# 09 命令模式、职责链模式

C-3 创建: 张林伟, 最后修改: 张林伟 2018-10-29 16:46

**命令模式**

将一个请求封装为一个对象，从而使我们可用不同的请求对客户进行参数化；对请求排队或者记录请求日志，以及支持可撤销的操作。

命令模式可以对发送者和接收者完全解耦，发送者与接收者之间没有直接引用关系，发送请求的对象只需要知道如何发送请求，而不必知道如何完成请求。这就是命令模式的模式动机。

Demo

接收者

代码块 Java

```java
// Receivable.java
public interface Receivable {
    void action();
}

// Receiver.java
public class Receiver implements Receivable {
    @Override public void action() {
        System.out.println("接受者：收到命令，执行操作！");
    }
}
```

命令

代码块 Java

```java
// Command.java
public interface Command {
    void execute();
}

// ConcreteCommand.java
public class ConcreteCommand implements Command {
    private Receivable receiver;
    public ConcreteCommand(Receivable receiver) {
        this.receiver = receiver;
    }
    @Override public void execute() {
        System.out.println("命令开始执行！");
        receiver.action();
    }
}
```

调用者

代码块 Java

```java
// Invoker.java
public interface Invoker {
    void invoke();
}

// ConcreteInvoker.java
public class ConcreteInvoker implements Invoker {
    private Command command;
    public ConcreteInvoker(Command command) {
        this.command = command;
    }
    @Override public void invoke() {
        System.out.println("开始发送命令...");
        command.execute();
```

```
    }
}
```

客户端

代码块 Java

```java
public class Client {

    public static void main(String[] args) {
        Receivable receiver = new Receiver();
        Command command = new ConcreteCommand(receiver);
        Invoker invoker = new ConcreteInvoker(command);

        invoker.invoke();
    }
}
```

**职责链模式**

使多个对象都有机会处理请求，从而避免请求的发送者和接收者之间的耦合关系。将这个对象连成一条链，并沿着这条链传递该请求，直到有一个对象处理它为止。

Demo

请求

代码块 Java

```java
@Data
@AllArgsConstructor
public class Request {
    private Integer code;
}
```

处理者父类

代码块 Java

```java
public abstract class Handler {
    @Setter
    public Handler successor;
    public abstract void handleRequest(Request request);
}
```

处理者具体类

代码块 Java

```java
// ConcreteHandler1.java
public class ConcreteHandler1 extends Handler {
    @Override public void handleRequest(Request request) {
        if (request.getCode() < 3) {
            System.out.println("Requset code < 3, ConcreteHandler1 处理请求完成");
        } else {
            System.out.println("ConcreteHandler1 无法处理请求，传递给下一个处理");
            setSuccessor(new ConcreteHandler2());
            successor.handleRequest(request);
        }
    }
}

// ConcreteHandler2.java
public class ConcreteHandler2 extends Handler {
    @Override public void handleRequest(Request request) {
        if (request.getCode() >= 3 && request.getCode() < 8) {
            System.out.println("3 =< Requset code < 8, ConcreteHandler2 处理请求完成");
        } else {
            System.out.println("ConcreteHandler2 无法处理请求，传递给下一个处理");
```

```
            setSuccessor(new ConcreteHandler3());
            successor.handleRequest(request);
        }
    }
}

// ConcreteHandler3.java
public class ConcreteHandler3 extends Handler {
    @Override public void handleRequest(Request request) {
        if (request.getCode() >= 8) {
            System.out.println("Requset code >= 8, ConcreteHandler3 处理请求完成");
        } else {
            System.out.println("ConcreteHandler3 无法处理请求，传递给下一个处理");
            throw new RuntimeException("请求无法处理");
        }
    }
}
```

客户端

代码块 Java

```
public class Client {

    public static void main(String[] args) {
        Handler handler = new ConcreteHandler1();

        Request request1 = new Request(1);
        handler.handleRequest(request1);

        System.out.println("============================");

        Request request2 = new Request(5);
        handler.handleRequest(request2);

        System.out.println("============================");

        Request request3 = new Request(11);
        handler.handleRequest(request3);
    }
}
```