**一、引言**

　　在现实生活中，有很多请求并不是一个人说了就算的，例如面试时的工资，低于1万的薪水可能技术经理就可以决定了，但是1万~1万5的薪水可能技术经理就没这个权利批准，可能就需要请求技术总监的批准，所以在面试的完后，经常会有面试官说，你这个薪水我这边觉得你这技术可以拿这个薪水的，但是还需要技术总监的批准等的话。这个例子也就诠释了本文要介绍的内容。生活中的这个例子真是应用了责任链模式。

**二、责任链模式介绍**

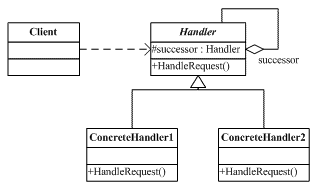
**2.1 责任链模式的定义**

　　从生活中的例子可以发现，某个请求可能需要几个人的审批，即使技术经理审批完了，还需要上一级的审批。这样的例子，还有公司中的请假，少于3天的，直属Leader就可以批准，3天到7天之内就需要项目经理批准，多余7天的就需要技术总监的批准了。介绍了这么多生活中责任链模式的例子的，下面具体给出面向对象中责任链模式的定义。

　　责任链模式指的是——某个请求需要多个对象进行处理，从而避免请求的发送者和接收之间的耦合关系。将这些对象连成一条链子，并沿着这条链子传递该请求，直到有对象处理它为止。

**2.2 责任链模式的结构图**

　　从责任链模式的定义可以发现，责任链模式涉及的对象只有处理者角色，但由于有多个处理者，它们具有共同的处理请求的方法，所以这里抽象出一个抽象处理者角色进行代码复用。这样分析下来，责任链模式的结构图也就不言而喻了，具体结构图如下所示。



　　主要涉及两个角色：

* 抽象处理者角色（Handler）：定义出一个处理请求的接口。这个接口通常由接口或抽象类来实现。
* 具体处理者角色（ConcreteHandler）：具体处理者接受到请求后，可以选择将该请求处理掉，或者将请求传给下一个处理者。因此，每个具体处理者需要保存下一个处理者的引用，以便把请求传递下去。

**2.3 责任链模式的实现**

　　有了上面的介绍，下面以公司采购东西为例子来实现责任链模式。公司规定，采购架构总价在1万之内，经理级别的人批准即可，总价大于1万小于2万5的则还需要副总进行批准，总价大于2万5小于10万的需要还需要总经理批准，而大于总价大于10万的则需要组织一个会议进行讨论。对于这样一个需求，最直观的方法就是设计一个方法，参数是采购的总价，然后在这个方法内对价格进行调整判断，然后针对不同的条件交给不同级别的人去处理，这样确实可以解决问题，但这样一来，我们就需要多重if-else语句来进行判断，但当加入一个新的条件范围时，我们又不得不去修改原来设计的方法来再添加一个条件判断，这样的设计显然违背了“开-闭”原则。这时候，可以采用责任链模式来解决这样的问题。具体实现代码如下所示。

[复制代码](javascript:void(0);)

namespace ChainofResponsibility

{

// 采购请求

public class PurchaseRequest

{

// 金额

public double Amount { get; set; }

// 产品名字

public string ProductName { get; set; }

public PurchaseRequest(double amount, string productName)

{

Amount = amount;

ProductName = productName;

}

}

// 审批人,Handler

public abstract class Approver

{

public Approver NextApprover { get; set; }

public string Name { get; set; }

public Approver(string name)

{

this.Name = name;

}

public abstract void ProcessRequest(PurchaseRequest request);

}

// ConcreteHandler

public class Manager : Approver

{

public Manager(string name)

: base(name)

{ }

public override void ProcessRequest(PurchaseRequest request)

{

if (request.Amount < 10000.0)

{

Console.WriteLine("{0}-{1} approved the request of purshing {2}", this, Name, request.ProductName);

}

else if (NextApprover != null)

{

NextApprover.ProcessRequest(request);

}

}

}

// ConcreteHandler,副总

public class VicePresident : Approver

{

public VicePresident(string name)

: base(name)

{

}

public override void ProcessRequest(PurchaseRequest request)

{

if (request.Amount < 25000.0)

{

Console.WriteLine("{0}-{1} approved the request of purshing {2}", this, Name, request.ProductName);

}

else if (NextApprover != null)

{

NextApprover.ProcessRequest(request);

}

}

}

// ConcreteHandler，总经理

public class President :Approver

{

public President(string name)

: base(name)

{ }

public override void ProcessRequest(PurchaseRequest request)

{

if (request.Amount < 100000.0)

{

Console.WriteLine("{0}-{1} approved the request of purshing {2}", this, Name, request.ProductName);

}

else

{

Console.WriteLine("Request需要组织一个会议讨论");

}

}

}

class Program

{

static void Main(string[] args)

{

PurchaseRequest requestTelphone = new PurchaseRequest(4000.0, "Telphone");

PurchaseRequest requestSoftware = new PurchaseRequest(10000.0, "Visual Studio");

PurchaseRequest requestComputers = new PurchaseRequest(40000.0, "Computers");

Approver manager = new Manager("LearningHard");

Approver Vp = new VicePresident("Tony");

Approver Pre = new President("BossTom");

// 设置责任链

manager.NextApprover = Vp;

Vp.NextApprover = Pre;

// 处理请求

manager.ProcessRequest(requestTelphone);

manager.ProcessRequest(requestSoftware);

manager.ProcessRequest(requestComputers);

Console.ReadLine();

}

}

}

[复制代码](javascript:void(0);)

　　既然，原来的设计会因为价格条件范围的变化而导致不利于扩展，根据“封装变化”的原则，此时我们想的自然是能不能把价格范围细化到不同的类中呢？因为每个价格范围都决定某个批准者，这里就联想到创建多个批准类，这样每个类中只需要针对他自己这个范围的价格判断。这样也就是责任链的最后实现方式了，具体的运行结果如下图所示。



**三、责任链模式的适用场景**

　　在以下场景中可以考虑使用责任链模式：

* 一个系统的审批需要多个对象才能完成处理的情况下，例如请假系统等。
* 代码中存在多个if-else语句的情况下，此时可以考虑使用责任链模式来对代码进行重构。

**四、责任链模式的优缺点**

　　责任链模式的优点不言而喻，主要有以下点：

* 降低了请求的发送者和接收者之间的耦合。
* 把多个条件判定分散到各个处理类中，使得代码更加清晰，责任更加明确。

　　责任链模式也具有一定的缺点，如：

* 在找到正确的处理对象之前，所有的条件判定都要执行一遍，当责任链过长时，可能会引起性能的问题
* 可能导致某个请求不被处理。

**五、总结**

　　责任链降低了请求端和接收端之间的耦合，使多个对象都有机会处理某个请求。如考试中作弊传纸条，泡妞传情书一般。在下一章将继续分享访问者模式。