链滴

设计模式 |07 命令模式

作者：qq692310342
原文链接：https://ld246.com/article/1567846637412
来源网站：链滴

开头说几句

博主的博客地址： https://www.jeffcc.top/

博主学习设计模式用的书是Head First的《设计模式》，强烈推荐配套使用！

什么是命令模式

权威定义：将“请求”封装成对象，以便使用不同的请求、队列或者日志来参数化其他对象。命令模也支持撤销操作。

博主的理解：关注的重点应该在于我们如何将一个个请求（命令）封装成对象，然后适配到命令调用手上，调用者不需要知道具体实现了什么操作，同时实现了将发出请求的对象和接受与执行请求的对之间的解耦。

设计原则

1. 封装变化
2. 多用组合，少用继承
3. 针对接口编程，不针对实现编程
4. 为交互对象之间的松耦合设计而努力
5. 类应该对拓展开放，对修改关闭
6. 依赖抽象，不依赖具体类

设计要点

1. 命令模式将发出请求的对象和执行请求的对象解耦

2. 在被解耦的两者之间是通过命令对象进行沟通的；命令对象封装了一个或者一组的动作
3. 调用者通过命令对象的execute()发出请求，这会使得接受者的动作被调用
4. 调用者可以接受命令当做参数，甚至可以在运行时动态进行
5. 命令可以支持撤销功能，方法是通过实现一个undo方法来回到execute方法执行之前的状态
6. 命令模式同时也支持使用宏命令，即无需修改源码来实现完成多个的功能的同时开关

实例设计

设计背景

实现一个多槽家用自动化遥控器，可以通过一个遥控器来实现控制起居室灯、花园洒水器，起居室吊、车库门、音响、所有的灯、舞会模式的开关，以及一些其他的拓展功能（能在不修改源码的前提下现），同时实现一个撤销功能（撤销上一个实现的命令）以及宏命令功能。

设计想法

通过设置一个命令的统一规范接口来规范化各种命令

然后设置各自命令的具体实现类

适配到遥控器中（通过数组来实现多遥控适配）

项目类图

可以看出所有命令类都实现了接口Command

项目结构

命令类规范化接口

```
package command;

/**
 * 命令的统一接口规范
 * 体现了面向接口编程思想
 */
public interface Command {
    //    调用命令的方法
    void execute();
    //     撤销命令的方法
```

```
    void undo();
}
```

设备实际操作类

```
package things;

/**
 * 所有灯光的实际控制类
 * 通过输出来模拟开关灯
 */
public class AllLight {

    public void on(){
        System.out.println("All light is on!");
    }
    public void off(){
        System.out.println("All light is off!");
    }
}
```

```
package things;

/**
 * 舞会模式的实际控制类
 * 通过输出来模拟开关舞会模式
 */
public class BallPattern {

    public void on(){
        System.out.println("Ball Pattern is on!");
    }
    public void off(){
        System.out.println("Ball Pattern is off!");
    }
}
```

```
package things;

/**
 * 车库门的实际控制类
 * 通过输出来模拟开关车库门
 */
public class GarageDoor {

    public void on(){
        System.out.println("Garage Door is on!");
    }
    public void off(){
        System.out.println("Garage Door is off!");
    }
```

```
}


package things;

/**
 * 花园洒水器的实际控制类
 * 通过输出来模拟开关洒水器
 */
public class GardenSprinkler {

    public void on(){
        System.out.println("Garden Sprinkler is on!");
    }
    public void off(){
        System.out.println("Garden Sprinkler is off!");
    }
}


package things;

/**
 * 起居室风扇的实际控制类
 * 通过输出来模拟开关以及其他功能那个风扇
 * 这里要实现撤销功能 就必须记录风扇的转速
 */
public class LivingRoomFan {

    public static final int HIGH = 3;
    public static final int MEDIUM = 2;
    public static final int LOW = 1;
    public static final int OFF = 0;
    private int speed;


    public void low() {
        System.out.println("Living Room Fan is on low speed!");
        speed = LOW;
    }

    public void medium() {
        System.out.println("Living Room Fan is on medium speed!");
        speed = MEDIUM;
    }

    public void high() {
        System.out.println("Living Room Fan is on high speed!");
        speed = HIGH;
    }
    public void off() {
        System.out.println("Living Room Fan is off!");
        speed = OFF;
    }
```

```
    public int getSpeed() {
        return speed;
    }
}
```

```
package things;

/**
 * 起居室灯光的实际控制类
 * 通过输出来模拟开关灯
 */
public class LivingRoomLight {
    public void on() {
        System.out.println("Living Room light is on!");
    }

    public void off() {
        System.out.println("Living Room light is off!");
    }
}
```

```
package things;

/**
 * 音响的实际控制类
 * 通过输出来模拟音响的开关
 */
public class Stereo {

    public void on(){
        System.out.println("Stereo is on!");
    }
    public void off(){
        System.out.println("Stereo is off!");
    }
}
```

