2. 命令模式

智能生活项目需求

看一个具体的需求

1） 我们买了一套智能家电，有照明灯、风扇、冰箱、洗衣机，我们只要在手机上安装 app 就可以控制对这些家电工作。

2） 这些智能家电来自不同的厂家，我们不想针对每一种家电都安装一个 App，分别控制，我们希望只要一个 app
就可以控制全部智能家电。

3） 要实现一个 app 控制所有智能家电的需要，则每个智能家电厂家都要提供一个统一的接口给 app 调用，这时 就可以考虑使用命令模式。

4） 命令模式可将“动作的请求者”从“动作的执行者”对象中解耦出来.

5） 在我们的例子中，动作的请求者是手机 app，动作的执行者是每个厂商的一个家电产品

命令模式基本介绍

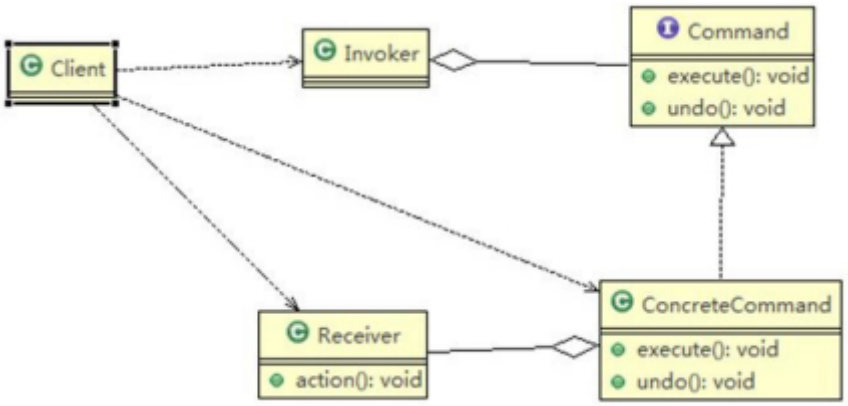
1） 命令模式（Command Pattern）：在软件设计中，我们经常需要向某些对象发送请求，但是并不知道请求的接收者是谁，也不知道被请求的操作是哪个，
我们只需在程序运行时指定具体的请求接收者即可，此时，可以使用命令模式来进行设计

2） 命令模式使得请求发送者与请求接收者消除彼此之间的耦合，让对象之间的调用关系更加灵活，实现解耦。

3) 在命名模式中，会将一个请求封装为一个对象，以便使用不同参数来表示不同的请求(即命名)，同时命令模式也支持可撤销的操作。

4) 通俗易懂的理解：将军发布命令，士兵去执行。其中有几个角色：将军（命令发布者）、士兵（命令的具体执行者）、命令(连接将军和士兵)。Invoker 是调用者（将军），Receiver 是被调用者（士兵），MyCommand 是命令，实现了 Command 接口，持有接收对象

命令模式的原理类图



对原理类图的说明-即(命名模式的角色及职责)

