



深圳大学
Shenzhen University

第11讲

职责链模式

软件体系结构与设计模式

Software Architecture & Design Pattern

深圳大学计算机与软件学院



主要内容

- ◆ 行为型模式
- ◆ 职责链模式动机与定义
- ◆ 职责链模式结构与分析
- ◆ 职责链模式实例与解析
- ◆ 职责链模式效果与应用

行为型模式概述

■ 行为型模式

- **行为型模式(Behavioral Pattern)** 关注系统中对象之间的交互，研究系统在运行时对象之间的相互通信与协作，进一步明确对象的职责
- 行为型模式：不仅仅关注类和对象本身，还重点关注它们之间的相互作用和职责划分



行为型模式概述

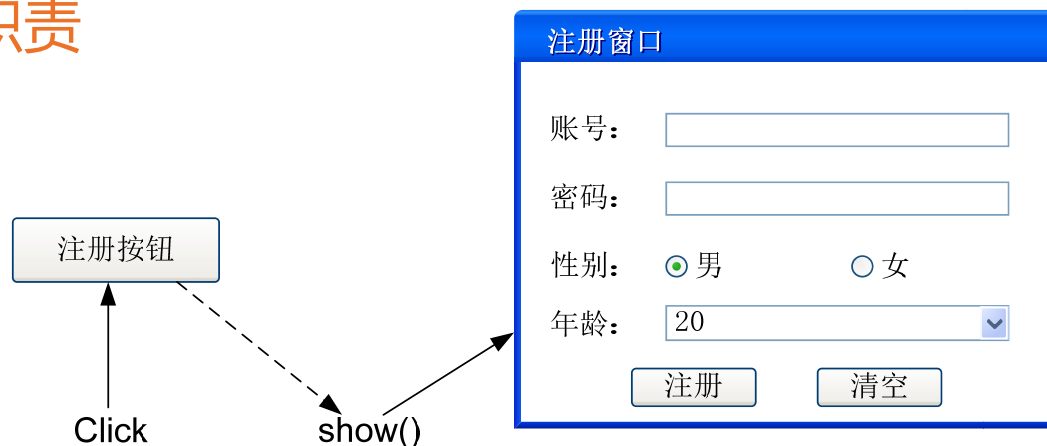
■ 行为型模式分类

□ 类行为型模式

- 使用**继承关系**在几个类之间分配行为，主要**通过多态等方式来分配父类与子类的职责**

□ 对象行为型模式

- 使用对象的**关联关系**来分配行为，主要**通过对象关联等方式来分配两个或多个类的职责**



行为型模式一览表

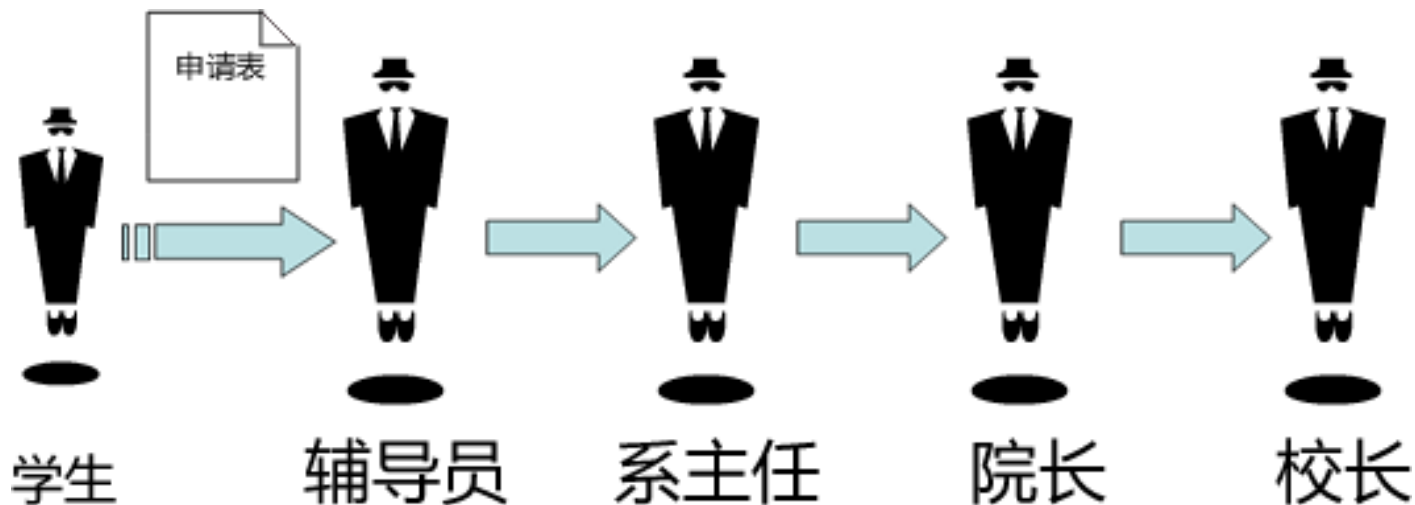
模式名称	定义	学习难度	使用频率
职责链模式 (Chain of Responsibility Pattern)	避免将一个请求的发送者与接收者耦合在一起，让多个对象都有机会处理请求。将接收请求的对象连接成一条链，并且沿着这条链传递请求，直到有一个对象能够处理它为止。	★★★★☆	★★☆☆☆
命令模式 (Command Pattern)	将一个请求封装为一个对象，从而让你可以用不同的请求对客户进行参数化，对请求排队或者记录请求日志，以及支持可撤销的操作。	★★★★☆	★★★★☆
解释器模式 (Interpreter Pattern)	给定一个语言，定义它的文法的一种表示，并定义一个解释器，这个解释器使用该表示来解释语言中的句子。	★★★★★	★☆☆☆☆
迭代器模式 (Iterator Pattern)	提供一种方法顺序访问一个聚合对象中的各个元素，且不用暴露该对象的内部表示。	★★★★☆	★★★★★
