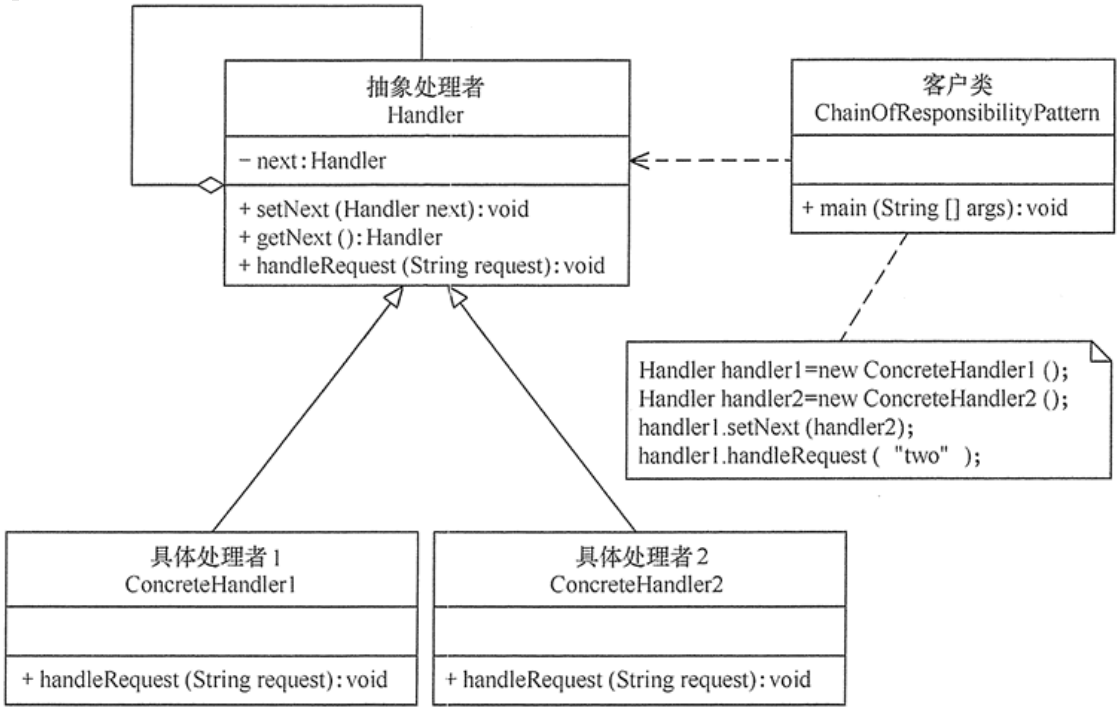
✿ 一、定义

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，并沿着该条链传递该请求，直到有一个对象处理它为止。

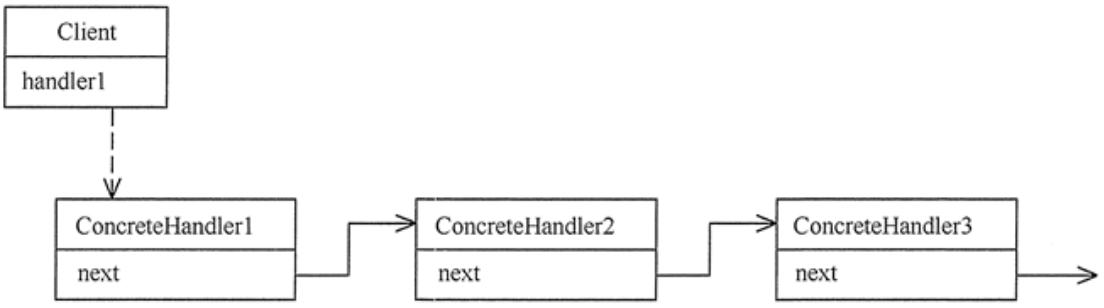
特点：在职责链模式中，客户只需要将请求发送到责任链上即可，无须关心请求的处理细节和请求的传递过程，所以职责链将请求的发送者和请求的处理者解耦了。

✿ 二、结构图

职责链模式主要包含以下角色。
抽象处理者（Handler）角色：定义一个处理请求的接口，包含抽象处理方法和一个后继连接。
具体处理者（Concrete Handler）角色：实现抽象处理者的处理方法，判断能否处理本次请求，如果可以处理请求则处理，否则将该请求转给它的后继者。
客户类（Client）角色：创建处理链，并向链头的具体处理者对象提交请求，它不关心处理细节和请求的传递过程。