1) Invoker 是调用者角色

2) Command: 是命令角色，需要执行的所有命令都在这里，可以是接口或抽象类

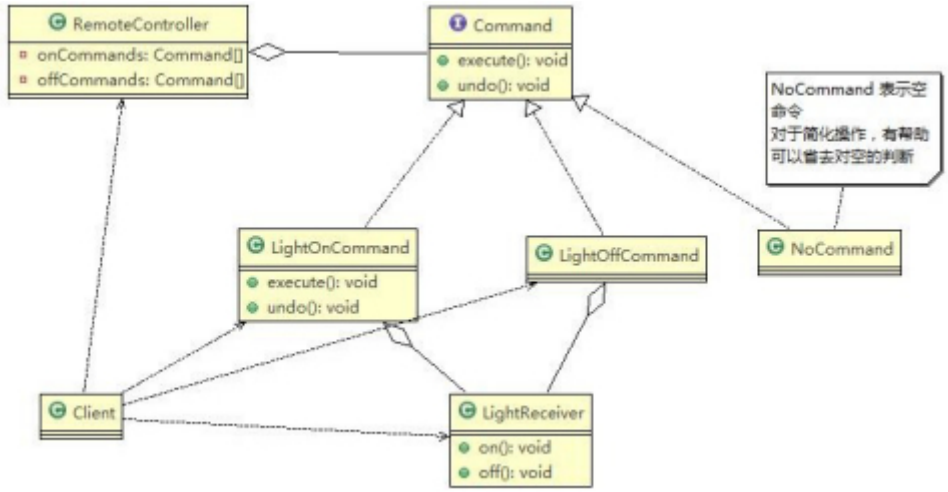
3) Receiver: 接受者角色，知道如何实施和执行一个请求相关的操作

4) ConcreteCommand: 将一个接受者对象与一个动作绑定，调用接受者相应的操作，实现 execute

命令模式解决智能生活项目

1) 编写程序，使用命令模式 完成前面的智能家电项目

2) 思路分析和图解




```
//创建命令接口
public interface Command {
   //执行动作(操作)
   public void execute();
   //撤销动作(操作)
   public void undo();
}
public class LightOnCommand implements Command {
   //聚合LightReceiver

   LightReceiver light;

   //构造器
   public LightOnCommand(LightReceiver light) {
      this.light = light;
   }
   @Override
   public void execute() {
      //调用接收者的方法
      light.on();
   }

   @Override
   public void undo() {
      //调用接收者的方法
      light.off();
   }
}

public class LightOffCommand implements Command {省略}
public class NoCommand implements Command {继承的方法不需要写任何内容}
public class LightReceiver {
   public void on() {
      System.out.println(" 电灯打开了.. ");
   }
```

```
    public void off() {
        System.out.println(" 电灯关闭了.. ");
    }
}
public class RemoteController {
    // 开 按钮的命令数组
    Command[] onCommands;
    Command[] offCommands;
    // 执行撤销的命令
    Command undoCommand;

    // 构造器，完成对按钮初始化
    public RemoteController() {
        onCommands = new Command[5];
        offCommands = new Command[5];
        for (int i = 0; i < 5; i++) {
            onCommands[i] = new NoCommand();
            offCommands[i] = new NoCommand();
        }
    }

    // 给我们的按钮设置你需要的命令
    public void setCommand(int no, Command onCommand, Command
offCommand) {
        onCommands[no] = onCommand;
        offCommands[no] = offCommand;
    }

    // 按下开按钮
    public void onButtonWasPushed(int no) { // no 0
        // 找到你按下的开的按钮， 并调用对应方法
        onCommands[no].execute();
        // 记录这次的操作，用于撤销
        undoCommand = onCommands[no];
    }

    // 按下开按钮
    public void offButtonWasPushed(int no) { // no 0
        // 找到你按下的关的按钮， 并调用对应方法
        offCommands[no].execute();
        // 记录这次的操作，用于撤销
        undoCommand = offCommands[no];
    }

    // 按下撤销按钮
    public void undoButtonWasPushed() {
        undoCommand.undo();
```

```
    }
}
```

命令模式在 Spring 框架 JdbcTemplate 应用的源码分析

1) Spring 框架的 JdbcTemplate 就使用到了命令模式

2) 代码分析

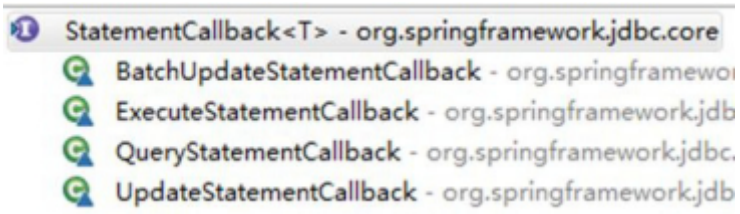
1) 模式角色分析说明

StatementCallback 接口 ,类似命令接口(Command)

class QueryStatementCallback implements StatementCallback<T>, SqlProvider ，匿名内部类， 实现了命令接口， 同时也充当命令接收者

命令调用者 是 JdbcTemplate ， 其中 execute(StatementCallback<T> action) 方法中，调用 action.doInStatement 方法. 不同的 实现 StatementCallback 接口的对象，对应不同的 doInStatemnt 实现逻辑

1另外实现 StatementCallback 命令接口的子类还有 QueryStatementCallback、



命令模式的注意事项和细节

1） 将发起请求的对象与执行请求的对象解耦。发起请求的对象是调用者，调用者只要调用命令对象的 execute()方法就可以让接收者工作，而不必知道具体的接收者对象是谁、是如何实现的，命令对象会负责让接收者执行请求的动作，也就是说：”请求发起者”和“请求执行者”之间的解耦是通过命令对象实现的，命令对象起到了纽带桥梁的作用。

2） 容易设计一个命令队列。只要把命令对象放到列队，就可以多线程的执行命令

3） 容易实现对请求的撤销和重做

4） 命令模式不足：可能导致某些系统有过多的具体命令类，增加了系统的复杂度，这点在在使用的时候要注意

5） 空命令也是一种设计模式，它为我们省去了判空的操作。在上面的实例中，如果没有用空命令，我们每按下一个按键都要判空，这给我们编码带来一定的麻烦。

6） 命令模式经典的应用场景：界面的一个按钮都是一条命令、模拟 CMD（DOS 命令）订单的撤销/恢复、触发- 反馈机制