链滴

GOF 设计模式小白教程之装饰者模式

作者：valarchie

原文链接：https://ld246.com/article/1567089348177

来源网站：链滴

装饰者模式（Decorator）

定义：

指在不改变现有对象结构的情况下，动态地给该对象增加一些职责（即增加其额外功能）的模式，它于对象结构型模式。

通俗解释：

在很多网络游戏当中，武器都可以进行附魔。比如一把普通的刀。可以附魔火属性，也可以再附魔冰性，还可以再附魔风属性。对于武器的附魔。相当于动态地给武器添加额外的功能。在装饰者模式中被装饰的对象和装饰对象继承的是同一个接口，所以对象被装饰过后还是跟原来的对象接口一致，所就可以不同组合的进行装饰。

代码：

抽象接口剑，拥有劈砍方法

```
public interface Sward {
    // 劈砍
    void chop();

}
```

钢剑实现了剑接口

```
public class IronSward implements Sward{

    @Override
    public void chop() {
        System.out.println("钢刀劈砍！");
    }

}
```

剑装饰器接口，通过实现剑的接口，并持有剑的对象。

```
public abstract class SwardDecorator implements Sward{

    protected Sward sward;

    public SwardDecorator(Sward sward) {
        this.sward = sward;
    }

}
```

具体的剑装饰器：火附魔、冰附魔、风附魔装饰器

```
public class FireSwardDecorator extends SwardDecorator {

    public FireSwardDecorator(Sward sward) {
```

```
        super(sward);
    }

    @Override
    public void chop() {
        sward.chop();
        System.out.print("附带火属性！");
    }

}

public class IceSwardDecorator extends SwardDecorator {

    public IceSwardDecorator(Sward sward) {
        super(sward);
    }

    @Override
    public void chop() {
        sward.chop();
        System.out.print("附带冰属性！");
    }

}


public class WindSwardDecorator extends SwardDecorator {

    public WindSwardDecorator(Sward sward) {
        super(sward);
    }

    @Override
    public void chop() {
        sward.chop();
        System.out.print("附带风属性！");
    }

}
```

测试装饰模式

```
public class TestDecorator {

    public static void main(String[] args) {

        Sward ironSward = new IronSward();

        // 进行火焰附魔
        Sward fire = new FireSwardDecorator(ironSward);
        // 进行冰霜附魔
        Sward ice = new IceSwardDecorator(fire);
        // 进行飓风附魔
        Sward wind = new WindSwardDecorator(ice);
```

```
        wind.chop();
    }
}
```

运行结果:

```
钢刀劈砍!
附带火属性!附带冰属性!附带风属性!
```

解析:

1. 采用装饰模式扩展对象的功能比采用继承方式更加灵活。
2. 可以设计出多个不同的具体装饰类，创造出多个不同行为的组合。