设备命令实现类（体现命令模式的特点）

```
package command.impl.allLight;

import command.Command;
import things.AllLight;

/**
 * 关闭所有灯的命令
 */
public class AllLightOff implements Command {

    //    初始化
```

```
    private AllLight allLight;

    //    需要传递参数过来才能知道要执行
    public AllLightOff(AllLight allLight) {
        this.allLight = allLight;
    }

    //    执行命令
    @Override
    public void execute() {
        allLight.off();
    }

    //    撤销命令
    @Override
    public void undo() {
        allLight.on();
    }
}


package command.impl.allLight;

import command.Command;
import things.AllLight;

/**
 * 开所有灯的命令
 */
public class AllLightOn implements Command {

    //    初始化
    private AllLight allLight;

    //    需要传递参数过来才能知道要执行
    public AllLightOn(AllLight allLight) {
        this.allLight = allLight;
    }

    //    执行命令
    @Override
    public void execute() {
        allLight.on();
    }

    //    撤销命令
    @Override
    public void undo() {
        allLight.off();
    }
}


package command.impl.ballPartern;
```

```java
import command.Command;
import things.BallPattern;

/**
 * 关闭舞会模式的命令
 */
public class BallParternOff implements Command {

    //    初始化
    private BallPattern ballPattern;

    //    需要传递参数过来才能知道要执行
    public BallParternOff(BallPattern ballPattern) {
        this.ballPattern = ballPattern;
    }

    //    执行命令
    @Override
    public void execute() {
        ballPattern.off();
    }

    //    撤销命令
    @Override
    public void undo() {
        ballPattern.on();
    }
}
```

```java
package command.impl.ballPartern;

import command.Command;
import things.BallPattern;

/**
 * 开启舞会模式的命令
 */
public class BallParternOn implements Command {

    //    初始化
    private BallPattern ballPattern;

    //    需要传递参数过来才能知道要执行
    public BallParternOn(BallPattern ballPattern) {
        this.ballPattern = ballPattern;
    }

    //    执行命令
    @Override
    public void execute() {
        ballPattern.on();
    }
```

```
    //    撤销命令
    @Override
    public void undo() {
        ballPattern.off();
    }
}


package command.impl.garageDoor;

import command.Command;
import things.GarageDoor;

/**
 * 关车库门的命令
 */
public class GarageDoorOff implements Command {

    private GarageDoor garageDoor;

    public GarageDoorOff(GarageDoor garageDoor) {
        this.garageDoor = garageDoor;
    }

    @Override
    public void execute() {
        garageDoor.off();
    }

    @Override
    public void undo() {
        garageDoor.on();
    }
}


package command.impl.garageDoor;

import command.Command;
import things.GarageDoor;

/**
 * 开车库门的命令
 */
public class GarageDoorOn implements Command {

    private GarageDoor garageDoor;

    public GarageDoorOn(GarageDoor garageDoor) {
        this.garageDoor = garageDoor;
    }

    @Override
```

```
    public void execute() {
        garageDoor.on();
    }

    @Override
    public void undo() {
        garageDoor.off();
    }
}


package command.impl.gardenSprinkler;

import command.Command;
import things.GardenSprinkler;

/**
 * 控制花园洒水器关的命令
 */
public class GardenSprinklerOff implements Command {
