853769620

chain of responsibility模式

场景：

（1）现在在某一个地方的业务流程，要执行功能1、功能2、功能3

（2）现在在另外一个地方的业务流程，是要先执行功能3，然后执行功能1和功能2

public **abstract class Handler** {

   protected Handler successor;

【后继者】

   public void **setSuccessor**(Handler successor) {

       this.successor = successor;

  }

   public abstract void handleRequest();

}

public class **ConcreteHandler1 extends Handler** {

   public void handleRequest() {

       System.out.println("ConcreteHandler1实现代码")

【让后继者执行其他逻辑】

       this.successor.handleRequest();

  }

}

public class **ConcreteHandler2 extends Handler** {

   public void handleRequest() {

       System.out.println("ConcreteHandler2实现代码")

       this.successor.handleRequest();

  }

}

public class **Client** {

   public static void main(String[] args) {

       Handler handler1 = new ConcreteHandler1();

       Handler handler2 = new ConcreteHandler2();

       handler1.setSuccessoro(handler2);

       handler1.handleRequest();

  }

}

责任链模式，那也是经常会使用到的，对于那种复杂的业务流程，一种方案是采取面向对象的设计，将不同的业务逻辑封装到不同的类里去，然后通通过方法中的调用来控制业务流向；另外一种就是采用责任链模式，将业务逻辑封装到不同的handler里面去，形成链条式调用。

什么时候用责任链模式呢？很简单，就是如果**你需要动态组装一个业务流程出来，流程中的每个handler业务逻辑需要复用，那就用责任链模式**。

这块在电商系统设计的时候，会结合业务场景来实践