中介者模式 (Mediator Pattern)	定义一个对象来封装一系列对象的交互。中介者模式使各对象之间不需要显式地相互引用，从而使其耦合松散，而且让你可以独立地改变它们之间的交互。	★★★★☆	★★☆☆☆

行为型模式一览表

模式名称	定义	学习难度	使用频率
备忘录模式 (Memento Pattern)	在不破坏封装的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态，这样可以在以后将对象恢复到原先保存的状态。	★★★★☆	★★★★☆
观察者模式 (Observer Pattern)	定义对象之间的一种一对多依赖关系，使得每当一个对象状态发生改变时，其相关依赖对象都得到通知并被自动更新。	★★★★☆	★★★★★
状态模式 (State Pattern)	允许一个对象在其内部状态改变时改变它的行为。对象看起来似乎修改了它的类。	★★★★☆	★★★★☆
策略模式 (Strategy Pattern)	定义一系列算法，将每一个算法封装起来，并让它们可以相互替换，策略模式让算法可以独立于使用它的客户变化。	★★★☆☆	★★★★☆
模板方法模式 (Template Method Pattern)	定义一个操作中算法的框架，而将一些步骤延迟到子类中。模板方法模式使得子类不改变一个算法的结构即可重定义该算法的某些特定步骤。	★★★★☆	★★★★☆
访问者模式 (Visitor Pattern)	表示一个作用于某对象结构中的各个元素的操作。访问者模式让你可以在不改变各元素的类的前提下定义作用于这些元素的新操作。	★★★★☆	★★★☆☆

职责链模式动机

■ 奖学金审批



- 辅导员、系主任、院长、校长都可以处理奖学金申请表，他们构成一个处理申请表的链式结构，申请表沿着这条链进行传递，这条链就称为职责链
- 职责链可以是一条直线、一个环或者一个树形结构，最常见的职责链是直线型，即沿着一条单向的链来传递请求



职责链模式定义

■ 对象行为型模式

职责链模式：避免将一个请求的发送者与接收者耦合在一起，让多个对象都有机会处理请求。将接收请求的对象连接成一条链，并且沿着这条链传递请求，直到有一个对象能够处理它为止。

