# 观察者[核心思想]

当被观察对象（目标对象）的状态发生改变时 ，会直接影响到观察对象的行为。

被观察者 和观察者 一般是 一对多的关系，一个被观察者 对应多个观察者，当一个被观察者的状态发生改变时，被观察者通知观察者， 然后可以在观察者内部 进行业务逻辑的处理。

业务场景,:购票送优惠卷

//传统写法

Public interface buytick{

//购票

Void buytick(User userid);

//送优惠卷

Void donateCoupon(User userid);

}

@AutoWride

Buttick tick;

@RequestMapping(“xx”)

Public void buytitck(User user){

tick.buytick(user.getId());

tick.donateCoupon(user.getId());

}

/\*\* 如果业务改了,购票不仅送优惠卷还送电影票,此时此刻我们应该要去改动业务源码..(不推荐)\*\*/

此时我们应该将功能模块化,

将触发源头作为一个监听的点,在去定义观察者.

这里我们可以将送优惠卷,送电影票的两个功能模块化,在将购票类里面添加需要的模块并且提供方法去执行这两个模块。

@RequestMapping(“xx”)

Public void buytitck(User user){

tick.notice(user.getId());

}

核心思想就是: 封装一个观察接口,被观察对象中添加所有苏姚观察接口的对象,当执行观察的方法时,遍历观察者的集合,执行业务逻辑。好处就是当需要添加新的逻辑时,不需要修改源文件,直接在集合中进行添加,删除。解耦合.

代码案例:

<https://blog.csdn.net/u011130752/article/details/45896645>

<http://blog.51cto.com/1027187712/1743743>

https://www.jianshu.com/p/d55ee6e83d66