11.职责链模式

责任链（Chain of Responsibility）模式的定义：为了避免请求发送者与多个请求处理者耦合在一起，将所有请求的处理者通过前一对象记住其下一个对象的引用而连成一条链；当有请求发生时，可将请求沿着这条链传递，直到有对象处理它为止。

学校 OA 系统的采购审批项目：需求是

采购员采购教学器材

1) 如果金额 小于等于 5000, 由教学主任审批 （0<=x<=5000）

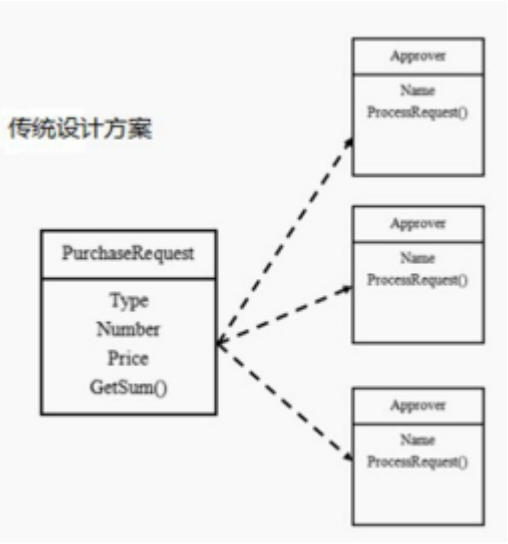
2) 如果金额 小于等于 10000, 由院长审批 (5000<x<=10000)

3) 如果金额 小于等于 30000, 由副校长审批 (10000<x<=30000)

4) 如果金额 超过 30000 以上，有校长审批 (30000<x)

请设计程序完成采购审批项目

传统方案解决 OA 系统审批，传统的设计方案(类图)



传统方案解决 OA 系统审批问题分析

1) 传统方式是：接收到一个采购请求后，根据采购金额来调用对应的 Approver (审批人)完成审批。

2) 传统方式的问题分析 ： 客户端这里会使用到 分支判断(比如 switch) 来对不同的采购请求处理， 这样就存在如下问题 (1) 如果各个级别的人员审批金额发生变化，在客户端的也需要变化 (2) 客户端必须明确的知道 有多少个审批级别和访问

3) 这样 对一个采购请求进行处理 和 Approver (审批人) 就存在强耦合关系，不利于代码的扩展和维护

4) 解决方案 =》 职责链模式

职责链模式基本介绍

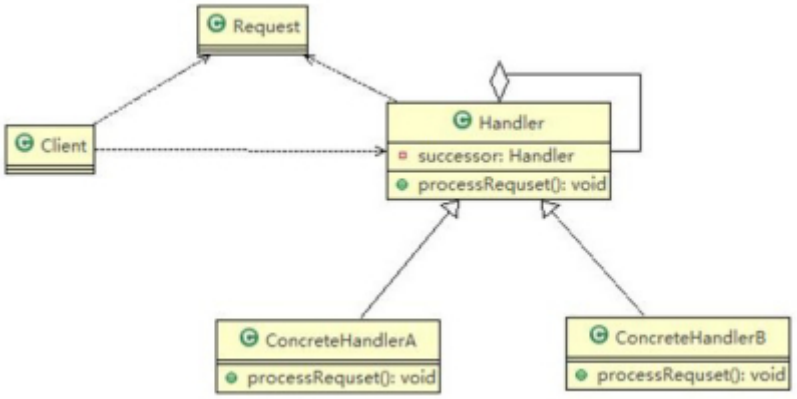
基本介绍

1) 职责链模式（Chain of Responsibility Pattern），又叫 责任链模式，为请求创建了一个接收者对象的链(简单示意图)。这种模式对请求的发送者和接收者进行解耦。

2) 职责链模式通常每个接收者都包含对另一个接收者的引用。如果一个对象不能处理该请求，那么它会把相同的请求传给下一个接收者，依此类推。

3) 这种类型的设计模式属于行为型模式

职责链模式的原理类图



对原理类图的说明-即(职责链模式的角色及职责)

1) Handler : 抽象的处理者，定义了一个处理请求的接口，同时含义另外 Handler

2) ConcreteHandlerA , B 是具体的处理者, 处理它自己负责的请求， 可以访问它的后继者(即下一个处理者)，如果可以处理当前请求，则处理，否则就将该