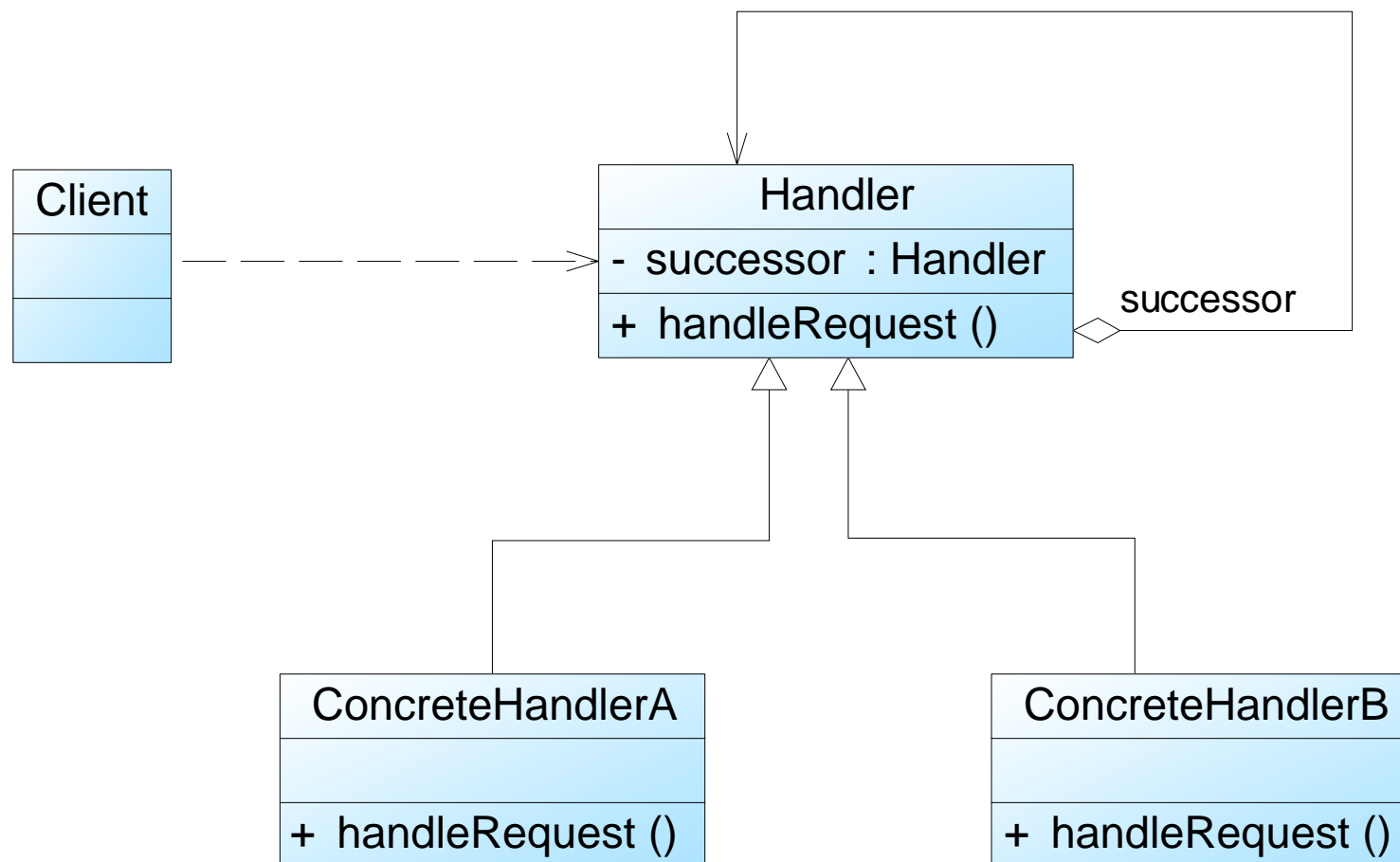
Chain of Responsibility Pattern: Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

- 将请求的处理者组织成一条链，并让请求沿着链传递，由链上的处理者对请求进行相应的处理
- 客户端无须关心请求的处理细节以及请求的传递，只需将请求发送到链上，将请求的发送者和请求的处理者解耦

职责链模式结构

■ 职责链模式包含如下角色：

- Handler: 抽象处理者
- ConcreteHandler: 具体处理者





职责链模式分析

■ 抽象处理者示例代码：

```
public abstract class Handler {  
    //维持对下家的引用  
    protected Handler successor;  
  
    public void setSuccessor(Handler successor) {  
        this.successor=successor;  
    }  
  
    public abstract void handleRequest(String request);  
}
```



职责链模式分析

■ 具体处理者示例代码：

```
public class ConcreteHandler extends Handler {  
    public void handleRequest(String request) {  
        if (请求满足条件) {  
            //处理请求  
        }  
        else {  
            this.successor.handleRequest(request); //转发请求  
        }  
    }  
}
```



