【推卸责任】

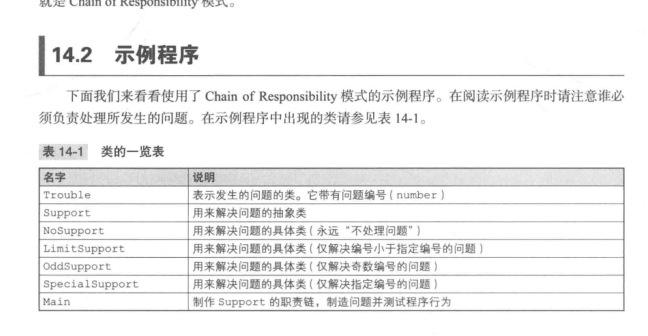
|14.1 Chain of Responsibility模式我们首先看看什么是推卸责任。假设现在我们要去公司领取资料。首先我们向公司前台打听要去哪里领取资料，她告诉我们应该去“营业窗口”。然后等我们到了“营业窗口”后，又被告知应该去“售后部门”。等我们好不容易赶到了“售后部门”，又被告知应该去“资料中心”，因此最后我们又不得不赶往“资料中心”。像这样，在找到合适的办事人之前，我们被不断地踢给一个又一个人，这就是“推卸责任”。

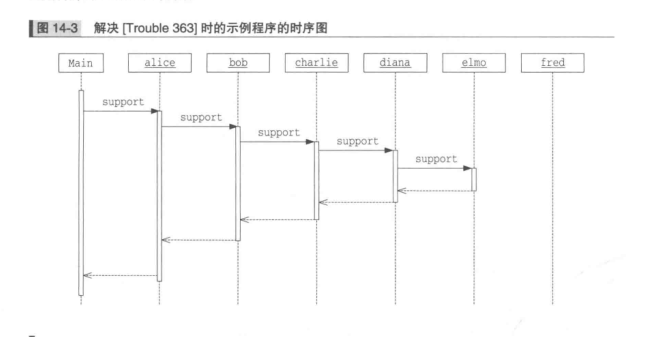
“推卸责任”听起来有些贬义的意思，但是有时候也确实存在需要“推卸责任”的情况。例如，当外部请求程序进行某个处理，但程序暂时无法直接决定由哪个对象负责处理时，就需要推卸责任。这种情况下，我们可以考虑将多个对象组成一条职责链，然后按照它们在职责链上的顺序一个一个地找出到底应该谁来负责处理。

这种模式被称为Chain of Responsibility模式。Responsibility有“责任”的意思，在汉语中，该模式称为“职责链”。总之，我们可以将它想象为推卸责任的结构，这有利于大家记住这种模式。

使用Chain of Responsibility模式可以弱化“请求方”和“处理方”之间的关联关系，让双方各自都成为可独立复用的组件。此外，程序还可以应对其他需求，如根据情况不同，负责处理的对象也会发生变化的这种需求。

当一个人被要求做什么事情时，如果他可以做就自己做，如果不能做就将“要求”转给另外一个人。下一个人如果可以自己处理，就自己做；如果也不能自己处理，就再转给另外一个人……这就是Chain of Responsibility模式。





|14.3 Chain of Responsibility模式中的登场角色在Chain of Responsibility模式中有以下登场角色。

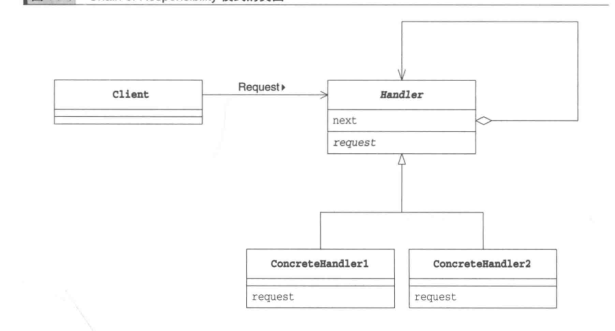
◆Handler（处理者）Handler角色定义了处理请求的接口（API）。Handler角色知道“下一个处理者”是谁，如果自己无法处理请求，它会将请求转给“下一个处理者”。当然，“下一个处理者”也是Handler角色。

在示例程序中，由Support类扮演此角色。负责处理请求的是support方法。

·ConcreteHandler（具体的处理者）ConcreteVisitor角色是处理请求的具体角色。在示例程序中，由NoSupport、LimitSupport、oddsupport、SpecialSupport等各个类扮演此角色。

·Client（请求者）Client角色是向第一个ConcreteHandler角色发送请求的角色。在示例程序中，由Main类扮演此角色。

Chain of Responsibility模式的类图如图14-4所示。



|14.4拓展思路的要点

弱化了发出请求的人和处理请求的人之间的关系

Chain of Responsibility模式的最大优点就在于它弱化了发出请求的人（Client角色）和处理请求的人（ConcreteHandler角色）之间的关系。Client角色向第一个ConcreteHandler角色发出请求，然后请求会在职责链中传播，直到某个ConcreteHandler角色处理该请求。

如果不使用该模式，就必须有某个伟大的角色知道“谁应该处理什么请求”，这有点类似中央集权制。而让“发出请求的人”知道“谁应该处理该请求”并不明智，因为如果发出请求的人不得不知道处理请求的人各自的责任分担情况，就会降低其作为可复用的组件的独立性。

补充说明为了简单起见，在示例程序中，我们让粉演Client角色的Main类负责串联起ConcreteHandler的职责链。

可以动态地改变职责链

在示例程序中，问题的解决是按照从Alice到Fred的固定顺序进行处理的。但是，我们还需要考虑负责处理的各个ConcreteHandler角色之间的关系可能会发生变化的情况。如果使用Chain of Responsibility模式，通过委托推卸责任，就可以根据情况变化动态地重组职责链。

如果不使用Chain of Responsibility模式，而是在程序中固定写明“某个请求需要谁处理”这样的对应关系，那么很难在程序运行中去改变请求的处理者。

在视窗系统中，用户有时需要可以自由地在视窗中添加控件（按钮和文本输入框等）。这时，Chain of Responsibility模式就有了用武之地。

|专注于自己的工作

“推卸”这个词虽然有贬义，但是反过来想，这样才可以使每个对象更专注于自己的工作，即每个ConcreteHandler角色都专注于自己所负责的处理。当自己无法处理时，ConcreteHandler角色就会干脆地对下一个处理者说一句“嘿，交给你了”，然后将请求转出去。这样，每个ConcreteHandler角色就能只处理它应该负责的请求了。

如果我们不使用Chain of Responsibility模式又会怎样呢？这时，我们需要编写一个“决定谁应该负责什么样的处理”的方法。亦或是让每个ConcreteHandler角色自己负责“任务分配”工作，即“如果自己不能处理，就转交给那个人。如果他也不能处理，那就根据系统情况将请求再转交给另外一个人”。

|推卸请求会导致处理延迟吗

使用Chain of Responsibility模式可以推卸请求，直至找到合适的处理请求的对象，这样确实提高了程序的灵活性，但是难道不会导致处理延迟吗？

确实如此，与“事先确定哪个对象负责什么样的处理，当接收到请求时，立即让相应的对象去处理请求”相比，使用Chain of Responsibility 模式确实导致处理请求发生了延迟。

不过，这是一个需要权衡的问题。如果请求和处理者之间的关系是确定的，而且需要非常快的处理速度时，不使用Chain of Responsibility模式会更好。

|14.5相关的设计模式

·Composite模式（第11章）Handler角色经常会使用Composite模式。

·Command模式（第23章）有时会使用Command模式向Handler角色发送请求。