**《设计模式》上机作业**

1240610927白一泽

计算机系软件技术4班

**本次上机学习模式类型：命令模式**

1. 该设计模式的作用和目的。

在软件系统中，“行为请求者”与“行为实现者”通常呈现一种“紧耦合”。但在某些场合，比如要对行为进行“记录、撤销/重做、事务”等处理，这种无法抵御变化的紧耦合是不合适的(而且会破坏设计原则中的开放封闭原则)。在这种情况下，如何将“行为请求者”与“行为实现者”解耦？将一组行为抽象为对象，可以实现二者之间的松耦合。这就是本文要说的命令模式。

命令模式是一个高内聚的模式，其定义为，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；对请求排队或者记录请求日志，以及支持可撤销的操作。

1. 该设计模式涉及的组成部分及其各部分的描述。

命令模式组成部分和各部分描述如下：

**Command：**定义命令的接口，声明执行的方法。

**ConcreteCommand：**命令接口实现对象，是“虚”的实现；通常会持有接收者，并调用接收者的功能来完成命令要执行的操作。

**Receiver：**接收者，真正执行命令的对象。任何类都可能成为一个接收者，只要它能够实现命令要求实现的相应功能。

**Invoker：**要求命令对象执行请求，通常会持有命令对象，可以持有很多的命令对象。这个是客户端真正触发命令并要求命令执行相应操作的地方，也就是说相当于使用命令对象的入口。

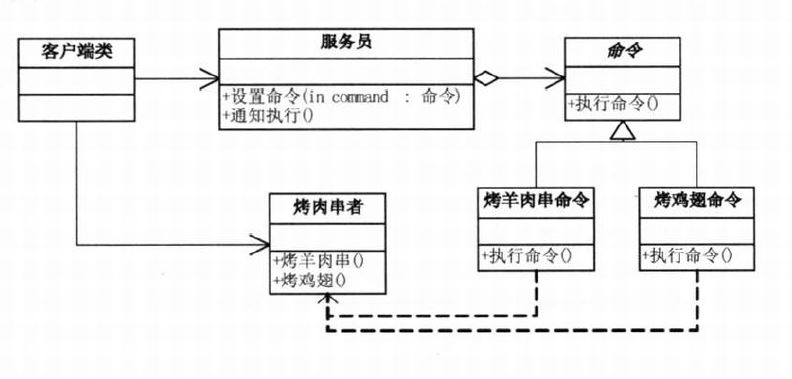
**Client：**创建具体的命令对象，并且设置命令对象的接收者。注意这个不是我们常规意义上的客户端，而是在组装命令对象和接收者，或许，把这个Client称为装配者会更好理解，因为真正使用命令的客户端是从Invoker来触发执行。

1. 该设计模式所涉及到的类的结构图



1. 该设计模式的举例说明。

比如一门面烧烤店就是用的命令模式，我们去店里吃烧烤，只用点餐，服务员记下我们点的东西，然后她通知烧烤的厨师给我们烤东西，而我们顾客都是排队的，而且可以换菜，和不要什么吃的，还可以增加，而且不会出错，因为有本子记着，所以，我们可以清楚的知道，有没算错账。这就是四轮马车“增删改查”。



核心代码如下(用C#写的)：

//客户端

static void Main(string[] args)

{

//开店前的准备

Barbecuer boy = new Barbecuer();

Command bakeMuttonCommand1 = new BakeMuttonCommand(boy);

Command bakeMuttonCommand2 = new BakeMuttonCommand(boy);

Command bakeChickenWingCommand1 = new BakeChickenWingCommand(boy);

Waiter girl = new Waiter();

//开门营业 顾客点菜

girl.SetOrder(bakeMuttonCommand1);

girl.SetOrder(bakeMuttonCommand2);

girl.SetOrder(bakeChickenWingCommand1);

//点菜完闭，通知厨房

girl.Notify();

Console.Read();

}

//服务员

public class Waiter

{

private IList<Command> orders = new List<Command>();

//设置订单

public void SetOrder(Command command)

{

if (command.ToString() == "命令模式.BakeChickenWingCommand")

{

Console.WriteLine("服务员：鸡翅没有了，请点别的烧烤。");

}

else

{

orders.Add(command);

Console.WriteLine("增加订单：" + command.ToString() + " 时间：" + DateTime.Now.ToString());

}

}

//取消订单

public void CancelOrder(Command command)

{

orders.Remove(command);

Console.WriteLine("取消订单：" + command.ToString() + " 时间：" + DateTime.Now.ToString());

}

//通知全部执行

public void Notify()

{

foreach (Command cmd in orders)

{

cmd.ExcuteCommand();

}

}

}

//抽象命令

public abstract class Command

{

protected Barbecuer receiver;

public Command(Barbecuer receiver)

{

this.receiver = receiver;

}

//执行命令

abstract public void ExcuteCommand();

}

//烤羊肉串命令

class BakeMuttonCommand : Command

{

public BakeMuttonCommand(Barbecuer receiver)

: base(receiver)

{ }

public override void ExcuteCommand()

{

receiver.BakeMutton();

}

}

//烤鸡翅命令

class BakeChickenWingCommand : Command

{

public BakeChickenWingCommand(Barbecuer receiver)

: base(receiver)

{ }

public override void ExcuteCommand()

{

receiver.BakeChickenWing();

}

}

//烤肉串者

public class Barbecuer

{

public void BakeMutton()

{

Console.WriteLine("烤羊肉串!");

}

public void BakeChickenWing()

{

Console.WriteLine("烤鸡翅!");

}

}

1. 该设计模式的效果分析

命令模式模式可应用于  
1，整个调用过程比较繁杂，或者存在多处这种调用。这时，使用Command类对该调用加以封装，便于功能的再利用。  
2，调用前后需要对调用参数进行某些处理。  
3，调用前后需要进行某些额外处理，比如日志，缓存，记录历史操作等。

命令模式有如下下效果：

1，将调用操作的对象和知道如何实现该操作的对象解耦。  
2，Command是头等对象。他们可以像其他对象一样被操作和扩展。  
3，你可将多个命令装配成一个符合命令。

4，增加新的Command很容易，因为这无需改变现有的类。

6，描述有关该设计模式在本书中并未提到的一些其它知识

命令模式能够比较容易的设计一个命令队列；在需要的情况下，可以很容易的将命令记入日志；接收请求的一方可以否决请求；其实命令模式最关键的一点就是可以把请求一个操作的对象和知道怎么执行一个操作的对象分开。但是不是碰到类似情况就一定要用命令模式呢？这个倒不一定，关键是你需不需要相关的功能，比如，命令模式支持撤销和恢复操作功能，你又不清楚需不需要这个功能，你也许会想还是实现好一些，万一以后要咋办,其实不需要实现，意思是不用命令模式。敏捷开发原则告诉我们，不要为代码添加基于猜测的，实际不需要的功能，如果不清楚该系统是否需要命令模式，一般不要着急去实现它，因为在需要的时候重构这个模式并不是太难，只有在真正需要如撤销/恢复的操作等功能时，把原来的代码重构为命令模式才有意义。