# 行为型模式

## 认识行为型模式

行为性模式（Behavior 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写作，进一步明确对象的职责。

在软件系统运行时对象并不是孤立存在的，它们可以通过相互通信协作完成某些功能，一个对象在运行时也将影响到其他对象的运行。

行为性模式（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互相通信与协作，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，还重点关注他们之间的相互作用和职责划分。

## 行为型模式的分类

行为型模式分为类行为型模式和对象行为型模式两种，其中类行为性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；而对象行为性模式则使用对象的关联关系来分配行为，主要通过队形关联等方式来分配两个或多个类的职责。根据合成复用原则，在系统中复用功能时要尽量使用关联关系来取代继承关系，因此大部分行为型设计模式都属于对象行为型设计模式。

## 行为型模式概览

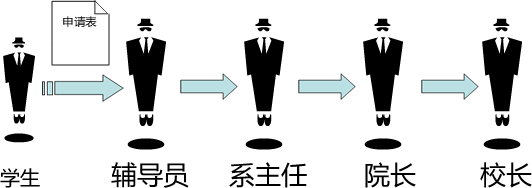
在GoF设计模式中包含11种行为型模式。如下：

|  |  |
| --- | --- |
| **模式名称** | **定义** |
| 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，知道有一个对象能够处理它为止 |
| 命令模式 | 将一个请求封装为一个对象，从而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 |
| 解释器模式 | 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 | 提供一种方法顺序访问一个聚合对象中的各个元素，而又不用暴露该对象的内部表示 |
| 中介者模式 | 定义一个对象来封装一系列对象的交互。中介者模式使个对象之间不需要显式地相互引用，从而使其耦合松散，而且可以独立地改变它们之间的交互 |
| 备忘录模式 | 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 | 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皆得到通知并被自动更新 |
| 状态模式 | 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 | 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而变化 |
| 模板方法模式 | 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可以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 | 表示一个作用于某对象结构中的各个元素的操作。访问者模式让用户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 |

# 职责链模式简介

## 概述

1. 在很多情况下可以处理某个请求的对象并不止一个，例如大学里的奖学金审批，学生在向辅导员提交审批表之后首先是辅导员签字审批，然后交给系主任签字审批，接着是院长审批，最后可能是校长审批，这个过程中奖学金申请表可以看成一个请求对象，而不同级别的审批者都可以处理该请求对象，除了辅导员外，学生不需要一一与其他审批者交互，只需要等待结果即可。在审批过程中如果某一个审批者认为不符合条件，则请求中止；否则将请求传递给下一个审批者，最后由校长来确定能够授予奖学金。该过程可以用如下示意图来表示：



在上述职责链模式图中，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。

1. 职责链可以是条一条直线、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。链上的每一个对象都是请求处理者，职责链模式可以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即可，将请求的发送者和请求的处理者解耦。这就是职责链模式的动机。
2. 职责链模式（Chain of Responsibility Pattern）的定义如下：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。

## 结构

职责链模式结构的核心在于引入了一个抽象处理者。其结构如下图所示：



由图可知，职责链模式包含以下两个角色。

1. Handler（抽象处理者）：它定义了一个处理请求的接口，一般设计为抽象类，由于不同的具体处理者处理请求的方式不同，因此在其中定义了抽象请求处理方法。每一个处理者的下家还是一个处理者，故在抽象处理者中定义了一个抽象处理者类型的对象（如结构图中的successor）作为其对下家的引用，通过该引用处理者可以练成一条链。
2. ConcreteHandler（具体处理者）：它是抽象处理者的子类，可以处理用户请求，在具体处理者类中实现了抽象处理者中定义的抽象请求处理方法，在处理请求之前需要进行判断，看是否有相应的处理权限，如果可以处理请求就处理它，否则将请求转发给后继者；在具体处理者中可以访问链中的下一个对象，以便请求的转发。

## 分类

职责链模式可分为纯的职责链模式和不纯的职责链模式两种。

1. **纯的职责链模式**

一个纯的职责链模式要求一个具体处理者对象只能在两个行为中选择一个，要么承担全部责任，要么将责任推给下家，不允许出现某一个具体处理者对象在承担了一部分或全部责任后又将责任向下传递的情况。而且纯的职责链模式中要求一个请求必须被某一个处理者对象所接收，不能出现某个请求未被任何一个处理者对象处理的情况。（下节实例中应用的是职责链模式）

1. **不纯的职责链模式**

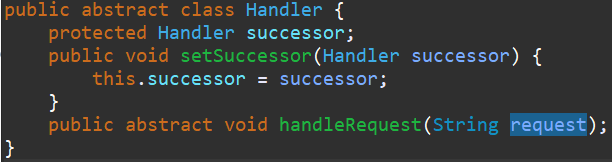
在一个不纯的职责链模式中允许某个请求被一个具体处理者部分处理后再向下传递，或者一个具体处理者处理完某处理请求后其后继处理者可以继续处理该请求，而且下一个请求可以最终不被任何处理者对象所接收并处理。