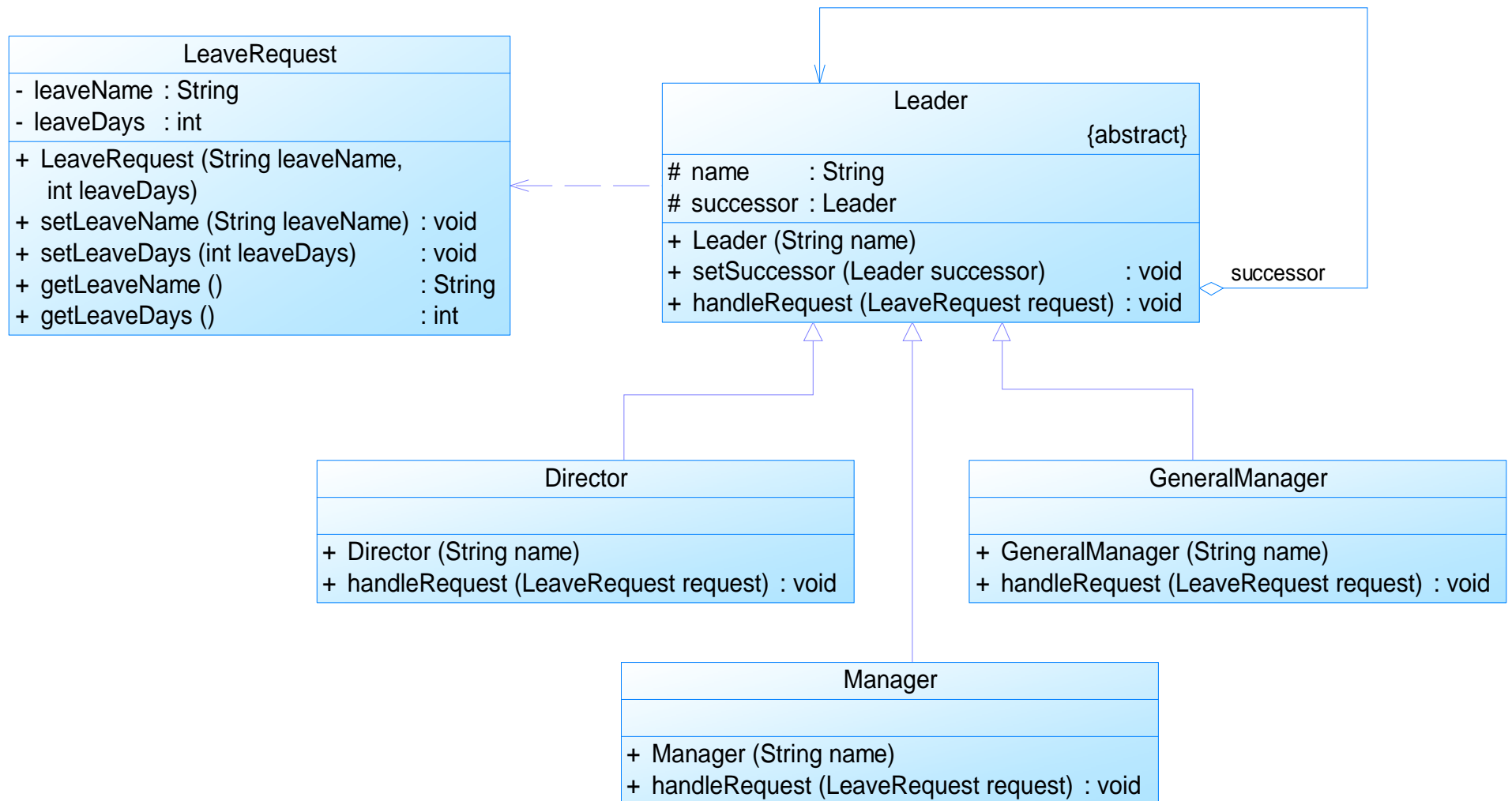
职责链模式实例

■ 审批假条：实例说明

- 某OA系统需要提供一个假条审批的模块，如果员工请假天数小于3天，主任可以审批该假条；如果员工请假天数大于等于3天，小于10天，经理可以审批；如果员工请假天数大于等于10天，小于30天，总经理可以审批；如果超过30天，总经理也不能审批，提示相应的拒绝信息。

