[置顶]设计模式是什么鬼（命令模式）

凸凹 设计模式 2018年12月30日 468 0 0

命令模式，通常指的是一个对象向另一个对象发送信息指令的行为模型，比如父母命令孩子写作业、将军命令士兵进攻等。我们经过分析拆解方法会得到三个模块，首先得有命令发送方，接着是被传递的命令本身，最后就是命令的接收执行方了。那么，这样拆解到底有什么好处？让我们先来看一个最简单的例子，电灯泡。

既然是电灯那一定对应通电和断电的行为接口了，两个接口方法互斥，我们就叫它Switchable吧。

```
public interface Switchable {//电器接口
    //通电
    public void on();
    //断电
    public void off();

}
```

对于具体的灯泡实现类，必然是通电亮，断电灭。

```
public class Bulb implements Switchable {

    @Override
    public void on(){
        System.out.println("通电，灯亮。");
    }

    @Override
    public void off(){
        System.out.println("断电，灯灭。");
    }

}
```

同样地，我们再增加一个设备，如果是风扇的话则是通电转，断电停。

```
public class Fan implements Switchable{
```

凸凹 官方
关注 私信
凸凹里歐，十余年以上开发经验，《设计
列文章作者，旨在用最通俗的方式诠释设

最新文章
设计模式是什么鬼（命令模式）
设计模式是什么鬼（建造者）
设计模式是什么鬼（抽象工厂）
设计模式是什么鬼（工厂方法）
设计模式是什么鬼（桥接）

极客快讯
Html2excel 1.3.0 版本发布，POI 新
2018年12月30日 36
Apache NetBeans 10.0 正式发布，
2018年12月30日 28
近期国际油价大跌，财报告诉你中石
2018年12月29日 70
“极速抢票”拼不过普通购票 春运抢
坑?
2018年12月29日 78
12月 Web 服务器调查：nginx 增长
份额最高
2018年12月29日 60

```
        System.out.println("通电，风扇转动。");
    }

    @Override
    public void off() {
        System.out.println("断电，风扇停止。");
    }

}
```

我们该如何操作呢？来吧，直接用电线接通电源。

```
public class Client {

    public static void main(String[] args) {

        System.out.println("===客户端用【电线】直接操作灯泡===");
        Bulb bulb = new Bulb();
        bulb.on();
        bulb.off();
        /*打印输出：
            ===客户端用【电线】直接操作灯泡===
            通电，灯亮。
            断电，灯灭。
        */
    }

}
```

也许用户是个糙人，直接用导线给通电了，简单粗暴，虽然没有错，但这看上去与设计模式没有任何瓜葛。为了体现出模式的优越性，我们需要让系统进化得更高级一些，于是我们决定加入另一个模块，开关控制。

```
public class Switcher {

    // 此开关与灯耦合，无法替换为其他电器。
    // private Bulb bulb = new Bulb();

    // 此开关与电器接口耦合，可任意替换电器。
    private Switchable switchable;

    // 替换电器方法
    public void setSwitchable(Switchable switchable) {
        this.switchable = switchable;
    }

    // 按键事件绑定

    // 按钮“开”按下
    public void buttonOnClick() {
        System.out.println("按下开.....");
        switchable.on();
    }

    // 按钮“关”按下
    public void buttonOffClick() {
        System.out.println("按下关.....");
        switchable.off();
    }

}
```

所以第7行我们声明的是Switchable接口引用，并提供第10行的替换电器方法给外界注入任何的设备。好了，我们换个方式运行程序。

```
public class Client {

    public static void main(String[] args) {

        System.out.println("===客户端用【开关】操作电器===");
        Switcher switcher = new Switcher();

        switcher.setSwitchable(new Bulb());//灯泡接入开关。
        switcher.buttonOnClick();
        switcher.buttonOffClick();
        switcher.setSwitchable(new Fan());//风扇接入开关。
        switcher.buttonOnClick();
        switcher.buttonOffClick();

        /*打印输出:
            ===客户端用【开关】操作电器===
            按下开.....
            通电，灯亮。
            按下关.....
            断电，灯灭。
            按下开.....
            通电，风扇转动。
            按下关.....
            断电，风扇停止。
         */
    }

}
```

这次看上去功能强大多了，开关可以随意地接入灯泡或者风扇，注入的是谁那么开关按钮直接就作用于谁，对于设备我们还可以继续扩展，设计模式开始体现优势了。等等，这个模式好像似曾相识的感觉，没错，这正是之前讲过的策略模式，可这跟命令模式有什么关系？不要着急，我们先看下这个开关策略模式是否满足了我们的需求。

