主讲：毛飞飞
成员：刘明超
陆新龙、李征

20、责任链模式

CHAIN OF RESPONSIBLEITY—晚上去上英语课，为了好开溜坐到了最后一排，哇，前面坐了好几个漂亮的MM哎，找张纸条，写上"Hi,可以做我的女朋友吗？如果不愿意请向前传"，纸条就一个接一个的传上去了，糟糕，传到第一排的MM把纸条传给老师了，听说是个老处女呀，快跑!

责任链模式：在责任链模式中，很多对象由每一个对象对其下家的引用而接起来形成一条链。请求在这个链上传递，直到链上的某一个对象决定处理此请求。客户并不知道链上的哪一个对象最终处理这个请求，系统可以在不影响客户端的情况下动态的重新组织链和分配责任。处理者有两个选择：承担责任或者把责任推给下家。一个请求可以最终不被任何接收端对象所接受。

第九章 责任链模式

责任链模式

使多个对象都有机会处理请求，从而避免请求的发送者和接收者之间的耦合关系。将这些对象连成一条链，并沿着这条链传递该请求，直到有一个对象处理它为止。

Chain of Responsibility Pattern

Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

一、概述

责任链模式是使用多个对象处理用户请求的成熟模式，责任链模式的关键是将用户的请求分派给许多对象，这些对象被组织成一个责任链，即每个对象含有后继对象的引用，并要求责任链上的每个对象，如果能处理用户的请求，就做出处理，不再将用户的请求传递给责任链上的下一个对象；如果不能处理用户的请求，就必须将用户的请求传递给责任链上的下一个对象。

