853769620

memento模式

场景

（1）类，计算出来一份中间数据

（2）这个类，之后需要基于这份中间数据再执行另外两次操作，每次操作都要基于这份中间数据去执行

（3）第一次基于中间数据操作过后，会对中间数据做出修改

（4）要求是第二次再执行操作的时候，需要**基于未修改之前的中间数据**来执行

【备忘录接口】

public **interface Memento** {

}

public class **Originator** {

   【中间数据】

   private String state;

   public void prepare() {

       this.state = "中间数据";

  }

   public void executeA() {

       System.out.println("基于中间数据【" + state +"】进行了A方法的执行");

       state += "，A方法的结果数据";

  }

   public void executeB() {

       System.out.println("基于中间数据【" + state +"】进行了A方法的执行");

       state += "，A方法的结果数据";

  }

【创建备忘录—存储 state】

   public Memento createMemento() {

       return new MementoImpl(state);

   }

【读取备忘录数据】

public void setMemento(Memento memento) {

       MementoImpl mementoImpl = (MementoImpl)memento;

       this.state = mementoImpl.getState();

  }

   private static class **MementoImpl implements Memento** {

       private String state;

       public MementoIMpl(String state) {

           this.state = state;

      }

       public String getState() {

           return state;

      }

  }

}

【保存备忘录】

public class **Caretaker** {

   private Memento memento;

   public void saveMemento(Memento memento) {

       this.memento = memento;

  }

   public Memento retriveMemento() {

       return this.memento;

  }

}

【具体使用】

public class **Client** {

   public static void main(String[] args) {

       Originator originator = new Originator();

       // 准备好了中间数据

       originator.prepare();

       // 将中间数据保存到备忘录中去

       Memento memento = originator.createMemento();

       // 将备忘录保存到备忘录管理器中去

       Caretaker caretaker = new Caretaker();

       caretaker.saveMemento(memento);

       // 基于中间数据执行了A方法，但是此时中间数据已经改变了

       originator.executeA();

       // 从备忘录管理器中获取备忘录

       memento = caretaker.retrieveMemento();

       // 将备忘录中保存好的中间数据重新设置到原发器中去，就将中间数据恢复成了之前备忘的状态

       originator.setMemento(memento);

       // 接着再次执行方法B

       originator.executeB();

  }

}

备忘录模式看起来有点复杂，其实不复杂，说白了，就是一个类一开始生产了一些数据，这些数据可能会被改变，但是需要将这些中间数据保存起来，给后面不同的操作使用这些中间数据

所以备忘录，顾名思义，就是用备忘录来保存中间数据，后面随时可以获取备忘录里的数据并且恢复这些数据

复杂业务场景，和真实项目的好处，业务足够复杂，才能支撑我们将各种各样的技术运用到项目实践中，纯真实复杂业务项目的驱动，大量的技术，还是理论，demo练习，学完，达不到架构师的水平，demo

广告系统，N个子系统，某一个子系统跟下游的子系统之间是基于MQ去通信的。但是此时要考虑说MQ如果挂了怎么办，写消息到MQ的子系统，一旦发现MQ挂了，将数据写入磁盘，暂存。后面MQ恢复了，再从磁盘里恢复这个数据。

这个模式其实使用场景很少见，属于冷门模式，**我这边打算是跟命令模式结合起来**，做电商系统里面不同子系统之间进程内通过阻塞队列异步通信的一个场景。因为**既然是阻塞队列，就要考虑到也许队列会填满，此时整个系统可能就会卡住，所以需要做一个防御措施，就是写入阻塞队列的时候如果满了，就将数据通过备忘录模式持久化到磁盘上去**，然后到了后面阻塞队列空闲出来的时候，再通过备忘录管理器，从磁盘上恢复备忘录里的数据回来