链滴

GOF 设计模式小白教程之代理模式

作者：valarchie

原文链接：https://ld246.com/article/1567168825306

来源网站：链滴

代理模式（Proxy）

定义：

由于某些原因需要给某对象提供一个代理以控制对该对象的访问。这时，访问对象不适合或者不能直引用目标对象，代理对象作为访问对象和目标对象之间的中介。

通俗解释：

好比去动车站买动车票，如果家离动车站比较远的话肯定就非常不方便。这时候我们就可以选择离家的动车站代售点买票。这里的代售点就相当于代理类。代售点出票也是先从动车站出，然后再售卖给们。比如Spring中的AOP原理也是代理模式，通过一个代理对象访问真实的对象。例如Hibernate的迟加载也是，load方法查询得到的对象仅仅只是一个代理对象，等到真正想要获取对象属性的时候才行数据库查询。

代码：

售票接口：代理类和被代理类共同实现的接口

```
public interface TicketSeller {
    // 售卖火车票
    void sellTicket();

}
```

火车站售票点，被代理的对象。

```
public class StationSeller implements TicketSeller{

    @Override
    public void sellTicket() {
        System.out.println("卖出一张票！");
    }

}
```

火车票代售点，代理对象。在售票的前后，需要加一些必要操作

```
public class ProxySeller implements TicketSeller{
    // 动车站售票机
    private TicketSeller stationSeller;

    public ProxySeller(TicketSeller stationSeller) {
        this.stationSeller = stationSeller;
    }

    @Override
    public void sellTicket() {

        connectingStationSeller();
        stationSeller.sellTicket();
```

```
        payTheTicket();

    }

    // 代售点出票前 需要先连接火车站售票系统
    private void connectingStationSeller() {
        System.out.println("连接火车站售票点，请求出票！");
    }

    // 出票后 需要将票款扣除手续费的钱支付给火车站
    private void payTheTicket() {
        System.out.println("将火车票钱扣除手续费剩下的钱支付给火车站！");
    }

}
```

测试代理模式

```
public class TestProxy {

    public static void main(String[] args) {

        ProxySeller proxy = new ProxySeller(new StationSeller());

        proxy.sellTicket();

    }

}
```

运行结果

```
连接火车站售票点，请求出票!
卖出一张票!
将火车票钱扣除手续费剩下的钱支付给火车站!
```

解析：

1. 代理模式在客户端与目标对象之间起到一个中介作用和保护目标对象的作用；
2. 代理对象可以很容易的加强展目标对象的功能；
3. 代理模式能将客户端与目标对象分离，在一定程度上降低了系统的耦合度；