# 15 装饰者模式

在程序开发中，我们其实一开始都不希望某个类天生非常庞大，包含许多职责。那么我们就可以使用装饰者模式来动态地给某个对象添加一些额外的职责，从而不会影响从这个类中派生的其他对象。

传统的面向对象给对象添加功能通常通过继承方式，不过这种不太灵活，一方面导致父子类存在耦合性，父类改变子类也改变，另一方面继承这种复用成为“白箱复用”，父类内部细节对子类是可见的，破坏了封装性

这里另外一种动态的给对象添加职责的方式成为装饰者decorator模式。不改变对象自身的基础上，在运行时添加职责和功能，更加轻便灵活。例如如果天冷了，就可以多穿外套。

## 模拟面向对象的装饰者模式

作为解释执行语言，JS中给对象动态添加职责其实很简单，虽然会改变对象自身，但是更符合JS语言特色：

```
let obj = {
    name: 'sven',
    address: '深圳市',
};

obj.address = obj.address + '福田区';
```

但是这种装饰者模式在JS中适用的场景并不多，例如如下的飞机类，通过装饰者模式升级Fire能力：

```
let Plane = function(){}

Plane.prototype.fire = function(){
    console.log('发射普通子弹')
}

let MissileDecorator = function(plane){
    this.plane = plane;
}

MissileDecorator.prototype.fire = function(){
    this.plane.fire();
```

```
    console.log('发射导弹')
}

let plane = new Plane();
plane = new MissileDecorator(plane);
plane.fire(); // 输出 发射普通子弹 发射导弹
```

这里导弹类接受plane对象并保存了这个参数，并给fire增加职责，通过链式引用形成一个聚合对象。

这里该plane对象的内部装饰过程对用户来说是透明，被装饰者也不需要感知是否被装饰过