    private GardenSprinkler gardenSprinkler ;

    public GardenSprinklerOff(GardenSprinkler gardenSprinkler) {
        this.gardenSprinkler = gardenSprinkler;
    }

    @Override
    public void execute() {
        gardenSprinkler.off();
    }

    @Override
    public void undo() {
        gardenSprinkler.on();
    }
}


package command.impl.gardenSprinkler;

import command.Command;
import things.GardenSprinkler;

/**
 * 控制花园洒水器开的命令
 */
public class GardenSprinklerOn implements Command {
    private GardenSprinkler gardenSprinkler ;

    public GardenSprinklerOn(GardenSprinkler gardenSprinkler) {
        this.gardenSprinkler = gardenSprinkler;
    }

    @Override
```

```
    public void execute() {
        gardenSprinkler.on();
    }

    @Override
    public void undo() {
        gardenSprinkler.off();
    }
}


package command.impl.livingRoomFan;

import command.Command;
import things.LivingRoomFan;

/**
 * 控制起居室电风扇HIGH档的命令
 */
public class LivingRoomFanHigh implements Command {
    private LivingRoomFan livingRoomFan;
//    初始化速度
    private int speed;

    public LivingRoomFanHigh(LivingRoomFan livingRoomFan) {
        this.livingRoomFan = livingRoomFan;
    }

    @Override
    public void execute() {
//        顺序不能倒
        speed = livingRoomFan.getSpeed();
        livingRoomFan.high();
    }

    @Override
    public void undo() {
//        需要做判断
        if(speed==LivingRoomFan.HIGH){
            livingRoomFan.high();
        }else if(speed==LivingRoomFan.MEDIUM){
            livingRoomFan.medium();
        }else if(speed==LivingRoomFan.LOW){
            livingRoomFan.low();
        }else if(speed==LivingRoomFan.OFF){
            livingRoomFan.off();
        }
    }
}


package command.impl.livingRoomFan;

import command.Command;
```

```java
import things.LivingRoomFan;

/**
 * 控制起居室电风扇LOW档的命令
 */
public class LivingRoomFanLow implements Command {
    private LivingRoomFan livingRoomFan;
//    初始化速度
    private int speed;

    public LivingRoomFanLow(LivingRoomFan livingRoomFan) {
        this.livingRoomFan = livingRoomFan;
    }

    @Override
    public void execute() {
        speed = livingRoomFan.getSpeed();
        livingRoomFan.low();
    }

    @Override
    public void undo() {
//        需要做判断
        if(speed==LivingRoomFan.HIGH){
            livingRoomFan.high();
        }else if(speed==LivingRoomFan.MEDIUM){
            livingRoomFan.medium();
        }else if(speed==LivingRoomFan.LOW){
            livingRoomFan.low();
        }else if(speed==LivingRoomFan.OFF){
            livingRoomFan.off();
        }
    }
}


package command.impl.livingRoomFan;

import command.Command;
import things.LivingRoomFan;

/**
 * 控制起居室电风扇MEDIUM档的命令
 */
public class LivingRoomFanMedium implements Command {
    private LivingRoomFan livingRoomFan;
//    初始化速度
    private int speed;

    public LivingRoomFanMedium(LivingRoomFan livingRoomFan) {
        this.livingRoomFan = livingRoomFan;
    }

    @Override
```

```
    public void execute() {
        speed = livingRoomFan.getSpeed();
        livingRoomFan.medium();
    }

    @Override