假设我们的设备不断扩展，比如有了电视机，收音机等等设备，它们不止是开关通电这种简单行为模式了，还可以有转换频道、变音量等等更多的行为。

口“强耦合了，也就是说它只能控制“灯泡或风扇”，开不能控制“电视或收音机”。

同时另一端我们的开关控制器也在不断进化，发展出了更多功能控制器、无线遥控器、甚至是手机App控制。

所以，如何把控制器与设备完全给拆解开势在必行，此时命令模式粉墨登场。现在我们得新定义出一组”命令“模块把控制器（发令者）与设备（执行者）彻底解耦，就以电视机和遥控器举例说明吧。

```
1  public interface Device extends Switchable{
2
3    //频道+
4    public void channelUp();
5     //频道-
6    public void channelDown();
7    //音量+
8    public void volumeUp();
9    //音量-
10   public void volumeDown();
11
12 }
```

注意代码第1行的接口继承，我们的高级设备接口则遗传了之前的简单通断电接口，并新增了调节频道和音量4个功能。接下来是电视机与收音机实现类。

```
1  public class TV implements Device {
2
3    @Override
4    public void on(){
5       System.out.println("电视机启动");
6    }
7
8    @Override
9    public void off(){
10      System.out.println("电视机关闭");
11   }
12
13   @Override
14   public void channelUp() {
15      System.out.println("电视机频道+");
16   }
17
18   @Override
19   public void channelDown() {
20      System.out.println("电视机频道-");
21   }
22
23   @Override
24   public void volumeUp() {
25      System.out.println("电视机音量+");
```

```
    public void volumeDown() {
        System.out.println("电视机音量-");
    }
}
```

```
public class Radio implements Device {

    @Override
    public void on(){
        System.out.println("收音机启动");
    }

    @Override
    public void off(){
        System.out.println("收音机关闭");
    }

    @Override
    public void channelUp() {
        System.out.println("收音机调频+");
    }

    @Override
    public void channelDown() {
        System.out.println("收音机调频-");
    }

    @Override
    public void volumeUp() {
        System.out.println("收音机音量+");
    }

    @Override
    public void volumeDown() {
        System.out.println("收音机音量-");
    }
}
```

没什么好说的，下来是解耦的重点了，我们在策略模式的基础上又增加一层中间模块，开始编写命令模块代码，首先是命令接口。

```
public interface Command {

    //执行命令操作
    public void exe();

    //反执行命令操作
    public void unexe();

}
```

命令接口有执行操作与反执行操作两个标准功能，然后定义其命令实现类，开关机命令、频道转换命令、以及音量调节命令。

```
public class SwitchCommand implements Command {

    private Device device;// 此处持有高级设备接口。

    public SwitchCommand(Device device) {
        this.device = device;
```

```
    public void exe() {
        device.on();// 执行命令调用开机操作
    }

    @Override
    public void unexe() {
        device.off();// 反执行命令调用关机操作
    }

}
```

```
public class ChannelCommand implements Command{

    private Device device;

    public ChannelCommand(Device device) {
        this.device = device;
    }

    @Override
    public void exe() {
        device.channelUp();
    }

    @Override
    public void unexe() {
        device.channelDown();
    }

}
```

```
public class VolumeCommand implements Command{

    private Device device;

    public VolumeCommand(Device device) {
        this.device = device;
    }

    @Override
    public void exe() {
        device.volumeUp();
    }

    @Override
    public void unexe() {
        device.volumeDown();
    }

}
```