职责链模式实例与解析

■ 审批假条：参考类图



职责链模式实例与解析

■ 职责链模式实例

□ 审批假条：参考代码

■ DesignPatterns之cor包

```
Leader.java
1 package cor;
2
3 public abstract class Leader
4 {
5     protected String name;
6     protected Leader successor;
7     public Leader(String name)
8     {
9         this.name=name;
10    }
11    public void setSuccessor(Leader successor)
12    {
13        this.successor=successor;
14    }
15    public abstract void handleRequest(LeaveRequest request);
16 }
```

```
3 public class LeaveRequest
4 {
5     private String leaveName;
6     private int leaveDays;
7
8     public LeaveRequest(String leaveName, int leaveDays)
9     {
10         this.leaveName=leaveName;
11         this.leaveDays=leaveDays;
12     }
13
14     public void setLeaveName(String leaveName) {
15         this.leaveName = leaveName;
16     }
17
18     public void setLeaveDays(int leaveDays) {
19         this.leaveDays = leaveDays;
20     }
21
22     public String getLeaveName() {
23         return (this.leaveName);
24     }
25
26     public int getLeaveDays() {
27         return (this.leaveDays);
28     }
29 }
```

Director.java

```
2
3 public class Director extends Leader
4 {
5     public Director(String name)
6     {
7         super(name);
8     }
9     public void handleRequest(LeaveRequest request)
10    {
11        if(request.getLeaveDays()<3)
12        {
13            System.out.println("主任" + name + "审批员工" +
14            request.getLeaveName() + "的请假条, 请假天数为" + request.getLeaveDays() + "天。");
15        }
16        else
17        {
18            if(this.successor!=null)
19            {
20                this.successor.handleRequest(request);
21            }
22        }
23    }
24 }
```



```
1 package com;  
2  
3 public class Manager extends Leader  
4 {  
5     public Manager(String name)  
6     {  
7         super(name);  
8     }  
9     public void handleRequest(LeaveRequest request)  
10    {  
11        if(request.getLeaveDays()<10)  
12        {  
13            System.out.println("经理" + name + "审批员工" +  
14            request.getLeaveName() + "的请假条, 请假天数为" +  
15            request.getLeaveDays() + "天。");  
16        }  
17        else  
18        {  
19            if(this.successor!=null)  
20            {  
21                this.successor.handleRequest(request);  
22            }  
23        }  
24    }  
25 }
```

ViceGeneralManager.java

```
1 package cor;
2
3 public class ViceGeneralManager extends Leader
4 {
5     public ViceGeneralManager(String name)
6     {
7         super(name);
8     }
9     public void handleRequest(LeaveRequest request)
10    {
11        if(request.getLeaveDays()<20)
12        {
13            System.out.println("副总经理" + name + "审批员工" +
14            request.getLeaveName() + "的请假条, 请假天数为" + request.getLeaveDays() + "天。");
15        }
16        else
17        {
18            if(this.successor!=null)
19            {
20                this.successor.handleRequest(request);
21            }
22        }
23    }
24 }
```

GeneralManager.java

```
1 package cor;
2
3 public class GeneralManager extends Leader
4 {
5     public GeneralManager(String name)
6     {
7         super(name);
8     }
9
10    public void handleRequest(LeaveRequest request)
11    {
12        if(request.getLeaveDays() < 30)
13        {
14            System.out.println("总经理" + name + "审批员工" +
15            request.getLeaveName() + "的请假条, 请假天数为" + request.getLeaveDays() + "天。");
16        }
17        else
18        {
19            System.out.println("莫非" + request.getLeaveName() +
20            "想辞职, 居然请假" + request.getLeaveDays() + "天。");
21        }
22    }
23 }
```

```
3 public class Client
4 {
5     public static void main(String args[])
6     {
7         Leader objDirector,objManager,objGeneralManager;
8         //Leader objDirector,objManager,objGeneralManager,objViceGeneralManager;
9
10        objDirector=new Director("王明");
11        objManager=new Manager("赵强");
12        objGeneralManager=new GeneralManager("李波");
13        //objViceGeneralManager=new ViceGeneralManager("肖红");
14
15        objDirector.setSuccessor(objManager);
16        //objManager.setSuccessor(objViceGeneralManager);
17        objManager.setSuccessor(objGeneralManager);
18        //objViceGeneralManager.setSuccessor(objGeneralManager);
19        LeaveRequest lr1=new LeaveRequest("张三",2);
20        objDirector.handleRequest(lr1);
21        LeaveRequest lr2=new LeaveRequest("李四",5);
22        objDirector.handleRequest(lr2);
23        LeaveRequest lr3=new LeaveRequest("王五",15);
24        objDirector.handleRequest(lr3);
25        LeaveRequest lr4=new LeaveRequest("赵六",25);
26        objDirector.handleRequest(lr4);
27    }
28 }
```