    public void undo() {
//          需要做判断
        if(speed==LivingRoomFan.HIGH){
            livingRoomFan.high();
        }else if(speed==LivingRoomFan.MEDIUM){
            livingRoomFan.medium();
        }else if(speed==LivingRoomFan.LOW){
            livingRoomFan.low();
        }else if(speed==LivingRoomFan.OFF){
            livingRoomFan.off();
        }
    }
}


package command.impl.livingRoomFan;

import command.Command;
import things.LivingRoomFan;

/**
 * 控制起居室电风扇Off档的命令
 */
public class LivingRoomFanOff implements Command {
    private LivingRoomFan livingRoomFan;
//    初始化速度
    private int speed;

    public LivingRoomFanOff(LivingRoomFan livingRoomFan) {
        this.livingRoomFan = livingRoomFan;
    }

    @Override
    public void execute() {
        speed = livingRoomFan.getSpeed();
        livingRoomFan.off();
    }

    @Override
    public void undo() {
//          需要做判断
        if(speed==LivingRoomFan.HIGH){
            livingRoomFan.high();
        }else if(speed==LivingRoomFan.MEDIUM){
            livingRoomFan.medium();
        }else if(speed==LivingRoomFan.LOW){
            livingRoomFan.low();
        }else if(speed==LivingRoomFan.OFF){
```

```
            livingRoomFan.off();
        }
    }
}


package command.impl.livingRoomLight;

import command.Command;
import things.LivingRoomLight;

/**
 * 实现起居室灯关的命令
 */
public class LivingRoomLightOff implements Command {
    private LivingRoomLight livingRoomLight;

    public LivingRoomLightOff(LivingRoomLight livingRoomLight) {
        this.livingRoomLight = livingRoomLight;
    }

    @Override
    public void execute() {
        livingRoomLight.off();
    }

    @Override
    public void undo() {
        livingRoomLight.on();
    }
}


package command.impl.livingRoomLight;

import command.Command;
import things.LivingRoomLight;

/**
 * 实现起居室灯开的命令
 */
public class LivingRoomLightOn implements Command {
    private LivingRoomLight livingRoomLight;

    public LivingRoomLightOn(LivingRoomLight livingRoomLight) {
        this.livingRoomLight = livingRoomLight;
    }

    @Override
    public void execute() {
        livingRoomLight.on();
    }

    @Override
```

```
    public void undo() {
        livingRoomLight.off();
    }
}


package command.impl.stereo;

import command.Command;
import things.Stereo;

/**
 * 关闭音响的命令
 */
public class StereoOff implements Command {
    //    初始化
    private Stereo stereo;

    public StereoOff(Stereo stereo) {
        this.stereo = stereo;
    }

    //    执行命令
    @Override
    public void execute() {
        stereo.off();
    }

    //    撤销命令
    @Override
    public void undo() {
        stereo.on();
    }
}


package command.impl.stereo;

import command.Command;
import things.Stereo;

/**
 * 开音响的命令
 */
public class StereoOn implements Command {
    //    初始化
    private Stereo stereo;

    public StereoOn(Stereo stereo) {
        this.stereo = stereo;
    }

    //    执行命令
    @Override
```

```
    public void execute() {
        stereo.on();
    }

    //    撤销命令
    @Override
    public void undo() {
        stereo.off();
    }
}
```

