mediator模式：853769620

场景【耦合严重】

（1）三个模块

（2）模块1调用模块2和模块3中功能；模块2要调用模块1和模块3；模块3要调用模块1和模块2

public **interface Mediator** {

   void changed(Colleague colleague);

}

public class **ConcreteMediator implements Mediator** {

   【持有全部模块，统一调度】

   private ColleagueA colleagueA;

   private ColleagueB colleagueB;

   public void changed(Colleague colleague) {

       【不同的Colleague子类执行不同的方法】

  }

}

【模块也要持有中介者】

public **abstract class Colleague** {

   private Mediator mediator;

   public Colleague(Mediator mediator) {

       this.mediator = mediator;

  }

   public Mediator getMediator() {

       return mediator;

  }

}

public class **ColleagueA extends Colleague** {

   public ColleagueA(Mediator mediator) {

       super(mediator);

  }

public void operate() {

【调用中介者】

       getMediator().changed(this);

  }

}

public class **ColleagueB extends Colleague** {

   public ColleagueA(Mediator mediator) {

       super(mediator);

  }

   public void operate() {

       getMediator().changed(this);

  }

}

这个模式，担心的就是系统中各个子系统之前互相之间调用，乱成一团。所以就将系统之间互相调用的逻辑给放到一个所谓的中介者里面去。每个系统如果要通知别的系统干个什么事儿，直接就是调用中介者，中介者负责去调用别的系统。

我们来思考一下这个本质，其实说白了，这个模式就是要让各个系统之间彻底解耦，不要互相强耦合在一起，互相调用过多，调用关系过于混乱。互相调用的时候通过一个中间的组件来解耦。

在实际的企业开发中，不是这么玩儿的，很少有见到说封装一个所谓的中介者，去让各个模块之间解耦，思考这个模式的本质，让各个模块之间解耦合

最最常见的一个方式，就是系统与系统之间，不是走直接的接口调用，而是基于MQ来解耦。录入过模块A要调用模块B和模块C，模块A发送一条消息到MQ里面去，模块B和模块C去消费这条消息，读到消息之后，知道模块A要调用自己，所以就执行对应的逻辑即可。常见于系统与系统之间的调用，可以基于MQ消息，异步执行的方式来调用，不需要同步调用和执行。

大家想一下，这不就是典型的各个子系统用MQ来解耦么，有些操作，互相直接不是直接调用，消息发送到MQ，通过MQ来解耦。我们在电商系统里也完全有这种场景，到后面我们设计系统的时候给大家分析。我们可以封装一个基于内存队列的异步解耦中介者组件，然后让系统间有些可以异步的操作通过中介者来执行，降低互相调用的复杂度。

模块A将消息发送到一个内存队列中去，其他的模块去内存队列中消费自己感兴趣的消息，来执行对应的操作，用队列替代了中介者，让各个模块之间解耦合，rabbitmq去做