职责链结构图



职责链流转图

✿ 三、优缺点

优点：
1.降低对象之间的耦合。使得一个对象无须知道到底是哪一个对象处理其请求以及链的结构，发送者和接收者也无须拥有对方的明确信息。
2.增强了系统的可扩展性。可以根据需要增加新的请求处理类，满足***开闭原则***。
3.增强了给对象指派职责的灵活性。当工作流程发生变化，可以动态地改变链内的成员或者调动它们的次序，也可动态地新增或者删除责任。

4.责任链简化了对象之间的连接。每个对象只需保持一个指向其后继者的引用，不需保持其他所有处理者的引用，这避免了使用众多的 if 或者 if…else 语句。
5.责任分担。每个类只需要处理自己该处理的工作，不该处理的传递给下一个对象完成，明确各类的责任范围，符合类的***单一职责原则***。
缺点：
1.不能保证每个请求一定被处理。由于一个请求没有明确的接收者，所以不能保证它一定会被处理，该请求可能一直传到链的末端都得不到处理。
2.系统性能将受到一定影响。对比较长的职责链，请求的处理可能涉及多个处理对象。
3.增加了客户端的复杂性。职责链建立的合理性要靠客户端来保证，增加了客户端的复杂性，可能会由于职责链的错误设置而导致系统出错，如可能会造成循环调用

✿ 四、使用场景

1.有多个对象可以处理一个请求，哪个对象处理该请求由运行时刻自动确定。
2.可动态指定一组对象处理请求，或添加新的处理者。
3.在不明确指定请求处理者的情况下，向多个处理者中的一个提交请求。
例如：
1.java web filter拦截器的实现，客户端发送的请求会沿着filter拦截器的链式结构传递，进行相应的处理。
2.Java 异常处理机制，可以观察java异常抛出的日志，可以看到异常的抛出是以链式结构层层抛出。

✿ 五、核心code

```
/**
 * 抽象处理者角色
 * @author wangmaogang
 */
public abstract class AbstractHandler {

 private AbstractHandler next;

 public void setNext(AbstractHandler next) {
  this.next = next;
  }

  public AbstractHandler getNext() {
  return next;
  }

  /***
  * 处理请求的方法
  * @param request
  */
  public abstract void handleRequest(String request);
}
```

```
/**
 * 具体处理者角色1
 * @author wangmaogang
```

```
 */
public class ConcreteHandler1 extends AbstractHandler {

 private String type = "one";

 @Override
  public void handleRequest(String request) {
  if(type.equals(request)){
  System.out.println("ConcreteHandler1处理该请求成功!");
  }
  else {
  if(this.getNext()!=null){
  this.getNext().handleRequest(request);
  }
  else{
  System.out.println("没有角色处理该请求!");
  }
  }
  }
}
/**
 * 具体处理者角色2
 * @author wangmaogang
 */
public class ConcreteHandler2 extends AbstractHandler {

 private String type = "two";

 @Override
  public void handleRequest(String request) {
  if(type.equals(request)){
  System.out.println("ConcreteHandler2处理该请求成功!");
  }
  else {
  if(this.getNext()!=null){
  this.getNext().handleRequest(request);
  }
  else{
  System.out.println("没有角色处理该请求!");
  }
```

```
    }
    }
}
```

```
public static void main(String[] args) {
    AbstractHandler handler1 = new ConcreteHandler1();
    AbstractHandler handler2 = new ConcreteHandler2();
    handler1.setNext(handler2);
    handler1.handleRequest("one");
    handler1.handleRequest("two");
    handler1.handleRequest("three");
}

//输出
//ConcreteHandler1处理该请求成功!
//ConcreteHandler2处理该请求成功!
//没有角色处理该请求!
```

✿ 六、扩展

职责链模式存在以下两种情况。

纯的职责链模式：一个请求必须被某一个处理者对象所接收，且一个具体处理者对某个请求的处理只能采用以下两种行为之一：自己处理（承担责任）；把责任推给下家处理。

不纯的职责链模式：允许出现某一个具体处理者对象在承担了请求的一部分责任后又将剩余的责任传给下家的情况，且一个请求可以最终不被任何接收端对象所接收。