宏命令类以及空命令类

```
package command.impl;

import command.Command;

/**
 * 宏命令
 */
public class MacroCommand implements Command {
    private Command[] commands;
    private Command[] undoCommands;

    public MacroCommand(Command[] commands) {
        this.commands = commands;
    }

    @Override
    public void execute() {
//        初始化撤销
        undoCommands = new Command[commands.length];
        for (int i = 0; i < commands.length; i++) {
            commands[i].execute();
            undoCommands[i] = commands[i];
        }
    }

//    宏撤销
    @Override
    public void undo() {
        for (Command undoCommand : undoCommands) {
            undoCommand.undo();
        }
    }
}


package command.impl;

import command.Command;

/**
```

```
 * 空命令对象 什么都不做
 */
public class NullCommand implements Command {

    @Override
    public void execute() {

    }

    @Override
    public void undo() {

    }
}
```

遥控器类（关键）

```
package controller;

import command.Command;
import command.impl.NullCommand;

/**
 * 遥控器类
 */
public class RemoteControl {
    //    数组存放命令分别为 开和关的命令数组
    private Command[] onCommands;
    private Command[] offCommands;
    //    撤销命令保存
    private Command undoCommand;

    public RemoteControl() {
//        为什么是10 因为风扇有三个档位 对应三组的命令  6+3+1宏命令=10
        this.onCommands = new Command[10];
        this.offCommands = new Command[10];
//    初始化命令  这里需要创建一个空命令对象来让没有命令的按键什么都不做
        Command noCommand = new NullCommand();
        for (int i = 0; i < onCommands.length; i++) {
            onCommands[i] = noCommand;
            offCommands[i] = noCommand;
        }
//        初始化 撤销命令
        undoCommand = noCommand;
    }

    /**
     * 设置遥控器上的按键命令
     *
     * @param solt       命令所在位置
     * @param onCommand  开命令
     * @param offCommand 关命令
     */
```

```
    public void setCommand(int solt, Command onCommand, Command offCommand) {
        onCommands[solt] = onCommand;
        offCommands[solt] = offCommand;
    }

    /**
     * 控制第几个按钮的开
     *
     * @param slot
     */
    public void onButtonWasPushed(int slot) {
        onCommands[slot].execute();
//        记录要撤销的命令
        undoCommand = onCommands[slot];
    }

    /**
     * 控制第几个按钮的关
     *
     * @param slot
     */
    public void offButtonWasPushed(int slot) {
        offCommands[slot].execute();
        //        记录要撤销的命令
        undoCommand = offCommands[slot];
    }

    /**
     * 调用撤销命令
     */
    public void undoButtonWasPushed(){
        undoCommand.undo();
    }


    /**
     * 重写toString
     * 在遥控器重置完成之后输出按钮情况
     *
     * @return
     */
    @Override
    public String toString() {
        StringBuffer stringBuffer = new StringBuffer();
        stringBuffer.append("\n----------------  Remote Control ---------------\n");
        for (int i = 0; i < onCommands.length; i++) {
            stringBuffer.append("[solt" + i + "] " + onCommands[i].getClass().getName() + "   " + o
fCommands[i].getClass().getName() + "\n");
        }
        return stringBuffer.toString();
    }
}
```

