# 基本概念

责任链模式（Chain of Responsibility Pattern）是一种行为型设计模式，它用于将请求的发送者和接收者解耦，使得多个对象都有机会处理这个请求。在责任链模式中，有一个请求处理链条，每个处理请求的对象都是一个节点，当请求进入这个链条时，链条上的节点逐一判断是否能够处理该请求，如果可以，则处理；否则，将请求传递给下一个节点，直到请求被处理为止。责任链模式定义了一个请求处理对象的链条，每个对象都可以处理请求或者将请求转发给下一个对象，直到有一个对象处理请求为止。在责任链模式中，请求发送者不需要知道链条中具体的处理对象，只需要将请求发送给链头即可，具体的处理过程和实现细节由链条中的对象来决定。责任链模式可以有效地解耦请求发送者和接收者，分离职责，提高系统的灵活性和可维护性。

# 应用场景

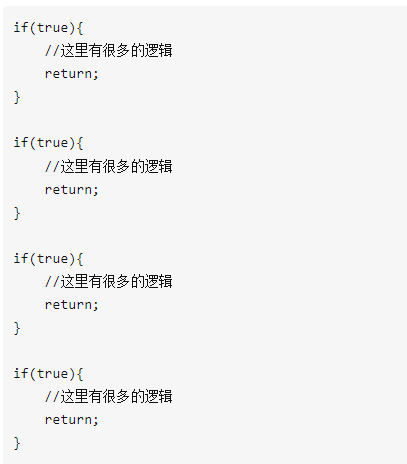
1、过滤器(Filter)：在Servlet中，过滤器就是使用责任链模式实现的。每个过滤器都可以决定是否处理请求，或者将其转发给下一个过滤器进行处理。

2、拦截器(Interceptor)：在Spring框架中，拦截器就是使用责任链模式实现的。拦截器可以对请求进行预处理或后处理，也可以将请求转发给下一个拦截器进行处理。

3、异常处理(Exception Handling)：在Java中，可以使用责任链模式来处理异常。首先，程序先尝试使用自定义的异常处理器来处理异常，如果该处理器无法处理异常，则将其转发给下一个处理器进行处理。

4、日志记录(Logger)：在Java中，可以使用责任链模式来记录日志。每个日志记录器都可以决定是否需要记录该日志，或者将其转发给下一个日志记录器进行记录。

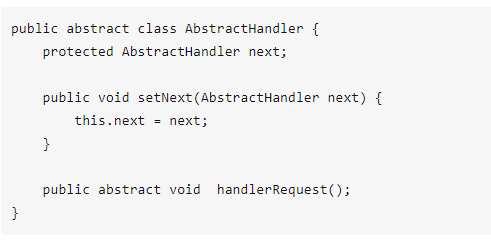
就我个人理解而言，如果一个逻辑是按照一定的步骤进行的，而步骤之间存在复杂的逻辑计算，那么可以考虑使用责任链设计模式。或者说，当你的代码中出现这种情况的时候，你也可以考虑通过责任链设计模式来改进。



# 基本使用

## 1、抽象处理者角色（Handler）

定义出一个处理请求的接口（或者抽象类）。如果需要，接口可以定义出一个方法以设定和返回对下家的引用。这个角色通常由一个 Java 抽象类或者 Java 接口实现。下图中 AbstractHandler 类的聚合关系给出了具体子类对下家的引用，抽象方法 handleRequest() 规范了子类处理请求的操作。



## 2、具体处理者角色（ConcreteHandler）

具体处理者接到请求后，可以选择将请求处理掉，或者将请求传给下一个处理者。

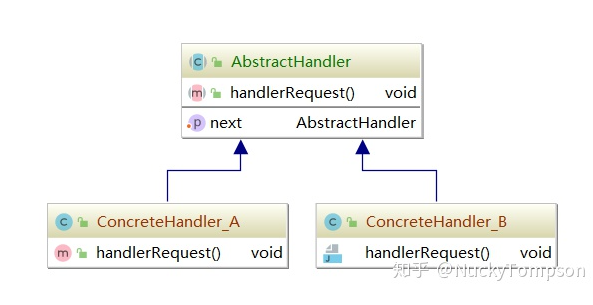


## 3、责任链客户端

责任链客户端设置处理者链，并且返回第一个处理者：



# 责任链模式的类结构图



在上述类结构图中，最上层是一个抽象类，抽象类持有自己的引用，其实是用来接收下一个处理者的。当然，大家也可以在抽象类的上层定义一个接口，这样扩展性在一定场景下会更优。

