大家好，我是楼仔！

突然想写这个设计模式，是因为刚看了FactoryBean，因为它通过装饰模式，来进一步修饰Bean对象，所以想看看这个模式是怎么使用的。我理解这个设计模式，就是基于原来的对象进行装饰。

# 装饰器模式

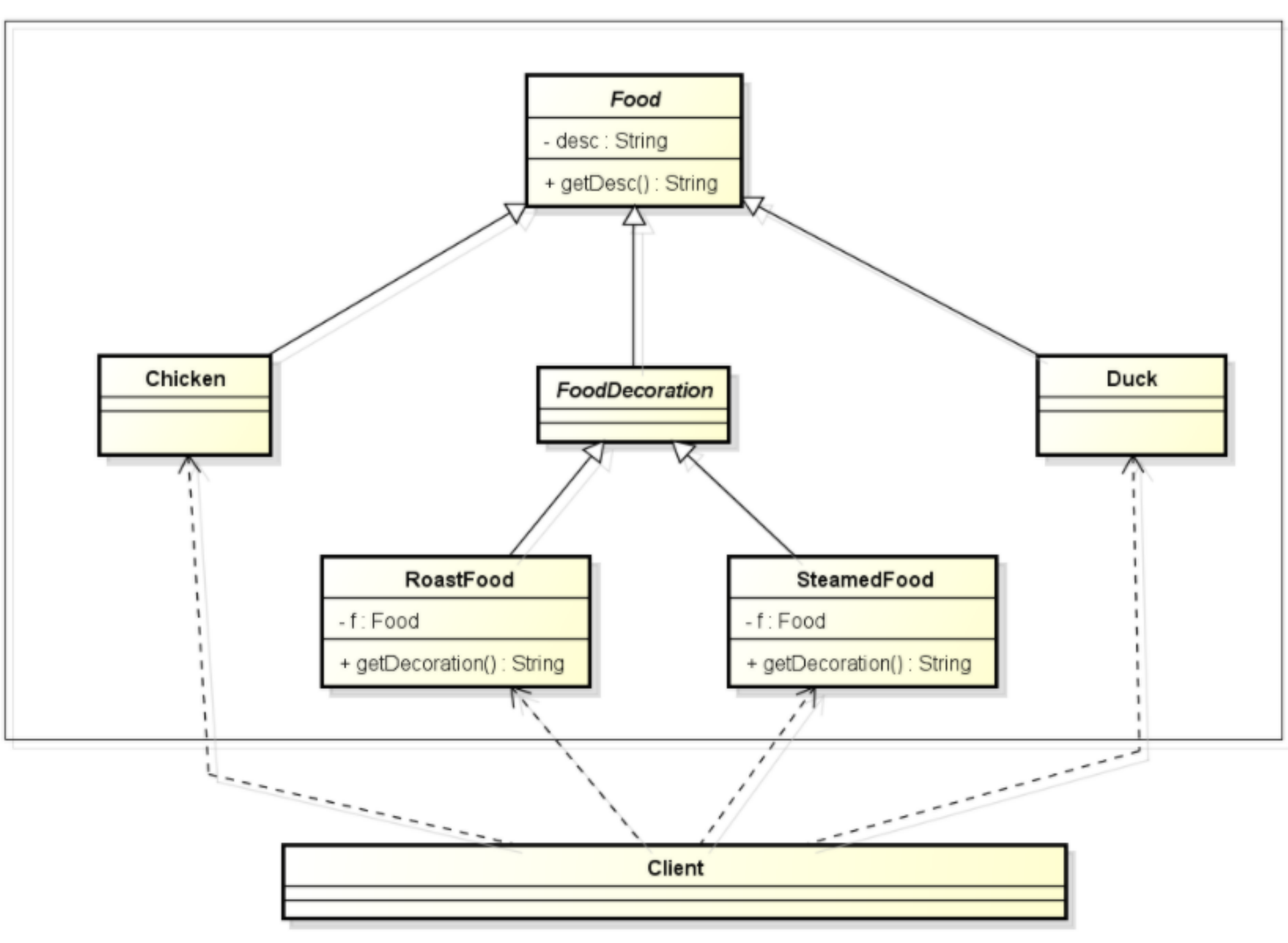
## 基本概念

装饰器模式（Decorator Pattern）允许向一个现有的对象添加新的功能，同时又不改变其结构。这种类型的设计模式属于结构型模式，它是作为现有的类的一个包装。