测试类

```
package test;

import command.Command;
import command.impl.MacroCommand;
import command.impl.allLight.AllLightOff;
import command.impl.allLight.AllLightOn;
import command.impl.ballPartern.BallParternOff;
import command.impl.ballPartern.BallParternOn;
import command.impl.garageDoor.GarageDoorOff;
import command.impl.garageDoor.GarageDoorOn;
import command.impl.gardenSprinkler.GardenSprinklerOff;
import command.impl.gardenSprinkler.GardenSprinklerOn;
import command.impl.livingRoomFan.LivingRoomFanHigh;
import command.impl.livingRoomFan.LivingRoomFanLow;
import command.impl.livingRoomFan.LivingRoomFanMedium;
import command.impl.livingRoomFan.LivingRoomFanOff;
import command.impl.livingRoomLight.LivingRoomLightOff;
import command.impl.livingRoomLight.LivingRoomLightOn;
import command.impl.stereo.StereoOff;
import command.impl.stereo.StereoOn;
import controller.RemoteControl;
import things.*;

/**
 * 测试遥控器
 */
public class RemoteControTest {
    public static void main(String[] args) {
//        初始化家居
        AllLight allLight = new AllLight();
        BallPattern ballPattern = new BallPattern();
        GarageDoor garageDoor = new GarageDoor();
        GardenSprinkler gardenSprinkler = new GardenSprinkler();
        LivingRoomFan livingRoomFan = new LivingRoomFan();
        LivingRoomLight livingRoomLight = new LivingRoomLight();
        Stereo stereo = new Stereo();

//        初始化各种命令
        Command allLightOn = new AllLightOn(allLight);
        Command allLightOff = new AllLightOff(allLight);

        Command ballParternOn = new BallParternOn(ballPattern);
        Command ballParternOff = new BallParternOff(ballPattern);

        Command garageDoorOn = new GarageDoorOn(garageDoor);
        Command garageDoorOff = new GarageDoorOff(garageDoor);

        Command gardenSprinklerOn = new GardenSprinklerOn(gardenSprinkler);
        Command gardenSprinklerOff = new GardenSprinklerOff(gardenSprinkler);

        Command livingRoomFanLow = new LivingRoomFanLow(livingRoomFan);
        Command livingRoomFanMedium = new LivingRoomFanMedium(livingRoomFan);
```

```
        Command livingRoomFanHigh = new LivingRoomFanHigh(livingRoomFan);
        Command livingRoomFanOff = new LivingRoomFanOff(livingRoomFan);

        Command livingRoomLightOn = new LivingRoomLightOn(livingRoomLight);
        Command livingRoomLightOff = new LivingRoomLightOff(livingRoomLight);

        Command stereoOn = new StereoOn(stereo);
        Command stereoOff = new StereoOff(stereo);

//      初始化遥控器
        RemoteControl remoteControl = new RemoteControl();
        remoteControl.setCommand(0,allLightOn,allLightOff);
        remoteControl.setCommand(1,ballParternOn,ballParternOff);
        remoteControl.setCommand(2,garageDoorOn,garageDoorOff);
        remoteControl.setCommand(3,gardenSprinklerOn,garageDoorOff);
        remoteControl.setCommand(4,livingRoomFanLow,livingRoomFanOff);
        remoteControl.setCommand(5,livingRoomFanMedium,livingRoomFanOff);
        remoteControl.setCommand(6,livingRoomFanHigh,livingRoomFanOff);
        remoteControl.setCommand(7,livingRoomLightOn,livingRoomLightOff);
        remoteControl.setCommand(8,stereoOn,stereoOff);

//      输出初始化信息
        System.out.println(remoteControl);

//      初始化完成
//      进行遥控控制

        System.out.println("----------- All Light ---------------");

//      开灯
        remoteControl.onButtonWasPushed(0);
//      关灯
        remoteControl.offButtonWasPushed(0);
//      撤销关灯 也就是重新开灯
        remoteControl.undoButtonWasPushed();
        System.out.println("----------- Garage Door ---------------");

        remoteControl.onButtonWasPushed(2);
        remoteControl.undoButtonWasPushed();
        remoteControl.offButtonWasPushed(2);
        System.out.println("----------- Fan ---------------");

//      先调到Low档位
        remoteControl.onButtonWasPushed(4);
//      调到Medium档位
        remoteControl.onButtonWasPushed(5);
//      撤销 就是回到low档位
        remoteControl.undoButtonWasPushed();

//      宏命令

//      添加宏命令进入遥控器   一键开启 所有灯光、花园洒水器、音响、起居室灯光
        Command macroCommandOn = new MacroCommand(new Command[]{allLightOn, gar
enSprinklerOn, stereoOn, livingRoomLightOn});
```

```
        Command macroCommandOff = new MacroCommand(new Command[]{allLightOff, gar
enSprinklerOff, stereoOff, livingRoomLightOff});
        remoteControl.setCommand(9,macroCommandOn,macroCommandOff);

//        重新显示遥控器参数
        System.out.println(remoteControl);

//        使用宏命令
        System.out.println("-------------- macroCommand --------------------");
        remoteControl.onButtonWasPushed(9);
//        撤销宏
        remoteControl.undoButtonWasPushed();

    }
}
```