二、责任链模式的结构与使用

模式的结构中包括两种角色：

- 处理者（Handler）
- 具体处理者（ConcreteHandler）

模式的UML类图

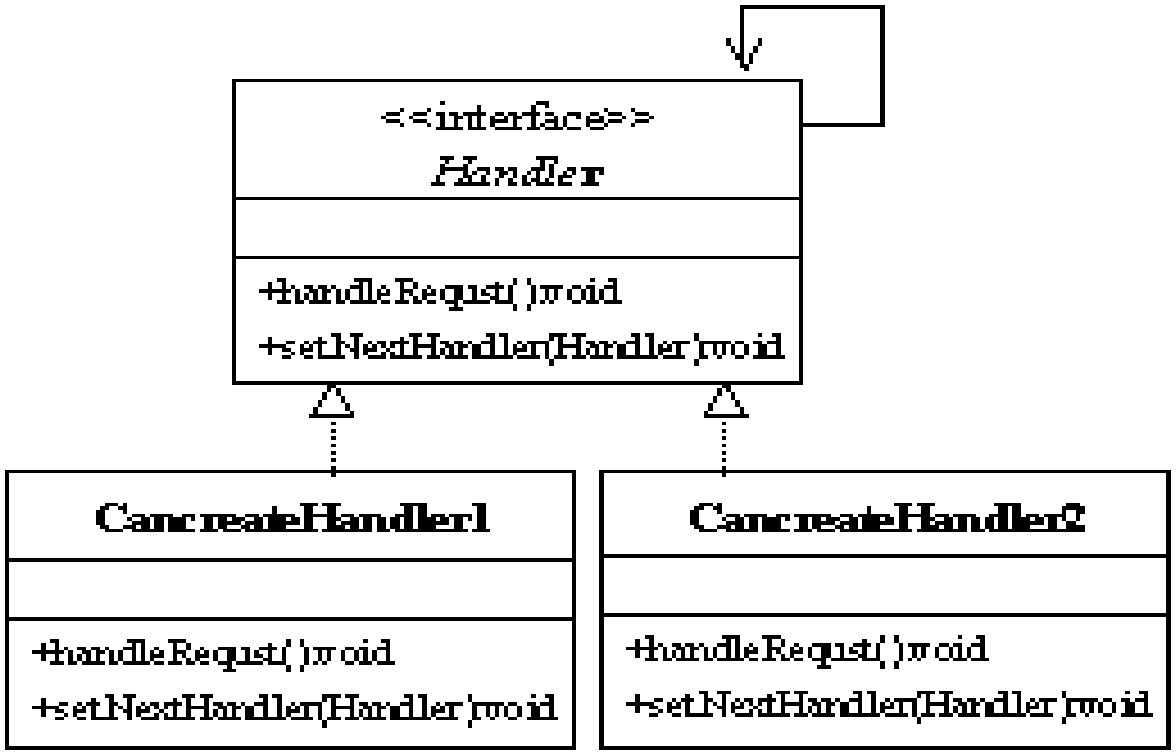


图9.2 责任链模式的类图

模式的结构的描述与使用

1．处理者（Handler）：Handler.java

```
public interface Handler{
   public abstract void handleRequest(String number);
   public abstract void setNextHandler(Handler handler);
}
```

2. 具体处理者（ConcreteHandler）_1: Beijing.java

```
import java.util.*;
public class Beijing implements Handler{
    private Handler handler;
    private ArrayList<String> numberList;
    Beijing(){
      numberList=new ArrayList<String>();
      numberList.add("11129812340930034");
      numberList.add("10120810340930632");
      numberList.add("22029812340930034");
      numberList.add("32620810340930632");
    }
    public void handleRequest(String number){
      if(numberList.contains(number))
        System.out.println("该人在北京居住");
      else{
        System.out.println("该人不在北京居住");
        if(handler!=null)
          handler.handleRequest(number);
      }
    }
    public void setNextHandler(Handler handler){
      this.handler=handler;
    }
}
```

2．具体处理者（ConcreteHandler）_2: Shanghai.java

```
import java.util.*;
public class Shanghai implements Handler{
    private Handler handler;
    private ArrayList<String> numberList;
    Shanghai(){
      numberList=new ArrayList<String>();
      numberList.add("34529812340930034");
      numberList.add("98720810340430632");
      numberList.add("36529812340930034");
      numberList.add("77720810340930632");
    }
    public void handleRequest(String number){
      if(numberList.contains(number))
        System.out.println("该人在上海居住");
      else{
        System.out.println("该人不在上海居住");
        if(handler!=null)
          handler.handleRequest(number);
      }
    }
    public void setNextHandler(Handler handler){
      this.handler=handler;
    }
```

2．具体处理者（ConcreteHandler）_3: Tianjin.java

```
import java.util.*;
public class Tianjin implements Handler{
    private Handler handler;
    private ArrayList<String> numberList;
    Tianjin(){
      numberList=new ArrayList<String>();
      numberList.add("10029812340930034");
      numberList.add("20020810340430632");
      numberList.add("30029812340930034");
      numberList.add("50020810340930632");
    }
    public void handleRequest(String number){
      if(numberList.contains(number))
        System.out.println("该人在天津居住");
      else{
        System.out.println("该人不在天津居住");
        if(handler!=null)
          handler.handleRequest(number);
      }
    }
    public void setNextHandler(Handler handler){
      this.handler=handler;
    }
}
```

3. 应用 Application.java

```
public class Application{
  private Handler beijing,shanghai,tianjin;
  public void createChain(){
    beijing=new Beijing();
    shanghai=new Shanghai();
    tianjin=new Tianjin();
    beijing.setNextHandler(shanghai);
    shanghai.setNextHandler(tianjin);
  }
  public void reponseClient(String number){
    beijing.handleRequest(number);
  }
  public static void main(String args[]){
    Application  application=new  Application();
    application.createChain();
    application.reponseClient("77720810340930632");;
  }
}
```

三、责任链模式的优点

- 责任链中的对象只和自己的后继是低耦合关系，和其他对象毫无关联，这使得编写处理者对象以及创建责任链变得非常容易。
- 当在处理者中分配职责时，责任链给应用程序更多的灵活性。
- 应用程序可以动态地增加、删除处理者或重新指派处理者的职责。
- 应用程序可以动态地改变处理者之间的先后顺序。
- 使用责任链的用户不必知道处理者的信息，用户不会知道到底是哪个对象处理了它的请求。