# 实现

## 实现原理

在职责链模式中很多对象由一个对象对其下家的引用连接起来形成一条链。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。发出这个请求的客户端并不知道链上的哪一个对象最终处理这个请求，这使得系统可以在不影响客户端的情况下动态地重新组织链和分配责任。

1. 职责链模式的核心在于抽象处理者类的设计，抽象处理者的典型代码如下：



在上述代码中，抽象处理者类定义了对下家的引用对象，以便将请求转发给下家，该对象的访问符可以设为protected，在其子类中可以使用；在抽象处理者类中声明了抽象的请求处理方法，具体实现交由子类完成。

1. 具体处理者是抽象处理者的子类，它有两个作用：一是处理请求，不同的具体处理者以不同的形式实现抽象请求处理方法handleRequest( )；二是转发请求，如果该请求超出了当前处理者类的权限，可以将该请求转发给下家。具体处理者类的典型代码如下：



在具体处理类中通过对请求进行判断可以做出相应的处理。

1. 需要注意的是，职责链模式并不负责创建职责链，职责链的创建工作必须由系统的其他部分来完成，一般是在使用该职责链的客户端中创建职责链。职责链模式降低了请求发送端和接收端之间的耦合，使多个对象都有机会处理这个请求。典型的客户端代码片段如下：



## 实例

实例说明：某企业的SCM（Supply Chain Management，供应链管理）系统中包含一个采购审批子系统。该企业的采购审批是分级进行的，即根据采购金额的不同由不同层次的主管人员来审批，主任可以审批5万元以下（不包括5万元）的采购单，副董事长可以审批5万元至10万元（不包括10万元）的采购单，董事长可以审批10万元至50万元（不包括50万元）的采购单，50万元及以上的采购单则需要开董事会讨论决定，如下图所示：

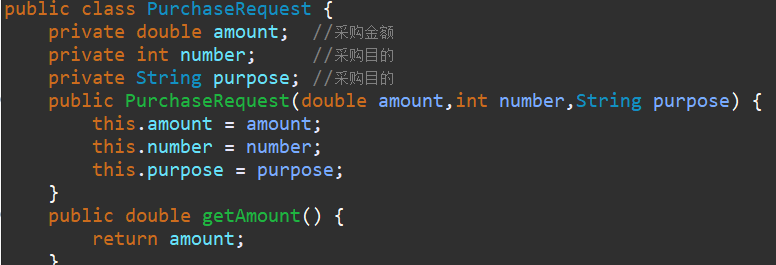


通过分析，本实例的结构图如下图所示：

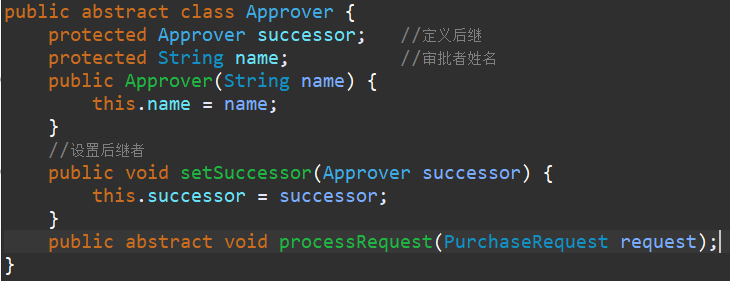


上图中，抽象类Approver充当抽象处理者，Director，VicePresident、President和Congress充当具体处理者，PurchaseRequest充当请求类。

