Receiver角色是Command角色执行命令时的对象，也可以称其为命令接收者。在示例程序中，由DrawCanvas类接收Drawcommand的命令。

·Client（请求者）Client角色负责生成ConcreteCommand角色并分配Receiver角色。在示例程序中，由Main类扮演此角色。在响应鼠标拖拽事件时，它生成了Drawcommand类的实例，并将扮演Receiver角色的DrawCanvas类的实例传递给了DrawCommand类的构造函数。

◆Invoker（发动者）Invoker角色是开始执行命令的角色，它会调用在Command角色中定义的接口（APl）。在示例程序中，由Main类和Drawcanvas类扮演此角色。这两个类都调用了Command接口中的execute方法。Main类同时扮演了Client角色和Invoker角色。

Command模式的类图如图22-5所示，时序图如图22-6所示。





|22.4拓展思路的要点

命令中应该包含哪些信息

关于“命令”中应该包含哪些信息这个问题，其实并没有绝对的答案。命令的目的不同，应该包含的信息也不同。Drawcommand类中包含了要绘制的点的位置信息，但不包含点的大小、颜色和形状等信息。

假设我们在DrawCommand类中保存了“事件发生的时间戳”，那么当重新绘制时，不仅可以正确地画出图形，可能还可以重现出用户鼠标操作的缓急。

在DrawCommand类中还有表示绘制对象的drawab1e字段。在示例程序中，由于只有一个DrawCanvas的实例，所有的绘制都是在它上面进行的，所以这个drawable字段暂时没有太大意义。但是，当程序中存在多个绘制对象（即Receiver角色）时，这个字段就可以发挥作用了。这是因为只要ConcreteCommand角色自己“知道”Receiver角色，不论谁来管理或是持有ConcreteCommand角色，都是可以执行execute方法的。

|保存历史记录

在示例程序中，Macrocommand类的实例（history）代表了绘制的历史记录。在该字段中保存了之前所有的绘制信息。也就是说，如果我们将它保存为文件，就可以永久保存历史记录。

|适配器

示例程序的Main类（代码清单22-6）实现了3个接口，但是并没有使用这些接口中的全部方法。例如MouseMotionListener接口中的以下方法。

public void mouseMoved（MouseEvent e）

public void mouseDragged（MouseEvente）

在这两个方法中，我们只用到了mouseDragged方法。

再例如，windowlistener接口中的以下方法。

publlc void windowClosing（WindowEvent e）

public void windowActivated（WindowEvent e）

public void windowclosed（WindowEvent e）

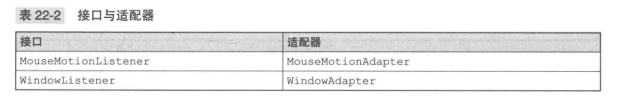
public void windowDeactivated（WindowEvent e）public void windowDeiconified（WindowEvent e）

publtc void windowIconified（WindowEvent e）

public void windowopened（WindowEvent e）

在这7个方法中，我们仅用到了windowclosing方法。

为了简化程序，java.awt.event包为我们提供了一些被称为适配器（Adapter）的类。例如，对于MouseMotionListener接口有MouseMotionAdapter类；对WindowListener接口有WindowAdapter类（表22-2）。这些适配器也是Adapter模式（第2章）的一种应用。



这里，我们以MouseMotionAdapter为例进行学习。该类实现了MouseMotionListener接口，即实现了在该接口中定义的所有方法。不过，所有的实现都是空（即什么都不做）的。因此，我们只要编写一个MouseMotionAdapter类的子类，然后实现所需要的方法即可，而不必在意其他不需要的方法。

特别是把Java匿名内部类（anonymous inner alass）与适配器结合起来使用时，可以更轻松地编写程序。请大家对比以下两段代码，一个是使用了接口MouseMotionListener的示例代码（代码清单22-7），另一个是使用了内部类MouseMotionAdapter的示例代码（代码清单22-8）。请注意，这里省略了其中的细节代码。





如果大家不熟悉内部类的语法，可能难以理解上面的代码。不过，我们仔细看一下代码清单22-8中的代码就会发现如下特点。

·new MouseMotionAdapter（）这里的代码与生成实例的代码类似

·之后的{.…}部分与类定义（方法的定义）相似

其实这里是编写了一个MouseMotionAdapter类的子类（匿名），然后生成了它的实例。请注意这里只需要重写所需的方法即可，其他什么都不用写。

另外需要说明的是，在编译匿名内部类时，生成的类文件的名字会像下面这样，其命名规则是

“主类名$编号.class”。

Main51.class在习题22-3中，请各位自己修改示例程序，练习如何使用MouseMotionAdapter类和WindowAdapter类。

|22.5相关的设计模式

·Composite模式（第11章）有时会使用Composite模式实现宏命令（macrocommand）。

272|第22章Command模式

◆Memento模式（第18章）有时会使用Memento模式来保存Command角色的历史记录。

◆Protype模式（第6章）有时会使用Protype模式复制发生的事件（生成的命令）。