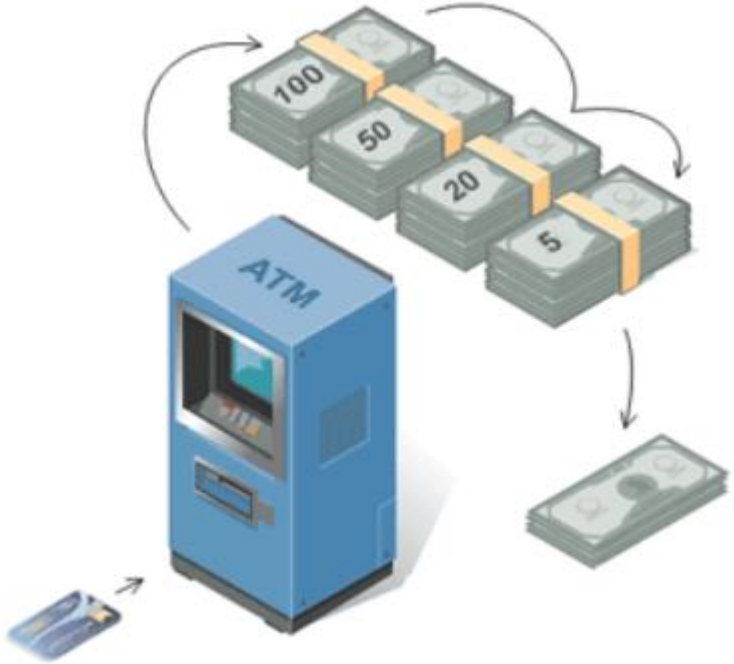
بررسی الگوهای طراحی–
Chain Of Responsibility

توسط : محمد حسین زارع
@zare88r: twitter
telegram
Join Us : @JavaLandCH

مفاهیم

- جدا کننده درخواست دهنده و دریافت کننده
- دریافت کننده یک reference به handler بعدی دارد
- موجب loose coupling میشود
- نمونه ها:
  - java.util.logging.Logger#log()
  - javax.servlet.Filter#doFilter()
  - Spring Security Filter Chain

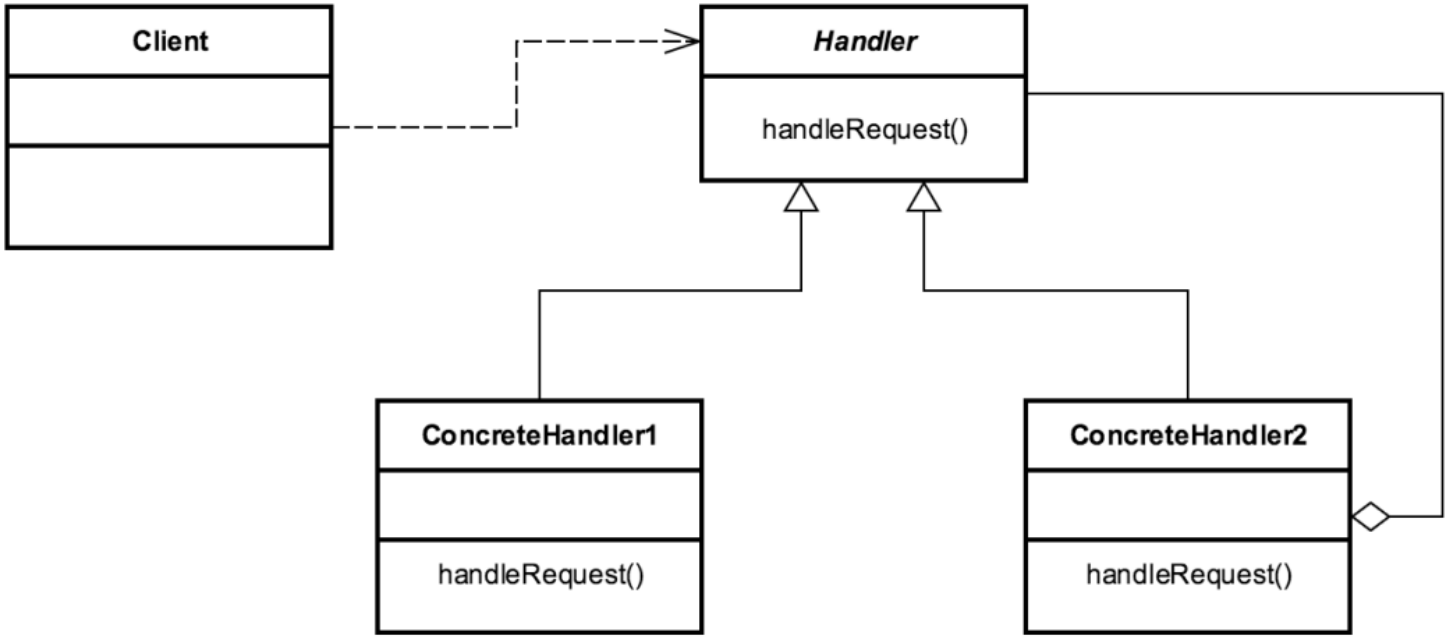
مفاهيم



طراحی

- زنجیره ای از اشیاء پاسخ دهنده
- Handler ها بر مبنای interface هستند
- یک پیاده ساز به ازای هر handler
- هر پیاده ساز به handler بعدی اشاره دارد