测试结果

```
---------------- Remote Control ---------------
[solt0] command.impl.allLight.AllLightOn   command.impl.allLight.AllLightOff
[solt1] command.impl.ballPartern.BallParternOn   command.impl.ballPartern.BallParternOff
[solt2] command.impl.garageDoor.GarageDoorOn   command.impl.garageDoor.GarageDoorO
f
[solt3] command.impl.gardenSprinkler.GardenSprinklerOn   command.impl.garageDoor.Gara
eDoorOff
[solt4] command.impl.livingRoomFan.LivingRoomFanLow   command.impl.livingRoomFan.Livi
gRoomFanOff
[solt5] command.impl.livingRoomFan.LivingRoomFanMedium   command.impl.livingRoomFan
LivingRoomFanOff
[solt6] command.impl.livingRoomFan.LivingRoomFanHigh   command.impl.livingRoomFan.Liv
ngRoomFanOff
[solt7] command.impl.livingRoomLight.LivingRoomLightOn   command.impl.livingRoomLight.
ivingRoomLightOff
[solt8] command.impl.stereo.StereoOn   command.impl.stereo.StereoOff
[solt9] command.impl.NullCommand   command.impl.NullCommand

----------- All Light ---------------
All light is on!
All light is off!
All light is on!
----------- Garage Door ---------------
Garage Door is on!
Garage Door is off!
Garage Door is off!
----------- Fan ---------------
Living Room Fan is on low speed!
Living Room Fan is on medium speed!
Living Room Fan is on low speed!

---------------- Remote Control ---------------
[solt0] command.impl.allLight.AllLightOn   command.impl.allLight.AllLightOff
[solt1] command.impl.ballPartern.BallParternOn   command.impl.ballPartern.BallParternOff
```

```
[solt2] command.impl.garageDoor.GarageDoorOn   command.impl.garageDoor.GarageDoorOf
[solt3] command.impl.gardenSprinkler.GardenSprinklerOn   command.impl.garageDoor.GaraeDoorOff
[solt4] command.impl.livingRoomFan.LivingRoomFanLow   command.impl.livingRoomFan.LivigRoomFanOff
[solt5] command.impl.livingRoomFan.LivingRoomFanMedium   command.impl.livingRoomFanLivingRoomFanOff
[solt6] command.impl.livingRoomFan.LivingRoomFanHigh   command.impl.livingRoomFan.LivngRoomFanOff
[solt7] command.impl.livingRoomLight.LivingRoomLightOn   command.impl.livingRoomLight.ivingRoomLightOff
[solt8] command.impl.stereo.StereoOn   command.impl.stereo.StereoOff
[solt9] command.impl.MacroCommand   command.impl.MacroCommand

-------------- macroCommand --------------------
All light is on!
Garden Sprinkler is on!
Stereo is on!
Living Room light is on!
All light is off!
Garden Sprinkler is off!
Stereo is off!
Living Room light is off!

Process finished with exit code 0
```

分析

实现了通过把命令封装成类来实现的遥控器设置。

命令模式的现实用途

队列请求

想象有一个工作队列，我们在一端不断的添加命令，然后工作队列的线程只需要不断的从队列中取出令并且调用命令的execute()方法，等待这个调用完成，然后将命令对象丢弃掉，再继续循环下去。

那么在这个过程中，我们的工作队列和要进行运行的对象是完全解耦的，工作队列不需要知道运行的象是什么方法，只需要运行execute()方法就可以了，这一角度也反映了命令模式的特点，将工作执行和调用者的解耦。

日志请求

日志系统需要我们实时记录下系统运行的动作，我们可以在Command接口中在规范新增两个方法store()以及load()方法。我们可以通过在每个命令进行的时候调用store方法保存动作到日志中，当遇到统死机的情况，就将命令对象重新加载，调用load方法来获取进行的动作，并且成批的进行execute法的调用以恢复到系统此前的状态。

事务系统

我们的命令模式也可以巧妙地利用在事务系统中，可以通过调用一整群的命令操作，并且要求要么全一起完成，要么没有进行任何操作。

END

2019年9月7日16:57:03