Chain Of Responsibility

Design Pattern

Branga Andrei - 10 June 2018

Intent

Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

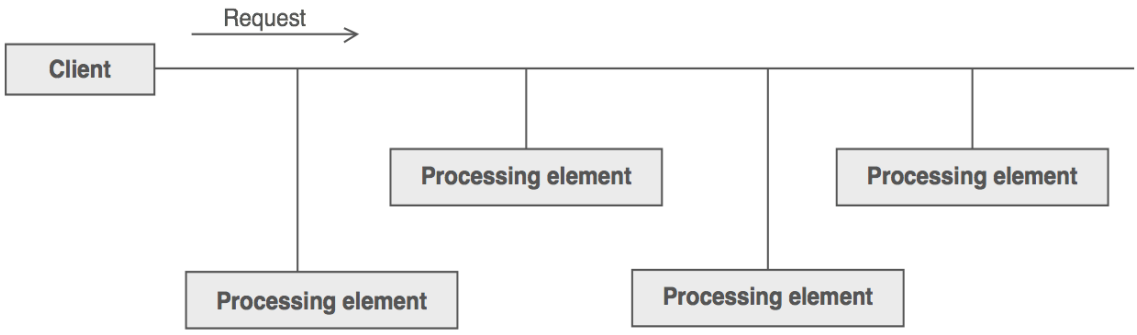
Launch-and-leave requests with a single processing pipeline that contains many possible handlers.

An object-oriented linked list with recursive traversal.

Explanation

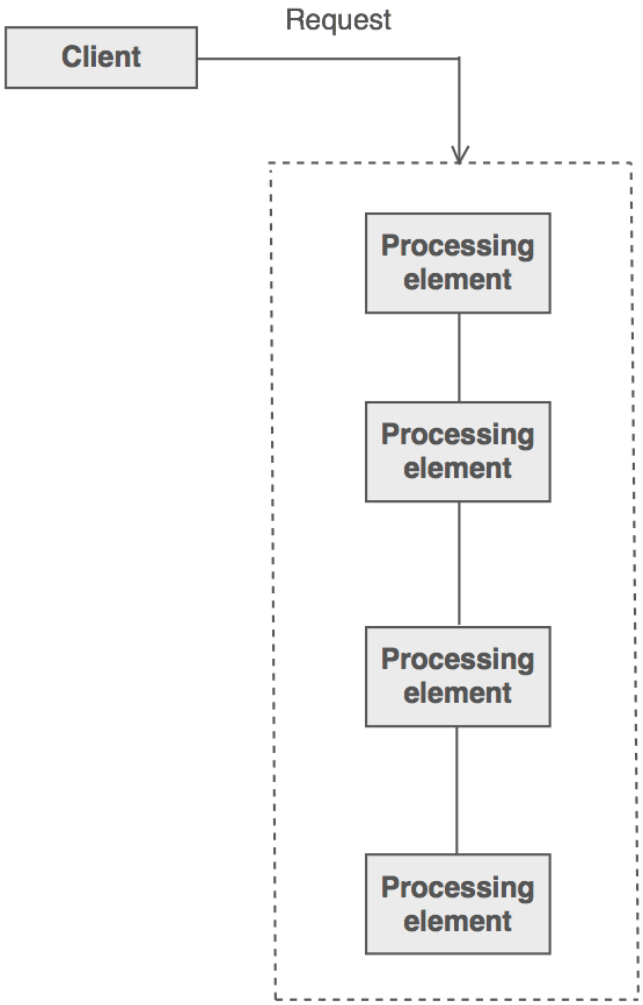
Problem

There is a potentially variable number of "handler" or "processing element" or "node" objects, and a stream of requests that must be handled. Need to efficiently process the requests without hard-wiring handler relationships and precedence, or request-to-handler mappings.



Solution discussion

Encapsulate the processing elements inside a "pipeline" abstraction; and have clients "launch and leave" their requests at the entrance to the pipeline.



The pattern chains the receiving objects together, and then passes any request messages from object to object until it reaches an object capable of handling the message. The number and type of handler objects isn't known a priori, they can be configured dynamically. The chaining mechanism uses recursive composition to allow an unlimited number of handlers to be linked.

Chain of Responsibility simplifies object interconnections. Instead of senders and receivers maintaining references to all candidate receivers, each sender keeps a single reference to the head of the chain, and each receiver keeps a single reference to its immediate successor in the chain.
Make sure there exists a "safety net" to "catch" any requests which go unhandled.
Do not use Chain of Responsibility when each request is only handled by one handler, or, when the client object knows which service object should handle the request.

https://sourcemaking.com/design_patterns/chain_of_responsibility
https://www.tutorialspoint.com/design_pattern/chain_of_responsibility_pattern.htm

Related patterns

- Chain of Responsibility, Command, Mediator, and Observer, address how you can decouple senders and receivers, but with different trade-offs. Chain of Responsibility passes a sender request along a chain of potential receivers.
- Chain of Responsibility can use Command to represent requests as objects.
- Chain of Responsibility is often applied in conjunction with Composite. There, a component's parent can act as its successor.

Common situations of use

55

The fact that you can break the chain at any point differentiates the Chain of Responsibility pattern from the Decorator pattern. Decorators can be thought of as executing all at once without any interaction with the other decorators. Links in a chain can be thought of as executing one at a time, because they each depend on the previous link.

Use the Chain of Responsibility pattern when you can conceptualize your program as a chain made up of links, where each link can either handle a request or pass it up the chain.

When I used to work with the Win32 API, I would sometimes need to use the hooking functionality it provides. Hooking a Windows message roughly follows the Chain of Responsibility pattern. When you hooked a message such as WM_MOUSEMOVE, your callback function would be called. Think of the callback function as the last link in the chain. Each link in the chain can decide whether to throw away the WM_MOUSEMOVE message or pass it up the chain to the next link.

If the Decorator pattern had been used in that example, you would have been notified of the WM_MOUSEMOVE message, but you would be powerless to prevent other hooks from handling it as well.

Another place the Chain of Command pattern is used is in game engines. Again, you can hook engine functions, events, and other things. In the case of a game engine, you don't want to simply add functionality. You want to add functionality and prevent the game engine from performing its default action.

share improve this answer

answered Apr 14 '09 at 14:54

William Brendel

25.1k ● 13 ● 65 ● 76

https://stackoverflow.com/questions/747913/why-would-i-ever-use-a-chain-of-responsibility-over-a-decorator - StackOverflow

Confusions

Question:

I am quite confused with Decorator and Chain of Responsibilities patterns. Per definition, Decorator adds additional responsibilities and Chain of Responsibilities pattern lets classes to share the responsibilities.

Can Decorator be called a Chain of Responsibilities on the same Object? How to really judge and chose between these two design patterns?

Answer:

There is a significant difference between the two patterns, even if they are implemented in much the same fashion.

With a decorator, you always do some work, and then always pass the message on to your "parent" object.
With chain of command, you might do some work. Only in the event that you don't do the work do you pass the message on to your "parent" object.

The decorator is "adding value" as a message goes past.
The Chain of Command, either handles the message, or passes it on to the "boss" to handle it.

https://coderanch.com/t/668194/certification/Difference-Decorator-Chain-Responsibility-patterns - CodeRanch

BELOW IS A LINK TO A FULL IMPLEMENTATION IN JAVA:

http://designpattern.co.il/ChainOfResponsibilty.html - DesignPattern