责任链模式概述

责任链模式是使用多个对象处理用户请求的成熟模式，责任链模式的关键是将用户的请求分派给许多对象，这些对象被组织成一个责任链，即每个对象含有后继对象的引用，并要求责任链上的每个对象，如果能处理用户的请求，就做出处理，不能处理就必须将用户的请求传递给责任链上的下一个对象。

责任链模式的结构

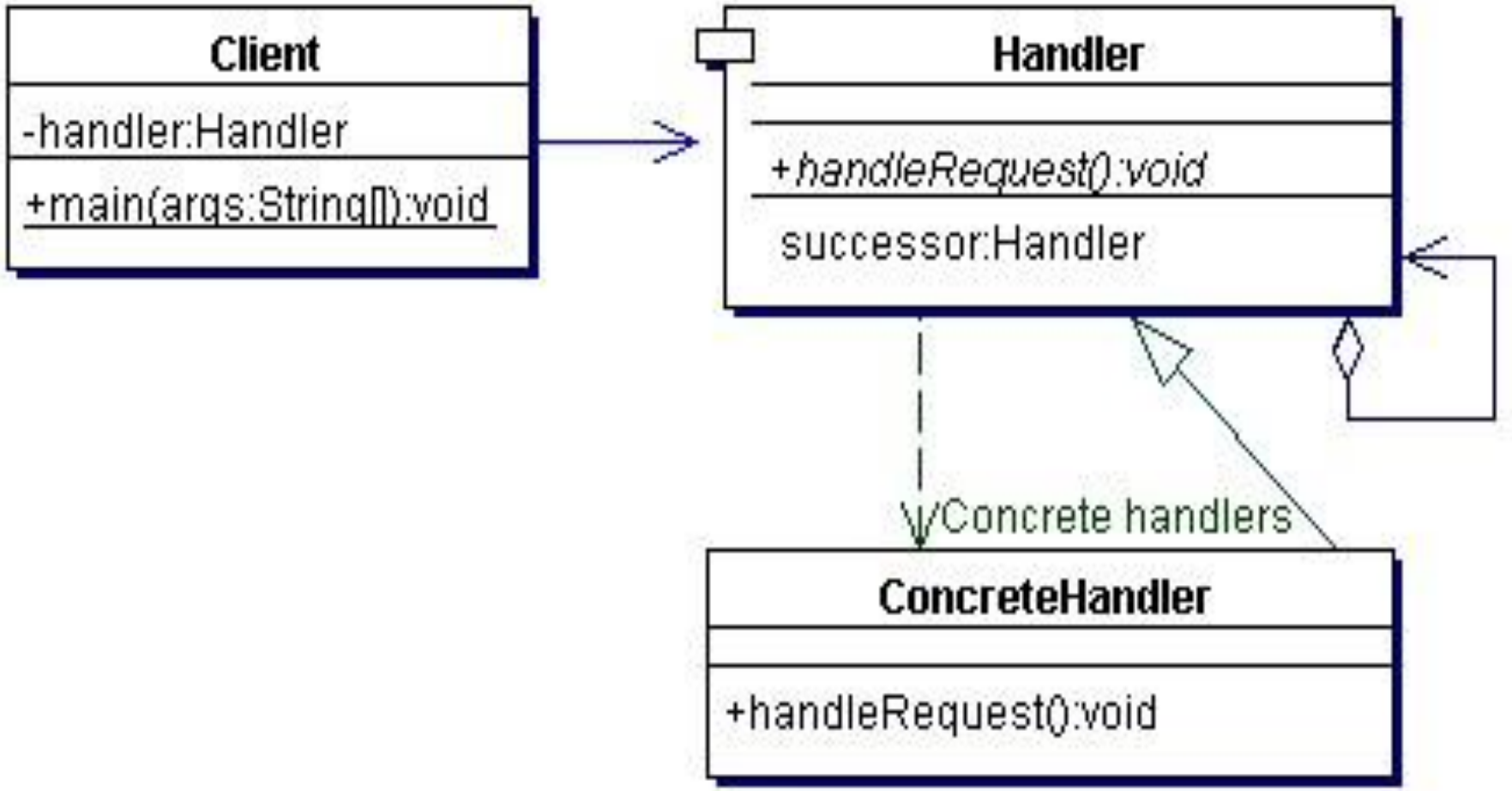
责任链模式是一种对象的行为模式，它所涉及到的角色如下：

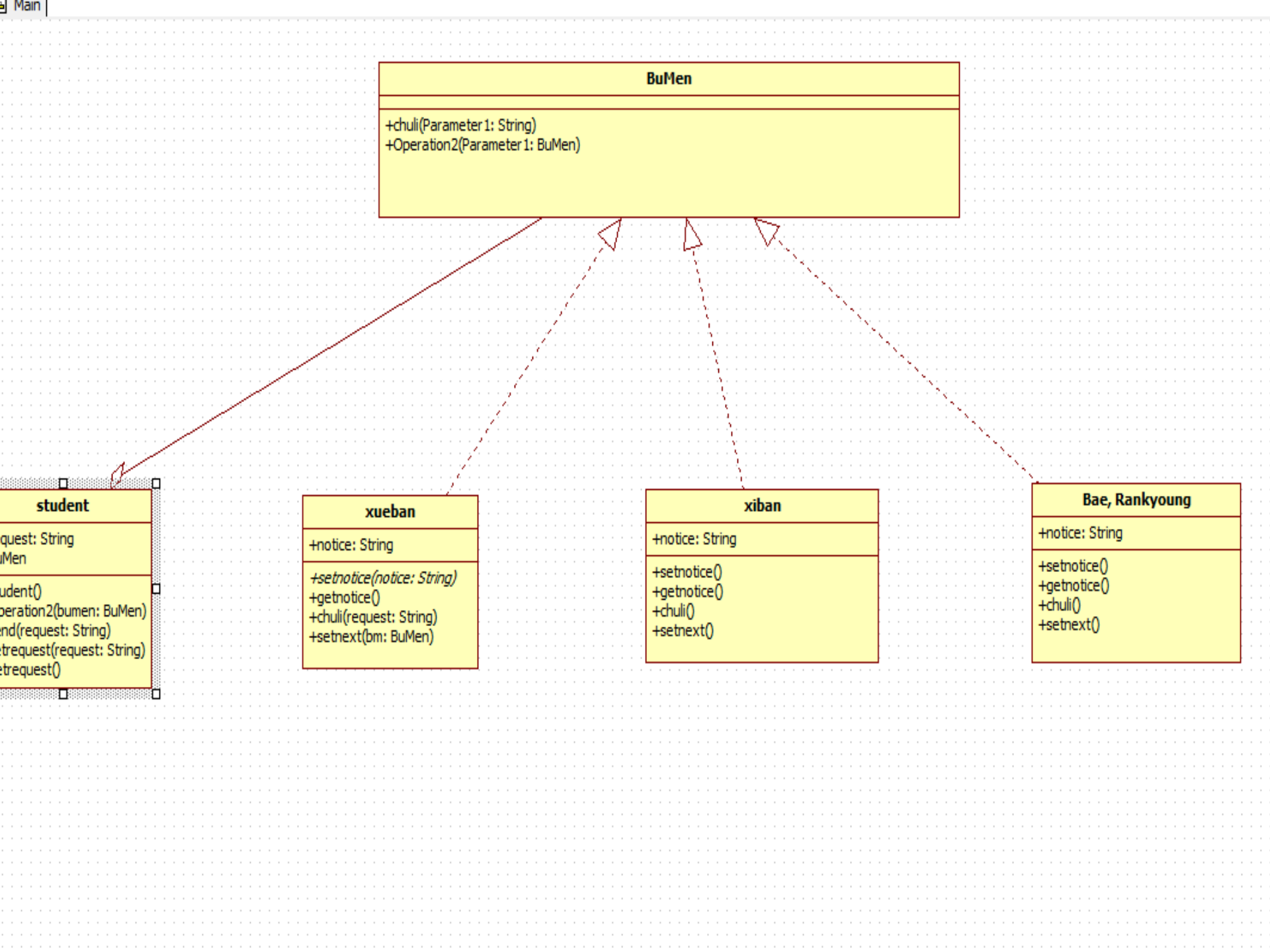
第一、抽象处理者（Handler）角色

定义出一个处理请求的接口；如果需要，接口可以定义出一个方法，以返回对下家的引用。下图给出了一个示意性的类图：

第二、具体处理者（ConcreteHandler）角色、处理接到请求后，可以选择将请求处理掉，或者将请 求传给下家。下图给出了一个示意性的类图。

责任链模式的静态类结构




Main
BuMen
+chuli(Parameter1: String)
+Operation2(Parameter1: BuMen)
student
quest: String
uMen
udent()
peration2(bumen: BuMen)
nd(request: String)
trequest(request: String)
