

Module 13 sample

Understanding and using Chain

13.1 Introducing the Chain component	1
13.2 Using the Chain component.....	2
13.3 Struts and the Chain component	6
13.4 Summary	11
Index	12

Chain of Responsibility is a pattern that allocates the task of handling a request to multiple classes, one after another. Each class attempts to handle the request as best as it can, until a class can handle it in a complete and precise manner. Classes in this chain have no knowledge of the other classes and act in isolation, with only the request parameters available for information and possible modification.

The Chain component from the Commons stable provides an API that you can use to model this Chain of Responsibility pattern in your applications. In this section, we'll take a brief look at this API.

13.1 Introducing the Chain component

Three important concepts in the Chain component define its API: command, chain, and context. We'll discuss these concepts before we introduce some examples.

13.1.1 A unit of work is a command

A *command* is an individual unit of work. Think of a command as a link in a chain of work tasks. However, unlike a link, the command doesn't have any knowledge of the previous and the next commands; nor does it know its position in the chain. The purpose of a command is to do some work and give the result in boolean terms to the calling function.

In Chain, a command is represented by the `Command` interface in the `org.apache.commons.chain` package. This interface defines a single method called `execute` that takes a `context` as a parameter and returns a boolean `true` or `false` to indicate success or failure.

13.1.2 All the commands make a chain

A *chain*, not surprisingly, is a collection of commands in order. Upon receiving a request, the chain starts executing the commands registered with it, in the order they were registered. Each command is executed until a command informs the chain that it has successfully completed the task, at which point processing stops and no further commands are executed. If no command is successful in completing the task, the chain returns with the response of the last command in the chain or an exception.

The `Chain` interface represents a chain in this API. There is a concrete implementation class called `ChainBase` in the `org.apache.commons.chain.impl` package. Chains are also executed using the `executeCommand`, which begins calling the `execute` method of the chain's individual commands until one of them returns `true` or no more commands are left to execute (or an exception is thrown).

13.1.3 Session state is in context

A chain runs in a particular *context*. A context, like a session context in web applications, represents state information. Each command in the chain gets a context, which represents the current state of the chain's variables. Not surprisingly, the context is implemented as a map, thus allowing name-value pairing.

A context is defined by the `Context` interface. An implementation is provided by the `ContextBase` class in the `org.apache.commons.chain.impl` package.

Other Chain concepts

In addition to these three concepts, Chain also introduces the idea of a *catalog*, which is defined by the `Catalog` interface, and which allows Chain and command instances to be identified together using a common key. Also, a *filter* is defined as a special type of command that can perform a post-processing function. The idea is that if a filter special command is part of a chain, and if it's called to perform work (by calling its `execute` method), the chain will also call a `postprocess` method to do some post-processing work after the `execute` method is called—*regardless* of the result of the `execute` method.

Let's see how the Chain component can be used.

13.2 Using the Chain component

Although the Chain component is designed to operate in environments that require complex workflow processing, it isn't tied to a particular implementation. It has been designed to keep its API separate from implementations; it supplies targeted implementations for the Web, Struts, portlets, and JavaServer Faces, but the core of the concept remains in the five interface definitions we talked about earlier. Therefore, in this section, we'll use the main three interfaces to develop a Chain of Responsibility application to illustrate how easy it is to work with this API. Make sure that you download `commons-chain.jar` and have it installed in your `CLASSPATH` before trying these examples.

The application we'll develop is very simple. The idea is to simulate a workflow situation where a `String` value needs to be pre- and post-processed. This processing alters the `String` to add extra characters. For example, if a `String` has the value, "Chain is great", the value is changed to " {Chain is great" by preprocessing and to " {Chain is great} " by post-processing. The process isn't complete without both operations. Listing 13.1 shows a common base class for this operation.

Listing 13.1 Superclass for processing a String

```
package com.manning.commons.chapter13;

import org.apache.commons.chain.Command;
import org.apache.commons.chain.Context;

public abstract class StringProcessorCommand implements Command {
    public abstract boolean execute(Context context);
    public String getMainStringKey() {
        return this.mainStringKey;
    }
    public void setMainStringKey(String mainStringKey) {
```

Implement
Command
interface ←

execute
command is
abstract ←