软件体系结构

Zhenyan Ji

Beijing Jiaotong University

行为型设计模式

责任链

责任链

目的：

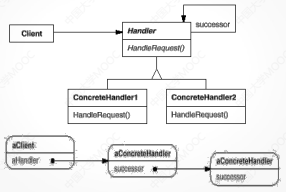
01 通过为多个对象提供处理请求的机会，避免将请求的发送者耦合到其接收者。

02 将接收对象链接并沿链传递请求，直到对象处理它。

责任链

- 链上的每个对象共享一个公共接口，用于处理请求和访问链上的后继者。

责任链模式类图



责任链

参与者

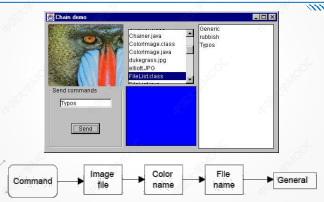
- **Handler**
  - 定义处理请求的接口
  - （可选）实现后继链接
- **ConcreteHandler**
  - 处理它负责的请求

责任链

- 可以访问其继任者
- 如果ConcreteHandler可以处理请求，它会这样做；否则它会将请求转发给其继任者

- **Client**
  - 启动对链上的ConcreteHandler对象的请求

例子



处理多个请求

注意，在CoR模式的基本结构中，Handler类只有一个方法handleRequest（）。这是一个Java接口：

```
public interaface Handler {
    public void handleRequest();
}
```

如果想处理不同类型的请求，例如帮助、打印和格式化请求，怎么办？

解决方案 1

解决方案 1:修改Handler接口以支持多种请求类型：

```
public interface Handler {
    public void handleHelp();
    public void handlePrint();
    public void handleFormat();
}
```

解决方案 1

现在任何具体的处理程序都必须实现此Handler接口的所有方法。

解决方案 1

实现Handler接口的具体handler:

```
public class ConcreteHandler implements Handler {
  private Handler successor;
  public ConcreteHandler(Handler successor)
  {
          this.successor = successor;     }
  public void handleHelp() {
  // We handle help ourselves, so help code
 is here.   }
```

```
public void handlePrint() {
        successor.handlePrint();        }
    public void handleFormat() {
        successor.handleFormat();}
}
```

解决方案 1

如果添加一种新的请求，需要修改接口，这意味着需要修改所有具体的处理程序！

解决方案 2

解决方案2：为每类请求提供分离的处理程序接口.

```
public interface HelpHandler {
    public void handleHelp(); }
public interface PrintHandler {
    public void handlePrint(); }
public interface FormatHandler {
    public void handleFormat(); }
```

解决方案 2

具体的处理程序可以实现这些接口中的一个（或多个）。具体处理程序必须具有对其处理的每种类型请求的后继引用，以防它需要将请求传递给其后继者。

解决方案 2

这是处理三种请求类型的具体处理程序.

```
public class ConcreteHandler implements
    HelpHandler, PrintHandler, FormatHandler {
    private HelpHandler helpSuccessor;
    private PrintHandler printSuccessor;
    private FormatHandler formatSuccessor;
```

```
public ConcreteHandler(HelpHandler
  helpSuccessor, PrintHandler printSuccessor,
  FormatHandler formatSuccessor) {
    this.helpSuccessor = helpSuccessor;
    this.printSuccessor = printSuccessor;
    this.formatSuccessor = formatSuccessor;
        }
public void handleHelp() {  //handle help,  so help
code is here.   }
public void handlePrint()
{printSuccessor.handlePrint();}
public void handleFormat()
{formatSuccessor.handleFormat();}
}
```

解决方案 3

解决方案3：在Handler接口中有个方法，它接受描述请求类型的参数.

```
public interface Handler {
    public void handleRequest(String request);
}
```

解决方案 3

具体handler 看起来像:

```
public class ConcreteHandler implements Handler {
    private Handler successor;
    public ConcreteHandler(Handler successor) {
            this.successor = successor;             }
    public void handleRequest(String request) {
            if (request.equals("Help")) {
                    // help code is here.
            } else     // Pass it on!
                    successor.handle(request);    }  }
```

责任链

适用性

- 当多个对象可以处理请求且事先不知道实际处理对象时
- 当请求遵循“处理或转发”模型 - 也就是说，某些请求可在生成的地方处理，而其他请求必须转发到另一个对象处理
- 当希望能够动态修改可处理请求的对象集时。
- 此方法为在对象间分配职责提供了额外的灵活性。
- 可能没有任何对象可处理请求，然而，链中的最后一个对象可能只是丢弃无法处理的请求。

责任链

- 优点：链可以动态地修改
- 缺点：必须分别在每个模块中实现链接，发送和转发代码。