职责链模式效果与应用

■ 职责链模式优点：

- 使得一个对象无须知道是其他哪一个对象处理其请求，降低了系统的耦合度
- 可简化对象之间的相互连接
- 给对象职责的分配带来更多的灵活性
- 增加一个新的具体请求处理者时无须修改原有系统的代码，只需要在客户端重新建链即可



职责链模式效果与应用

■ 职责链模式缺点：

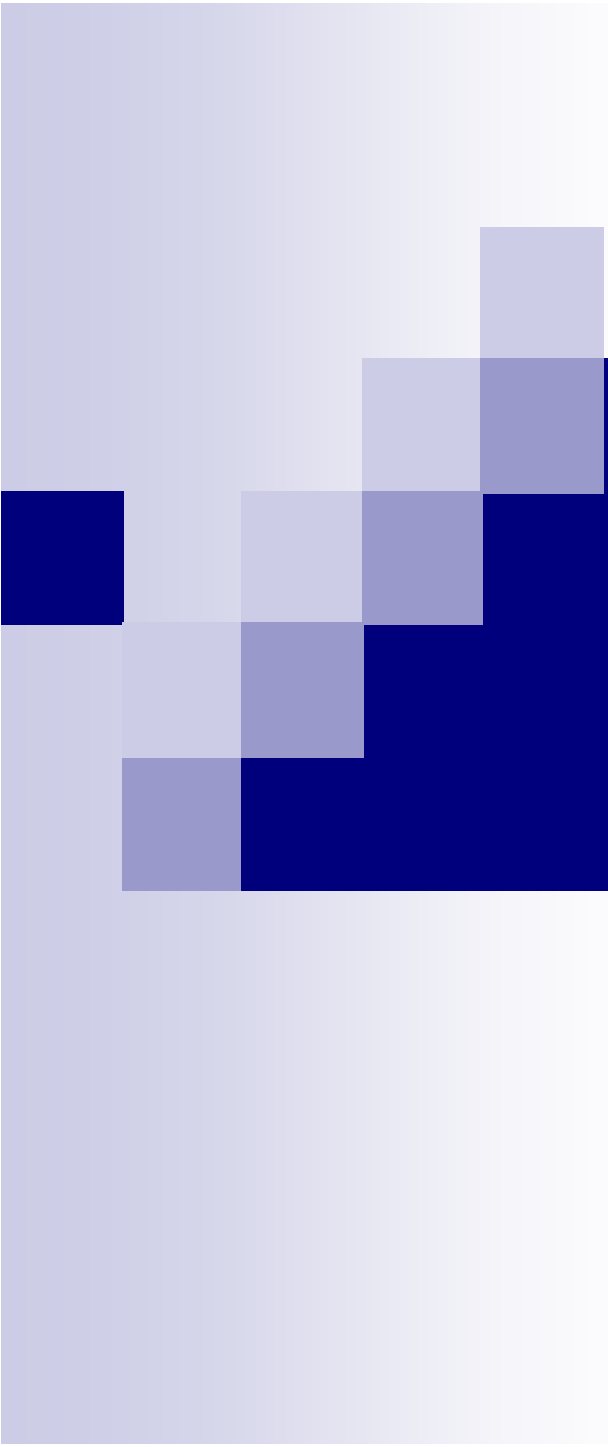
- 不能保证请求一定会被处理
- 对于比较长的职责链，系统性能将受到一定影响，在进行代码调试时不太方便
- 如果建链不当，可能会造成循环调用，将导致系统陷入死循环



职责链模式效果与应用

■ 在以下情况下可以使用职责链模式：

- 有多个对象可以处理同一个请求，具体哪个对象处理该请求待运行时刻再确定
- 在不明确指定接收者的情况下，向多个对象中的一个提交一个请求
- 可动态指定一组对象处理请求



谢谢