Java 设计模式之命令模式（十四）

一、前言

本篇主题为行为型模式中的第二个模式–命令模式。上篇 Java 设计模式主题为[《Java 设计模式之模板方法模式（十三）》]。

二、简单介绍

2.1 定义

命令模式将一个请求封装为一个对象，从而使你可用不同的请求对客户进行参数化；对请求排队或记录请求日志，以及支持可撤消的操作。

2.2 参与角色

1. Command：声明执行操作的接口。
2. ConcreteCommand：将一个接收者对象绑定于一个动作。调用接收者相应的操作，以实现Execute。
3. Invoker：执行请求命令的执行者。
4. Receiver：请求命令接收者。

2.3 应该场景

1. 抽象出待执行的动作以参数化某对象。
2. 在不同的时刻指定、排列和执行请求。
3. 支持取消操作。
4. 支持修改日志，这样当系统崩溃时，这些修改可以被重做一遍。

三、实现方式

我们以去饭店吃饭为例。我们先点单，然后让厨师做菜。

厨师类：

```
public class Chef {

    public void cookDuck() {
        System.out.println("做北京烤鸭");
    }

    public void cookChicken() {
        System.out.println("做宫保鸡丁");
    }
}
```

客户端：

```
public class Client {

    public static void main(String[] args) {

        Chef chef=  new Chef();

        chef.cookChicken();
        chef.cookChicken();

        chef.cookDuck();
    }
}
```

打印结果：

```
做宫保鸡丁
做宫保鸡丁
做北京烤鸭
```

功能基本实现了，但是几个问题：

1. 假设来饭店吃饭的人多了，厨师做的菜也相应多了，那么就意味着客户端需要重复调用 cookXXX 方法，不友好。
2. 假设下单后，客人不想吃某样菜，需要取消指定的菜单，但代码中没有取消功能。
3. 点餐到做菜的流程不够明确，现在客户端代码中体现的流程是厨师既直接接收客人点餐请求又要执行做菜操作。

正常流程是客人将点单请求（多个）发给饭店服务员，再由服务员到后厨将客人的请求转达给厨师，厨师才开始做菜。

我们通过命令模式解决上述问题：

厨师（Invoker）：代码不变

Command 和 实现类：

```
public interface Command {

    public void action();
}

class CookDuckCommand implements Command {

    private Chef chef;

    public CookDuckCommand(Chef chef) {
        this.chef = chef;
    }

    @Override
    public void action() {
        this.chef.cookDuck();
    }

}

class CookChickenCommand implements Command {

    private Chef chef;

    public CookChickenCommand(Chef chef) {
        this.chef = chef;
    }

    @Override
    public void action() {
        this.chef.cookChicken();
    }

}
```

服务员（Receiver）：

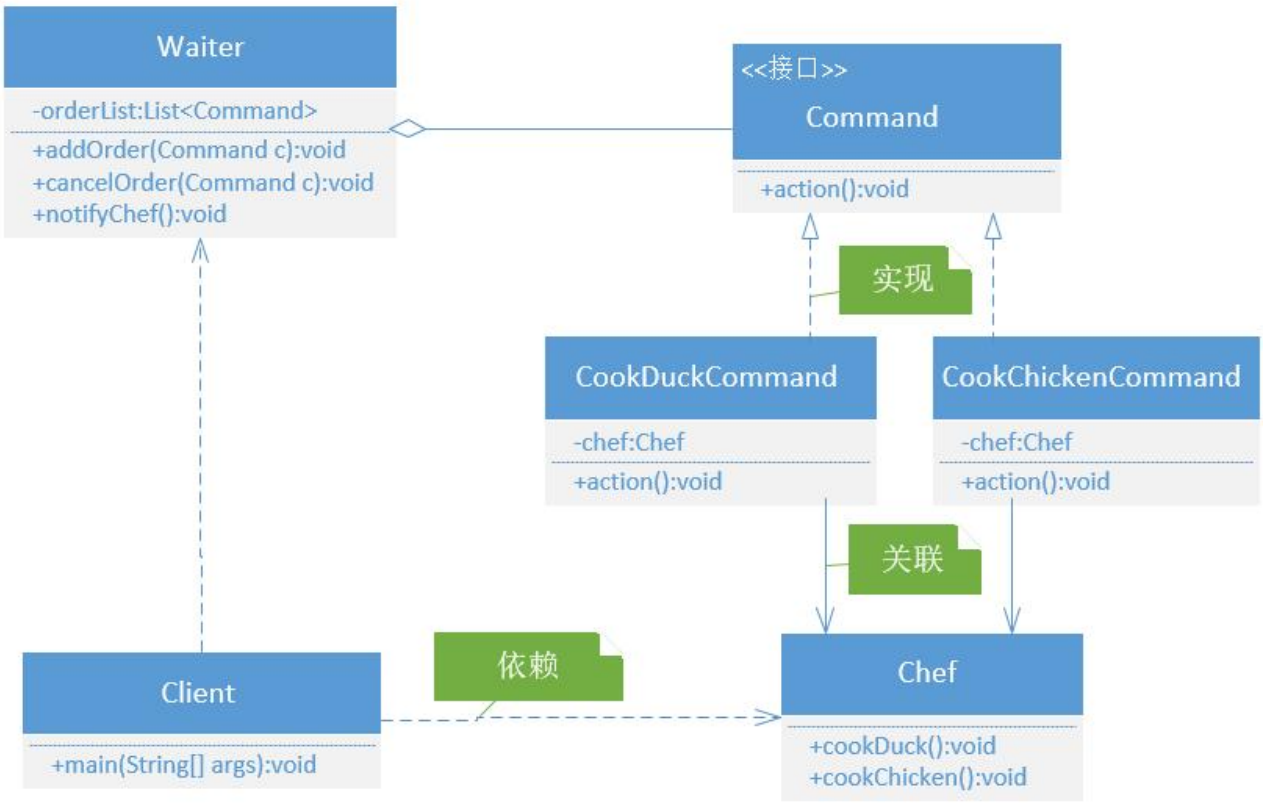
```
public class Waiter {

    private List<Command> orderList = new ArrayList<>();

    public void addOrder(Command command) {
        this.orderList.add(command);
    }

    public void cancelOrder(Command command) {
        this.orderList.remove(command);
    }

    public void notifyChef() {
        for (Command command : orderList) {
            command.action();
        }
    }
}
```

客户端：

```
public class Client {

    public static void main(String[] args) {
        Chef chef = new Chef();
        // 客人点单
        Command command1 = new CookChickenCommand(chef);
        Command command2 = new CookChickenCommand(chef);
        Command command3 = new CookDuckCommand(chef);

        // 服务员记录客人点单
        Waiter waiter = new Waiter();
        waiter.addOrder(command1);
        waiter.addOrder(command2);
        waiter.addOrder(command3);

        // 鸭有骚味，不要了
        waiter.cancelOrder(command3);

        // 一次性通知
        waiter.notifyChef();
    }
}
```

在客户端代码中，我们可以下达多个点单命令，也可以取消命令。

UML 类图表示如下：


Waiter
-orderList:List<Command>
+addOrder(Command c):void
+cancelOrder(Command c):void
+notifyChef():void
<<接口>>
Command
+action():void
实现
CookDuckCommand
-chef:Chef
+action():void
CookChickenCommand
-chef:Chef
+action():void
关联
Client
+main(String[] args):void
依赖
Chef
+cookDuck():void
+cookChicken():void