我将带大家一起去看一看在面向对象编程中出现的23大设计模式和9大设计原则。设计原则是我们开发研究设计模式的目的，而设计模式则是达到设计原则的手段和方法。在记录笔记过程中，会在不同的设计模式中穿插讲解设计原则。
本笔记知识点主要来自于《Head First 设计模式》

行为型模式三：命令模式

定义：
将“请求”封装为对象，以便使用不同对请求来参数化其他对象。命令模式也支持可撤销的操作。

组成：

- Command: 请求抽象类，或者抽象接口。声明所有具体请求的执行方法。
- ConcreteCommand： 具体请求类，通常含有一个Receiver实例变量；并且调用接受者的方法来执行方法；
- Receiver： 接受者，真正请求执行的主体。
- Invoker： 请求的发送者，一般持有命令对象，然后调用方法来触发接受者的执行方法；
- Client：这是客户，创建具体的请求发送者和一系列的请求对象。

在上面的解释中，实质上发生请求的主体是Invoker，而执行请求的主体是Receiver.两者之间是用请求对象来进行交互，实际上请求发送者并不知道哪个接受者会执行命令，他只是负责发送指令而已。

优点：

- 将请求发送者和执行者解耦合
- 可以很容易将新请求加入到发送者中

缺点：

- 会存在大量的具体请求类

适用情况：

- 要求调用者和接受者解耦合；
- 系统需要队列指令，或者在不同时候执行指令；
- 系统需要撤销或者重做操作；
- 系统需要将一系列请求捆绑，即宏指令；

看下面的例子：

```
我们用命令模式来模拟遥控器控制电视机。
//首先有个Command接口
public interface Command {
    public void execute();
}
//开电视机指令
public class turnOnCommand implements Command {
    private Tv tv;
    public turnOnCommand(Tv tv) {
        this.tv = tv;
    }
    public void execute() {
        tv.turnOn();
    }
}
//关电视机指令
public class turnOffCommand implements Command {
    private Tv tv;
    public turnOffCommand(Tv tv) {
        this.tv = tv;
    }
    public void execute() {
        tv.turnOff();
    }
}
//换电视台
public class changeChannelCommand implements Command {
    private Tv tv;
    private int channel;
    public changeChannelCommand(Tv tv, int channel) {
        this.tv = tv;
        this.channel = channel;
    }
    public void execute() {
        tv.changeChannel(channel);
    }
}

//电视机是接受者
public class Tv {
    public void turnOn() {
        System.out.println("Turn on TV..");
    }
    public void turnOff() {
        System.out.println("Turn off TV..");
    }
    public void changeChannel(int channel) {
        System.out.println("Change channel to " + channel);
    }
}

//遥控器是发送请求者，即Invoker
public class RemoteControl {
    Command onCommand;
    Command offCommand;
    Command changeCommand;
    public RemoteControl(Command onCommand, Command offCommand, Co
mmand changeCommand) {
        this.onCommand = onCommand;
        this.offCommand = offCommand;
        this.changeCommand = changeCommand;
    }
    public void turnOn() {
        onCommand.execute();
    }
    public void turnOff() {
        offCommand.execute();
    }
    public void changeChannel() {
        changeCommand.execute();
    }
}
//测试类
public class Client {
    public static void main(String[] args) {
        Tv tv = new Tv();
        Command onCommand = new onCommand(tv);
        Command offCommand = new offCommand(tv);
        Command changeCommand = new changeCommand(tv, 2);
        RemoteControl control = new RemoteControl(onCommand, offCo
mmand, changeCommand);
        control.turnOn();
        control.turnOff();
        control.changeChannel();
    }
}
```

总结：

- 命令模式的关键在于将命令封装起来，使发送命令者和执行命令者分开来；
- 每一个命令都是一个实现命令接口的具体实现类，该类连接命令发送者和接受者。每一个命令类里面都应该有一个接受者类的实例变量；
- 命令模式的关键就在于为命令引入了抽象，使得命令发送者是针对接口编程，而不是实现编程，只有实现了抽象命令接口的命令才能与接受者相连；