# 02 装饰模式、代理模式

C-3 创建: 张林伟, 最后修改: 张林伟 2018-09-21 10:01

## 装饰模式

动态地给一个对象添加一些额外的职责，就增加功能来说，装饰模式比生成子类更为灵活。

装饰模式是为已有功能动态地添加更多功能的一种方式。

当系统需要新功能的时候，这些**新功能仅用来装饰原有类的核心职责或主要行为，即原有类核心逻辑不变**，而且仅仅为了满足一些特定情况才会执行的特殊行为的需要，**如果往主类中加代码来添加这些新功能，会增加主类的复杂性**，而采用装饰模式，包装所要装饰的对象以执行特定情况的特殊行为。

Demo

接口

代码块 Java

```java
public interface Person {
    void dress();
}
```

接口实现类

代码块 Java

```java
public class PersonImpl implements Person {
    @Override public void dress() {
        System.out.println("persons dress clothes");
    }
}
```

装饰器

代码块 Java

```java
public class Decorator implements Person {
    private Person person;

    public Decorator(Person person) {
        this.person = person;
    }

    @Override public void dress() {
        person.dress();
    }
}
```

装饰器实现类

代码块 Java

```java
// StudentDecorator.java
public class StudentDecorator extends Decorator {

    public StudentDecorator(Person person) {
        super(person);
    }

    @Override
    public void dress() {
        System.out.println("students dress school uniform");
    }
}

// TeacherDecorator.java
public class TeacherDecorator extends Decorator {

```

```
    public TeacherDecorator(Person person) {
        super(person);
    }

    @Override
    public void dress() {
        System.out.println("teachers dress work clothes");
    }
}
```

客户端

代码块 Java

```
public class Client {

    public static void main(String[] args) {

        Person person = new PersonImpl();

        Person student = new StudentDecorator(person);
        student.dress();

        Person teacher = new TeacherDecorator(student);
        teacher.dress();

    }

}
// 结果:
// students dress school uniform
// teachers dress work clothes
```

举例：Spring 的 AOP

# 代理模式

为其他对象提供一种代理以控制对这个对象的访问。

Demo：代理模式

举例：JDK的java.io.*包，输入输出流

# 装饰模式 vs 代理模式

代理模式中，代理类对被代理的对象有**控制**权，决定其执行或者不执行。

装饰模式中，装饰类对代理对象没有控制权，只能为其增加一层装饰，以**加强**被装饰对象的功能，仅此而已。