1. PurchaseRequest：采购单类，充当请求类



1. Approver：审批者类，充当抽象处理者



1. Director：主任类，充当具体处理者



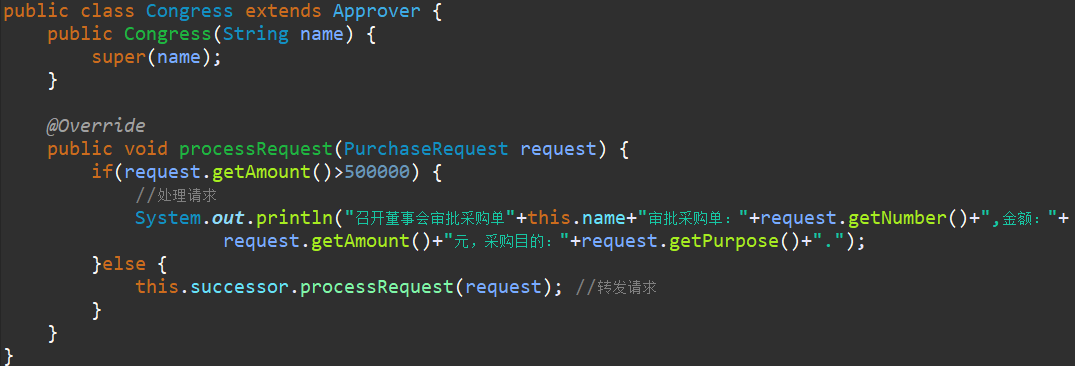
1. VicePresident：副董事长类，充当具体处理者



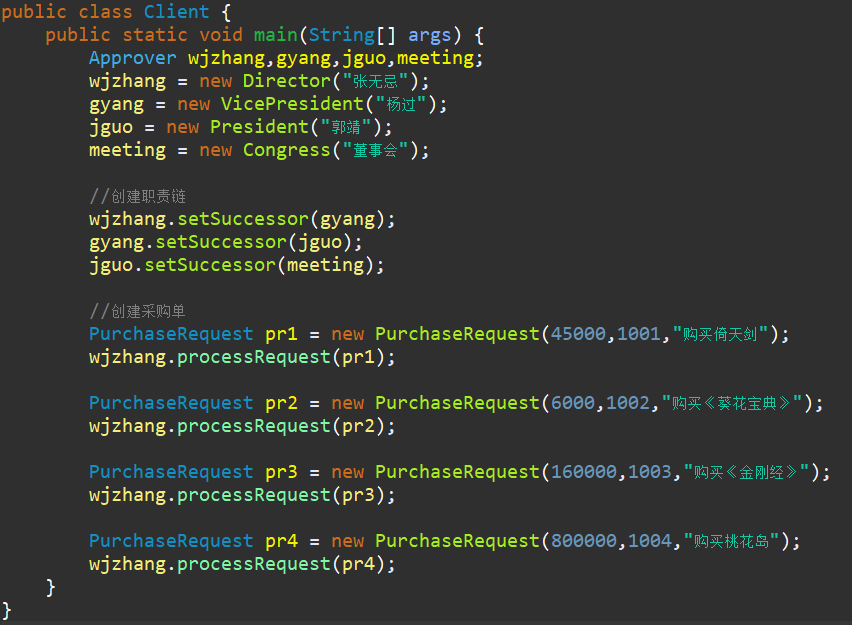
1. President：董事长类，充当具体处理者



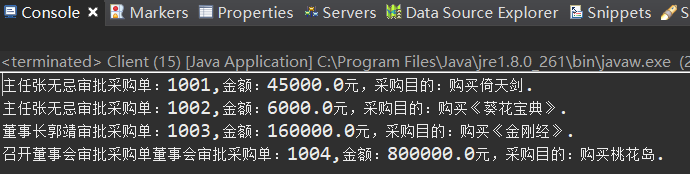
1. Congress：董事会类，充当具体处理者



1. Client：客户端测试类



编译并运行程序，输出结果如下：



# 优缺点及适用环境

职责链模式通过建立一条链来组织请求的处理者，请求将沿着链进行传递，请求发送者无须知道请求在何时、何处以及如何被处理，实现了请求发送者与处理者的解耦。在软件开发中，如果遇到有多个对象可以处理同一请求可以应用职责链模式，例如在Web应用开发中创建多个过滤器（Filter）链来对请求数据进行过滤，在工作流系统中实现公文的分级审批等，使用职责链模式可以较好地解决此类问题。Java语言中的异常处理（Exception Handlers）机制也是职责链模式的典型应用之一，不同的catch子句可以处理不同类型的异常，这些catch子句构成了一条处理异常对象的职责链。

## 优点

职责链模式的优点如下：

1. 职责链模式使得一个对象无须知道是其他哪一个对象处理其请求，对象仅需知道该请求会被处理即可，接收者和发送者都没有对方的明确消息，并且链中的对象不需要知道链的结构，由客户端负责职责链的创建，降低了系统的耦合度。
2. 请求处理对象仅需要维持一个指向其后继者的引用，而不需要维持它对所有的候选处理者的引用，可简化对象之间的相互连接。
3. 在给对象分派职责时，职责链可以带来更多的灵活性，可以通过在运行时对该链进行动态的增加或修改来增加或改变处理一个请求的职责。
4. 在系统中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，从这一点来看是符合开闭原则的。

## 缺点

职责链模式的缺点主要如下：

1. 由于一个请求没有明确的接受者，那么就不能保证它一定会被处理，该请求可能一直到链的末端都得不到处理；一个请求也可能因职责链没有被正确配置而得不到处理。
2. 对于比较长的职责链，请求的处理可能涉及多个处理对象，系统性能将受到一定的影响，而且在进行代码调试时不太方便。
3. 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。

## 适用环境

在以下情况下可以考虑适用职责链模式：

1. 有多个对象可以处理同一请求，具体哪个对象处理该请求待运行时刻再确定，客户端只需将请求提交到链上，而无须关心请求的处理对象是谁以及它是如何处理的。
2. 在不明确指定接受者的情况下向多个对象中的一个提交一个请求。
3. 可动态指定一组对象处理请求，客户端可以动态创建职责链来处理请求，还可以改变链中处理者之间的先后次序。