请求交个 后继者去处理，从而形成一个职责链

3) Request， 含义很多属性，表示一个请求

职责链模式解决 OA 系统采购审批

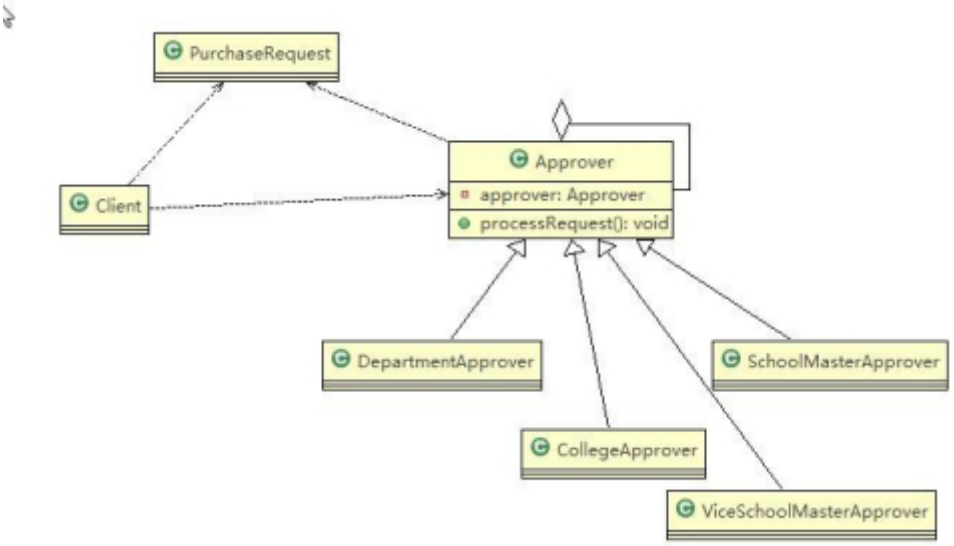
1) 应用实例要求

编写程序完成学校 OA 系统的采购审批项目：需求采购员采购教学器材

如果金额 小于等于 5000, 由教学主任审批如果金额 小于等于 10000, 由院长审批

如果金额 小于等于 30000, 由副校长审批如果金额 超过 30000 以上，有校长审批

2) 思路分析和图解(类图)




```
//请求类
public class PurchaseRequest {
  private int type = 0; //请求类型
  private float price = 0.0f; //请求金额
  private int id = 0;
  //构造器
  public PurchaseRequest(int type, float price, int id) {
    this.type = type;
    this.price = price;
    this.id = id;
  }
  public int getType() {
    return type;
  }
  public float getPrice() {
    return price;
```

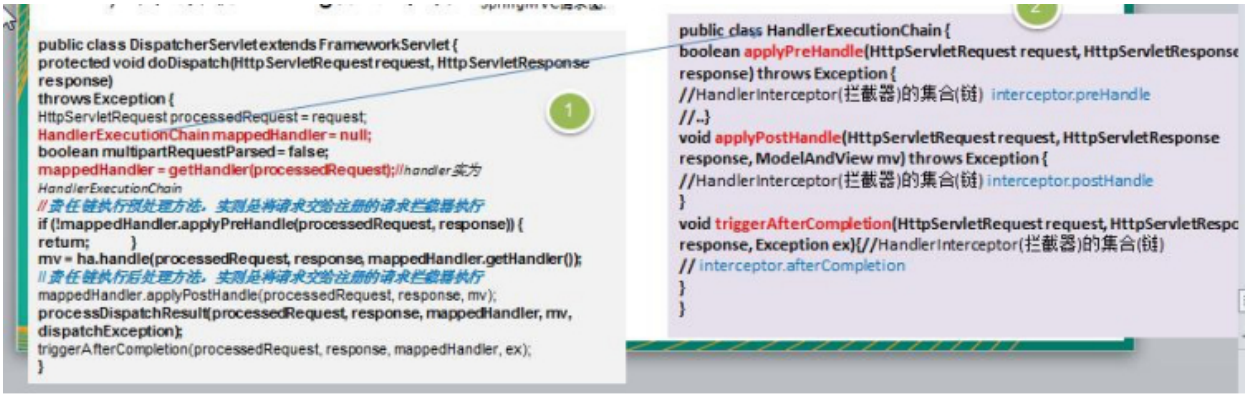
```
  }
  public int getId() {
    return id;
  }
}
public abstract class Approver {
  Approver approver;  //下一个处理者
  String name; // 名字

  public Approver(String name) {
    // TODO Auto-generated constructor stub
    this.name = name;
  }

  //下一个处理者
  public void setApprover(Approver approver) {
    this.approver = approver;
  }
  //处理审批请求的方法，得到一个请求, 处理是子类完成，因此该方法做成抽象
  public abstract void processRequest(PurchaseRequest purchaseRequest);

}
public class CollegeApprover extends Approver {

  public CollegeApprover(String name) {
    // TODO Auto-generated constructor stub
    super(name);
  }

  @Override
  public void processRequest(PurchaseRequest purchaseRequest) {
    // TODO Auto-generated method stub
    if(purchaseRequest.getPrice() < 5000 && purchaseRequest.getPrice() <= 10000)
{
      System.out.println(" 请求编号 id= " + purchaseRequest.getId() + " 被 " +
this.name + " 处理");
    }else {
      approver.processRequest(purchaseRequest);
    }
  }
}
public class DepartmentApprover extends Approver {}
public class SchoolMasterApprover extends Approver {}
public class ViceSchoolMasterApprover extends Approver {}
```

职责链模式在 SpringMVC 框架应用的源码分析

1) SpringMVC-HandlerExecutionChain 类就使用到职责链模式

2) SpringMVC 请求流程简图

3) 代码分析+Debug 源码+说明



1) 对源码总结

springmvc 请求的流程图中，执行了 拦截器相关方法 interceptor.preHandler 等等

在处理 SpringMvc 请求时，使用到职责链模式还使用到适配器模式

HandlerExecutionChain 主要负责的是请求拦截器的执行和请求处理,但是他本身不处理请求，只是将请求分配给链上注册处理器执行，这是职责链实现方式,减少职责链本身与处理逻辑之间的耦合,规范了处理流程

HandlerExecutionChain 维护了 HandlerInterceptor 的集合， 可以向其中注册相应的拦截器.

职责链模式的注意事项和细节

1) 将请求和处理分开，实现解耦，提高系统的灵活性

2) 简化了对象，使对象不需要知道链的结构

3) 性能会受到影响，特别是在链比较长的时候，因此需控制链中最大节点数量，一般通过在 Handler 中设置一个最大节点数量，在 setNext()方法中判断是否已经超过阀值，超过则不允许该链建立，避免出现超长链无意识地破坏系统性能

4) 调试不方便。采用了类似递归的方式，调试时逻辑可能比较复杂

5) 最佳应用场景：有多个对象可以处理同一个请求时，比如：多级请求、请假/加薪等审批流程、Java Web 中 Tomcat对 Encoding 的处理、拦截器

6)可以适当用循环链,防止在某一个环节进行处理