trequest()
xueban
+notice: String
+setnotice(notice: String)
+getnotice()
+chuli(request: String)
+setnext(bm: BuMen)
xiban
+notice: String
+setnotice()
+getnotice()
+chuli()
+setnext()
Bae, Rankyoung
+notice: String
+setnotice()
+getnotice()
+chuli()
+setnext()

纯的与不纯的责任链模式

一个纯的责任链模式要求一个具体的处理者对象只能在两个行为中选择一个：一是承担责任，二是把责任推给下家。不允许出现某一个具体处理者对象在承担了一部分责任后又把责任向下传的情况。

在一个纯的责任链模式里面，一个请求必须被某一个处理者对象所接受；在一个不纯的责任链模式里面，一个请求可以最终不被任何接受端对象所接受。

纯的责任链模式的实际例子很难找到，一般看到的例子均是不纯的责任链模式的实现。

下面的情况下使用责任链模式

第一、系统已经有一个由处理者对象组成的链。这个链可能由复合模式给出，

第一、当有多于一个的处理者对象会处理一个请求，而且在事先并不知道到底由哪一个处理者对象处理一个请求。这个处理者对象是动态确定的。

第二、当系统想发出一个请求给多个处理者对象中的某一个，但是不明显指定是哪一个处理者对象会处理此请求。

第三、当处理一个请求的处理者对象集合需要动态地指定时。

责任链模式与其它模式的关系

复合模式（Composite Pattern） 当责任链模式中的对象链属于一个较大的结构时，这个较大的结构可能符合复合模式。

命令模式（Command Pattern） 责任链模式使一个特定的请求接收对象对请求或命令的执行变得不确定。而命令模式使得一个特定的对象对一个命令的执行变得明显和确定。

模版方法模式（Template Method） 当组成责任链的处理者对象是按照复合模式组成一个较大的结构的责成部分的话，模版方法模式经常用来组织单个的对象的行为。

使用责任链模式的长处

责任链模式减低了发出命令的对象和处理命令的对象之间的耦合，它允许多与一个的处理者对象根据自己的逻辑来决定哪一个处理者最终处理这个命令。换言之，发出命令的对象只是把命令传给链结构的起始者，而不需要知道到底是链上的哪一个节点处理了这个命令。

显然，这意味着在处理命令上，允许系统有更多的灵活性。哪一个对象最终处理一个命令可以因为由那些对象参加责任链、以及这些对象在责任链上的位置不同而有所不同。

使用责任链模式的不足

责任链模式可能会带来一些额外的性能损耗，因为它要从链子开头开始遍历。