طراحی


Client
Handler
handleRequest()
ConcreteHandler1
handleRequest()
ConcreteHandler2
handleRequest()

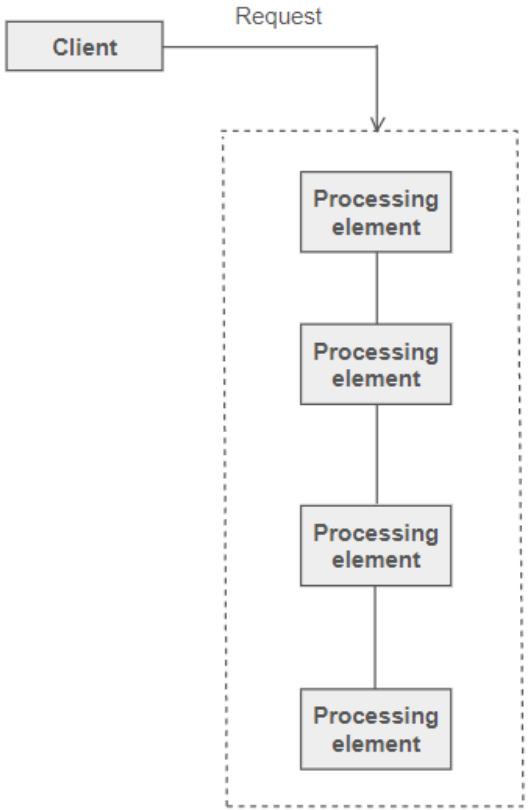
مثال 1

```
Logger logger = Logger.getLogger("Logger1");
logger.setLevel(Level.FINEST);

ConsoleHandler handler = new ConsoleHandler();
handler.setLevel(Level.FINER);
logger.addHandler(handler);

FileHandler fileHandler = new FileHandler("data.xml");
fileHandler.setLevel(Level.FINEST);
logger.addHandler(fileHandler);

logger.finest("Finest Message");
logger.finer("Finer Message");
logger.fine("Fine Message");
```


Client
Request
Processing element
Processing element
Processing element
Processing element

یک نمونه

```
Message message = Message.getBuilder()
        .setValue("Hi , Message Number 2 , contact us with 90909")
        .addDestination(Message.DestinationType.SMS, "09906555455")
        .addDestination(Message.DestinationType.FAX, "0312554554")
        .addDestination(Message.DestinationType.EMAIL,
                "hosein@test.com").createMessage();

MessageService messageService = new MessageService();
messageService.sendMessage(message);
```

یک نمونه

```
public abstract class AbstractMessageHandler {
    protected final AbstractMessageHandler nextHandler;

    public AbstractMessageHandler(AbstractMessageHandler nextHandler) {
        this.nextHandler = nextHandler;
    }

    public abstract void handle(Message message);

}
```

یک نمونه

```
public class EmailMessageHandler extends AbstractMessageHandler {
    public EmailMessageHandler(AbstractMessageHandler nextHandler) {
        super(nextHandler);
    }

    @Override
    public void handle(Message message) {
        if(message.getDestinations().containsKey(Message.DestinationType.EMAIL)){
            String email= message.getDestinations().get(Message.DestinationType.EMAIL);
            String value = message.getValue();
            System.out.printf("Email sent to %s , message : %s",email,value);
            System.out.println();
        }
        if(nextHandler!=null){
            nextHandler.handle(message);
        }
    }
}
```

یک نمونه

```
public class MessageService {
    public void sendMessage(Message message) {
        AbstractMessageHandler nextHandler = null;
        for (Message.DestinationType destinationType
                        : message.getDestinations().keySet()) {
            switch (destinationType) {
                case EMAIL:
                    nextHandler = new EmailMessageHandler(nextHandler);
                    break;
                case SMS:
                    nextHandler = new SMSMessageHandler(nextHandler);
                    break;
                case FAX:
                    nextHandler = new FaxMessageHandler(nextHandler);
                    break;
                default:
                    throw new IllegalStateException("The Handler not supported yet");
            }
        }
        if (nextHandler != null)
            nextHandler.handle(message);
    }
}
```

جمع بندی

- ارتباط sender را از receiver قطع میکند
- در زمان runtime پیکربندی صورت میپذیرد به همین دلیل هم ریسک دارد
- به شکل سلسله مراتبی پیاده سازی میشود
- در صورتی که زنجیره خیلی بزرگ باشد احتمال مشکلات performance دارد