使用场景

- 在不影响其他对象的情况下，以动态、透明的方式给单个对象添加职责。
- 需要动态地给一个对象增加功能，这些功能也可以动态地被撤销。 当不能采用继承的方式对系统进行扩充或者采用继承不利于系统扩展和维护时。

这种模式创建了一个装饰类，用来包装原有的类，并在保持类方法签名完整性的前提下，提供了额外的功能。



## 装饰器示例

先定义一个Food的抽象类：

```java
public abstract class Food {
    protected String desc;
    public abstract String getDesc();
}
```

然后实例化2个实物对象，比如鸡肉和鸭肉：

```java
public class Chicken extends Food {
    public Chicken() {
        this.desc = "鸡肉";
    }
    @Override
    public String cook() {
        return this.desc;
    }
}
```

```java
public class Duck extends Food {
    public Duck() {
        this.desc = "鸭肉";
    }
    @Override
    public String cook() {
        return this.desc;
    }
}
```

再定义一个装饰的抽象类，这个是继承了Food：

```java
public abstract class FoodDecoration extends Food {
    @Override
    public abstract String cook();
}
```

再定义两个不同的装饰类，一个是烤，一个是蒸：

```java
public class RoastFood extends FoodDecoration {
    private Food food;
    public RoastFood(Food f){
        this.food = f;
    }
    @Override
    public String cook() {
        return getDecoration() + food.cook();
    }
    private String getDecoration(){
        return "烤";
    }
}
```

```java
public class SteamedFood extends FoodDecoration{
    private Food food;
    public SteamedFood(Food food) {
        this.food = food;
    }
    private String getDecoration() {
        return "蒸";
    }
    @Override
    public String cook() {
        return this.getDecoration() + food.cook();
    }
}
```

最后我们可以对实物进行装饰：

```java
public static void main(String args[]) {
    // 测试单纯的食物
    Chicken chicken = new Chicken();
    System.out.println(chicken.cook());
    // 测试单重修饰的食物
    RoastFood roastFood = new RoastFood(chicken);
    System.out.println(roastFood.cook());
    // 测试多重修饰的食物
    SteamedFood steamedFood = new SteamedFood(roastFood);
    System.out.println(steamedFood.cook());
}
// 输出：
// 鸡肉
// 烤鸡肉
// 蒸烤鸡肉
```

评价一下，装饰类主要是需要定义一个通用的抽象方法或者接口，供实体对象和装饰对象继承，并通过装饰类修饰该对象。因为通过修饰类对外暴露的cook()方法和原对象一致（肯定是一致的，因为他们重写同一个方法），所以这个装饰类对象就可以当做源对象使用，不同的是对源对象做了一层装饰。

所以我们在“多重修饰”中可以看到，我们把装饰后的对象roastFood，直接当做实物对象使用，完全没有任何问题，虽然它本质上和实物对象不同，但是使用效果确是一致的（这个和我们平时开发一样，将一个对象作为自己的成员类变量，然后对外暴露的方法，其实就是把该成员类变量的方法包一层，道理是一样的）。

所以回过头来，我们再看FactoryBean，其实它应该也有一个成员类变量，我们通过Bean初始化FactoryBean时，就可以通过FactoryBean对这个Bean进行装饰。