代码很简单，但是系统模组相对复杂，所以一定要搞清楚各模块间关系再继续。最后一个模块是遥控器类，也就是命令发送方了。我们保持简单，遥控器集成了OK按键以及上下左右方向键。

```
public class Controller {
    private Command okCommand;
    private Command verticalCommand;
    private Command horizontalCommand;

    // 绑定OK键命令
    public void bindOKCommand(Command okCommand) {
```

```
11    //绑定上下方向键命令
12    public void bindVerticalCommand(Command verticalCommand) {
13        this.verticalCommand = verticalCommand;
14    }
15
16    //绑定左右方向键命令
17    public void bindHorizontalCommand(Command horizontalCommand) {
18        this.horizontalCommand = horizontalCommand;
19    }
20
21    //开始按键映射命令
22    public void buttonOKHold() {
23        System.out.print("长按OK按键.....");
24        okCommand.exe();
25    }
26
27    public void buttonOKClick() {
28        System.out.print("单击OK按键.....");
29        okCommand.unexe();
30    }
31
32    public void buttonUpClick() {
33        System.out.print("单击↑按键.....");
34        verticalCommand.exe();
35    }
36
37    public void buttonDownClick() {
38        System.out.print("单击↓按键.....");
39        verticalCommand.unexe();
40    }
41
42    public void buttonLeftClick() {
43        System.out.print("单击←按键.....");
44        horizontalCommand.unexe();
45    }
46
47    public void buttonRightClick() {
48        System.out.print("单击→按键.....");
49        horizontalCommand.exe();
50    }
51 }
```

这个遥控器持有三个命令组件，并且于第7行开始定义命令绑定方法，最后从第22行开始定义各按键触发方法并映射到相应的命令操作上。可以看到，控制器对设备一无所知，也就是它上面不再绑定有任何设备了，而是只绑定命令。最后，客户端又换了一种方式运行程序。

```
1  public class Client {
2
3      public static void main(String[] args) {
4          System.out.println("===客户端用【可编程式遥控器】操作电器===");
5          Device tv = new TV();
6          Device radio = new Radio();
7          Controller controller = new Controller();
8
9          //绑定【电视机】的【命令】到【控制器按键】
10         controller.bindOKCommand(new SwitchCommand(tv));
11         controller.bindVerticalCommand(new ChannelCommand(tv));//上下调台
12         controller.bindHorizontalCommand(new VolumeCommand(tv));//左右调音
13
14         controller.buttonOKHold();
15         controller.buttonUpClick();
```

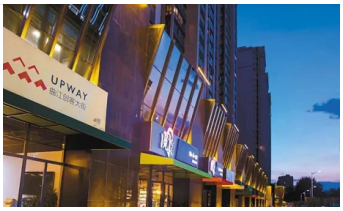
```
19
20      /*打印输出:
21          ===客户端用【可编程式遥控器】操作电器===
22          长按OK按键.....电视机启动
23          单击↑按键.....电视机频道+
24          单击↑按键.....电视机频道+
25          单击↓按键.....电视机频道-
26          单击→按键.....电视机音量+
27      */
28
29      //绑定【收音机】的【命令】到【控制器按键】
30      controller.bindOKCommand(new SwitchCommand(radio));
31      controller.bindVerticalCommand(new VolumeCommand(radio));//上下调音
32      controller.bindHorizontalCommand(new ChannelCommand(radio));//左右调台
33
34      controller.buttonOKHold();
35      controller.buttonUpClick();
36      controller.buttonUpClick();
37      controller.buttonRightClick();
38      controller.buttonDownClick();
39
40      /*打印输出:
41          长按OK按键.....收音机启动
42          单击↑按键.....收音机音量+
43          单击↑按键.....收音机音量+
44          单击→按键.....收音机调频+
45          单击↓按键.....收音机音量-
46      */
47
48  }
49
50 }
```

很显然，客户端可以肆意妄为地组装各个模块了，也就是说可以遥控电视，也可以遥控收音机，或许绑定上下键调音量，或许是换成左右键调音量，甚至可以定义一个宏命令去控制灯泡的切换开关实现一种霓虹灯闪烁的效果（读者可以思考怎样实现），而对于控制器端本身，同样可以继续扩展，或许干脆替换个游戏手柄或者键盘，一样可以发号施令。

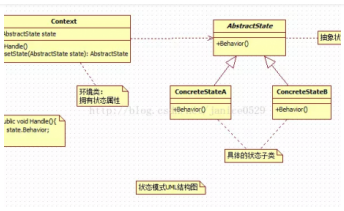
至此，发令控制方与接受执行方完全被拆解开，这让我们实现了对各模块的自由扩展，对指令映射、设备绑定的灵活操控，松散的系统得以成就繁多模块解耦的最终目的。

本文作者：凸凹

设计模式(61) 设计模式学习(1)



设计模式是什么鬼（门面）



23种设计模式介绍-状态模式

简说外观模式

简说工厂模式

简说装饰模式

23种设计模式介绍-工厂模式

发表评论

评分 ☆ ☆ ☆ ☆ ☆

您的评论一针见血（必填，该内容可在后台设置）

昵称 邮箱

网址

发表评论