[C#设计模式系列：命令模式（Command）](http://www.cnblogs.com/libingql/p/3651624.html)

**1、命令模式简介**

1.1>、定义

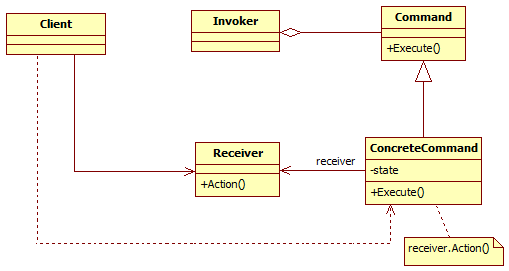
　　命令模式的目的是解除命令发出者和接收者之间的紧密耦合关系，使二者相对独立，有利于程序的并行开发和代码的维护。命令模式的核心思想是将请求封装为一个对象，将其作为命令发起者和接收者的中介，而抽象出来的命令对象又使得能够对一系列请求进行操作，如对请求进行排队，记录请求日志以及支持可撤销的操作等。

1.2>、使用频率

http://images.cnitblog.com/blog/56886/201312/29144513-80252c8296d141e4950c09815aa236d0.gif 中高

**2、命令模式结构**

2.1>、结构图



2.2>、参与者

　　命令模式参与者：

　　◊ Command：命令抽象类，声明一个执行操作的接口Execute，该抽象类并不实现这个接口，所有的具体命令都继承自命令抽象类。

　　◊ ConcreteCommand

　　　　° 定义一个接收者对象与动作之间的请求绑定

　　　　° 调用Receiver的操作，实现Execute方法

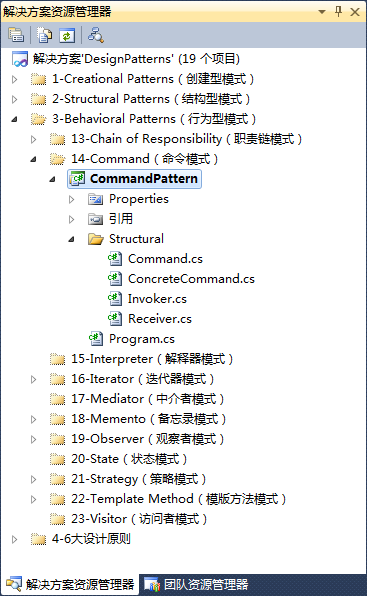
　　◊ Invoker：命令的接收者，将命令请求传递给相应的命令对象，每个ConcreteCommand都是一个Invoker的成员

　　◊ Receiver：命令的接收者，知道如何实施与执行一个请求相关的操作

　　◊ Client：客户端程序，创建一个具体命令对象并设定它的接收者

　　Command对象作为Invoker的一个属性，当点击事件发生时，Invoker调用方法Invoke()将请求发送给ConcreteCommand，再由ConcreteCommand调用Execute()将请求发送给Receiver，Client负责创建所有的角色，并设定Command与Invoker和Receiver之间的绑定关系。

**3、命令模式结构实现**

****

Receiver.cs

using System;

namespace CommadMode.Structural

{

/// <summary>

/// 命令的接收者

/// <para>知道如何实施与执行一个请求相关的操作</para>

/// </summary>

public class Receiver

{

public void Action()

{

Console.WriteLine("Called Receiver.Action()");

}

}

}



Command.cs

[复制代码](javascript:void(0);)

namespace CommadMode.Structural

{

/// <summary>

/// 命令抽象类

/// </summary>

public abstract class Command

{

protected Receiver receiver;

public Command(Receiver receiver)

{

this.receiver = receiver;

}

public abstract void Execute();

}

}

[复制代码](javascript:void(0);)

ConcreteCommand.cs



namespace CommadMode.Structural

{

/// <summary>

/// 具体命令

/// </summary>

public class ConcreteCommand : Command

{

public ConcreteCommand(Receiver receiver)

: base(receiver)

{

}

public override void Execute()

{

receiver.Action();

}

}

}

[复制代码](javascript:void(0);)

　　Invoker.cs

[复制代码](javascript:void(0);)

namespace CommadMode.Structural

{

/// <summary>

/// 命令的接收者

/// <para>将命令请求传递给相应的命令对象，每个ConcreteCommand都是一个Invoker的成员</para>

/// </summary>

public class Invoker

{

private Command \_command;

public void SetCommand(Command command)

{

this.\_command = command;

}

public void ExecuteCommand()

{

\_command.Execute();

}

}

}

[复制代码](javascript:void(0);)

　　Program.cs

[复制代码](javascript:void(0);)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using DesignPatterns.CommandPattern.Structural;

namespace DesignPatterns.CommandPattern

{

class Program

{

static void Main(string[] args)

{

Receiver receiver = new Receiver();

Command command = new ConcreteCommand(receiver);

Invoker invoker = new Invoker();

invoker.SetCommand(command);

invoker.ExecuteCommand();

}

}

}

[复制代码](javascript:void(0);)

　　运行输出：

Called Receiver.Action()

请按任意键继续. . .

**4、命令模式应用分析**

　　命令模式适用情形：

　　1>、将用户界面和行为分离，使两者的开发可以并行不悖。

　　2>、在需要指定、排列和执行一系列请求的情况下，适用命令模式。

　　3>、支持修改日志。

　　命令模式优点：

　　1>、命令模式将调用操作对象和知道如何实现该操作对象的解耦。

　　2>、在Command要增加新的处理操作对象很容易，可以通过创建新的继承自Command的子类来实现。

　　3>、命令模式可以和Memento模式结合使用，支持取消的操作。

　　4>、支持日志、请求队列和复合命令。