# 优缺点

优点

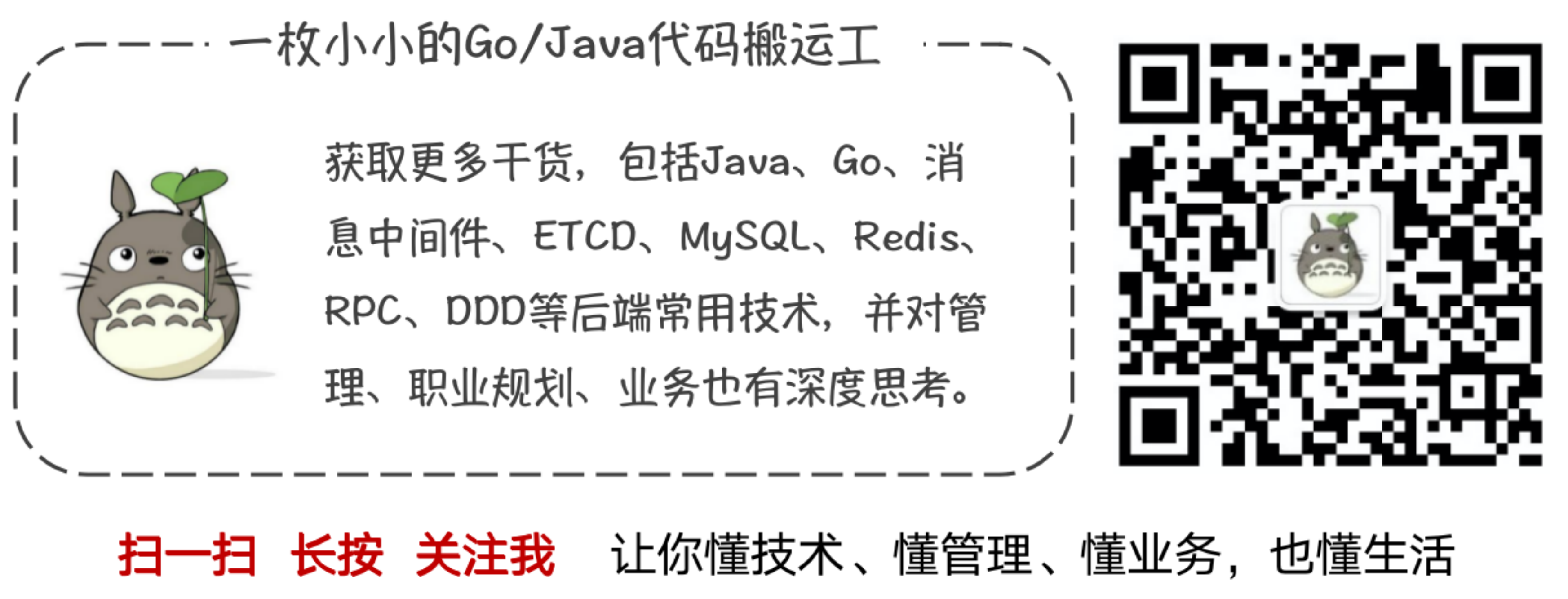
1. 装饰者模式可以提供比继承更多的灵活性
2. 可以通过一种动态的方式来扩展一个对象的功能，在运行时选择不同的装饰器，从而实现不同的行为。
3. 通过使用不同的具体装饰类以及这些装饰类的排列组合，可以创造出很多不同行为的组合。可以使用多个具体装饰类来装饰同一对象，得到功能更为强大的对象。
4. 具体构件类与具体装饰类可以独立变化，用户可以根据需要增加新的具体构件类和具体装饰类，在使用时再对其进行组合，原有代码无须改变，符合“开闭原则”。

缺点

1. 会产生很多的小对象，增加了系统的复杂性
2. 这种比继承更加灵活机动的特性，也同时意味着装饰模式比继承更加易于出错，排错也很困难，对于多次装饰的对象，调试时寻找错误可能需要逐级排查，较为烦琐。

# 学习交流

可以扫下面二维码，关注「楼仔」公众号。



一枚小小的Go/Java代码搬运工

获取更多干货，包括Java、Go、消息中间件、ETCD、MySQL、Redis、RPC、DDD等后端常用技术，并对管理、职业规划、业务也有深度思考。

扫一扫 长按 关注我 让你懂技术、懂管理、懂业务，也懂生活

长按二维码，回复 **「加群」**，欢迎一起学习交流哈~~ 👏🏻👏🏻👏🏻

楼仔

湖北 武汉

扫一扫 长按
加技术群的备注：加群

> 尽信书则不如无书，因个人能力有限，难免有疏漏和错误之处，如发现 bug 或者有更好的建议，欢迎批评指正，不吝感激。