Chain of Responsibility Pattern

责任链模式

As the name suggests, the chain of responsibility pattern creates a chain of receiver objects for a request. This pattern decouples sender and receiver of a request based on type of request. This pattern comes under behavioral patterns.

In this pattern, normally each receiver contains reference to another receiver. If one object cannot handle the request then it passes the same to the next receiver and so on.

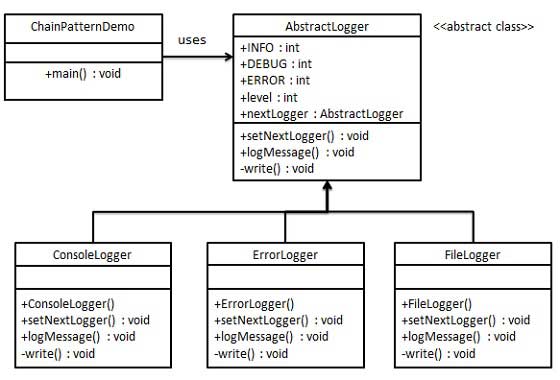
顾名思义，责任链模式对于一个请求会去创建一条接受者链。这种模式会解耦一种类型的请求中接受者和发送者，该模式模式属于行为模式的一种。

在该模式当中，一般情况下，每个接受者都会有另外一个接受者的引用，如果该接受者无法处理该请求，请求会通过引用传入下一个接受者。

Implementation

We have created an abstract class *AbstractLogger* with a level of logging. Then we have created three types of loggers extending the *AbstractLogger*. Each logger checks the level of message to its level and print accordingly otherwise does not print and pass the message to its next logger.

我们创建一个抽象类AbstaractLogger，该类中含有日志的记录等级。然后我们会创建三个logger实现AbstractLogger。每一个logger都会检查并且比较消息的等级与该logger等级，判断打印或者不打印，并且传递消息到下一个logger。



Step 1

Create an abstract logger class.

创建一个抽象的Logger类

*AbstractLogger.java*

public abstract class AbstractLogger {

public static int INFO = 1;

public static int DEBUG = 2;

public static int ERROR = 3;

protected int level;

//next element in chain or responsibility

protected AbstractLogger nextLogger;

public void setNextLogger(AbstractLogger nextLogger){

this.nextLogger = nextLogger;

}

public void logMessage(int level, String message){

if(this.level <= level){

write(message);

}

if(nextLogger !=null){

nextLogger.logMessage(level, message);

}

}

abstract protected void write(String message);

}

Step 2

Create concrete classes extending the logger.

创建具体的类实现AtstractLogger

*ConsoleLogger.java*

public class ConsoleLogger extends AbstractLogger {

public ConsoleLogger(int level){

this.level = level;

}

@Override

protected void write(String message) {

System.out.println("Standard Console::Logger: " + message);

}

}

*ErrorLogger.java*

public class ErrorLogger extends AbstractLogger {

public ErrorLogger(int level){

this.level = level;

}

@Override

protected void write(String message) {

System.out.println("Error Console::Logger: " + message);

}

}

*FileLogger.java*

public class FileLogger extends AbstractLogger {

public FileLogger(int level){

this.level = level;

}

@Override

protected void write(String message) {

System.out.println("File::Logger: " + message);

}

}

Step 3

Create different types of loggers. Assign them error levels and set next logger in each logger. Next logger in each logger represents the part of the chain.

创建不同类型的logger。设置每个Logger的错误等级以及每个logger的Next logger，每一个logger的NextLogger都是当前链的一部分。

*ChainPatternDemo.java*

public class ChainPatternDemo {

private static AbstractLogger getChainOfLoggers(){

AbstractLogger errorLogger = new ErrorLogger(AbstractLogger.ERROR);

AbstractLogger fileLogger = new FileLogger(AbstractLogger.DEBUG);

AbstractLogger consoleLogger = new ConsoleLogger(AbstractLogger.INFO);

errorLogger.setNextLogger(fileLogger);

fileLogger.setNextLogger(consoleLogger);

return errorLogger;

}

public static void main(String[] args) {

AbstractLogger loggerChain = getChainOfLoggers();

loggerChain.logMessage(AbstractLogger.INFO,

"This is an information.");

loggerChain.logMessage(AbstractLogger.DEBUG,

"This is an debug level information.");

loggerChain.logMessage(AbstractLogger.ERROR,

"This is an error information.");

}

}

Step 4

Verify the output.

校验输出

Standard Console::Logger: This is an information.

File::Logger: This is an debug level information.

Standard Console::Logger: This is an debug level information.

Error Console::Logger: This is an error information.

File::Logger: This is an error information.

Standard Console::Logger: This is an error information.