

# 物品锻造 与 **Decorator** 模式

Design Pattern - Decorator

张子阳

[www.tracefact.net](http://www.tracefact.net)

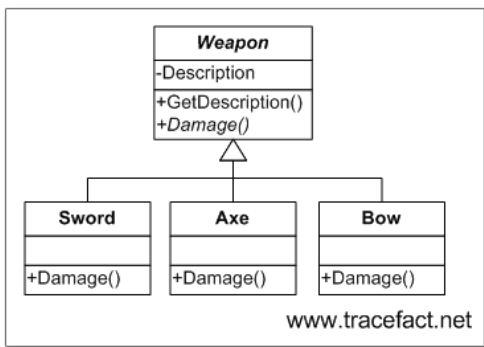
[jimmy\\_dev@163.com](mailto:jimmy_dev@163.com)

# 引言

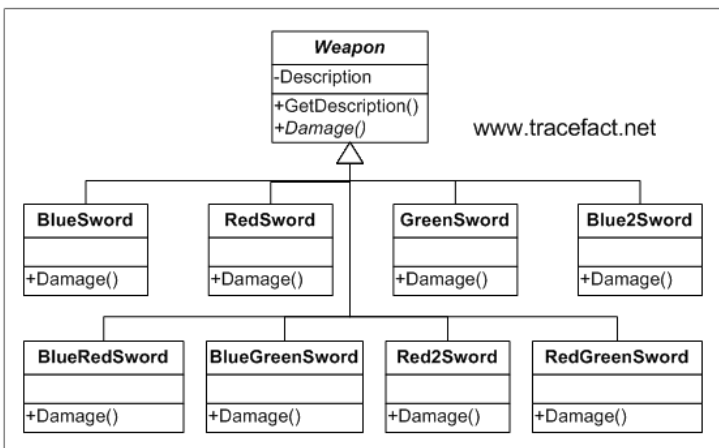
物品锻造是各类奇幻游戏中的常见功能，就拿众所周知的 Diablo 来说吧。假设角色拥有一把单手剑，可能基础攻击力只有 13，但是它有三个装备孔。当给剑镶嵌一颗蓝宝石的时候，它就拥有了额外的冰冻效果并多加 2 点攻击力；当给剑镶嵌一颗红宝石的时候，它又拥有了额外的火焰伤害并多加 3 点攻击力；当给剑镶嵌一颗绿宝石的时候，它又拥有了额外的中毒伤害并多加的 4 点攻击力。当然，也可以三个孔都镶嵌同一色的宝石。本文将说明如何使用 Decorator 模式来完成这样的设计。

## 使用继承来扩展

我们首先想到应该有个基类 Weapon，它供所有各式各样的武器继承，比如说 Sword、Axe、Bow。Description 字段代表武器的说明，比如 “One-Hand light Sword”，Damage() 方法则用于获取武器的伤害，GetDescription 用于获取武器的说明。在不考虑宝石的情况下，我们得到下面的设计：



现在我们考虑如何创建镶嵌有宝石的武器。我们首先考虑到可以用继承来实现这样的设计，结果却发现如果我们需要定义所有嵌宝石的剑 (Sword)，就需要  $3+6+7 = 16$  个类 (NOTE: 三个物品孔，每个孔都有 蓝、红、绿 三种选择，可以两个或者三个孔同一色)，如果我们给镶嵌了两颗蓝一颗红宝石的剑命名为 Blue2RedSword，给三色不同不剑命名为 BlueRedGreenSword，其余的类推。那么，我们会得到下面这样庞大的类体系 (只绘制了部分)：



而这仅仅是开始，如果我们需要再添一种宝石，比如说白色，它可以附加诅咒的效果；或者我们需要给武器再添加一个物品孔，那么我们的类的数目将迅速的由十几个变成几十个。

我们发现使用继承的问题：**使用继承时将会创建出大量的类**。除此以外，使用继承，也意味着我们需要实例化一个特定的子类以获取我们需要的功能(方法)，这在编译阶段(compile time)就已经确定，类的客户端不能控制何时(run time)根据需要改变，除非再实例化另一个子类。

## 使用复合来扩展

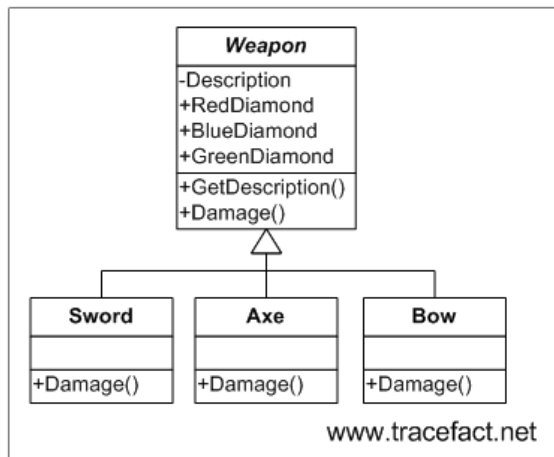
我们发现继承会带来两个主要的问题，所以我们考虑换一种方式来思考，我们可以使用复合来完成它。说详细一点，就是我们将 蓝宝石(BlueDiamond)、红宝石(RedDiamond)、绿宝石(GreenDiamond) 作为实体变量(instance variable)复合到基类中，然后在基类的 Damage() 方法中计算出所有宝石额外增加的伤害(此时基类的 Damage() 方法不再是抽象的)。

```
public abstract class Weapon{
    public virtual int Damage(){
        int total = 0;
        if(redDiamond!=null)
            total += redDiamond.Damage(); //附加红宝石的伤害
        if(blueDiamond!=null)
            total += blueDiamond.Damage(); //附加蓝宝石的伤害
        if(greenDiamond!=null)
            total += greenDiamond.Damage(); //附加绿宝石的伤害
        return total;
    }
}
```

而在实体子类中，我们覆盖这个方法，在方法内部先调用基类方法获取宝石的附加伤害，然后再给它加上武器本身的伤害。

```
public class Sword: Weapon{
    public override int Damage(){
        return base.Damage + 15; // 15 是剑本身伤害
    }
}
```

此时的图应该变成这样：



相对于继承，复合看上去要好得多，它的类的数目要少的多，并且又可以在运行时决定是否给武器镶嵌宝石，但是使用复合仍存在问题：

- 宝石与剑是紧密耦合在一起的，当我们想要为武器添加一个白宝石，那么我们需要给 Weapon 基类再添加一个 BlueDiamond 字段，同时还需要修改基类的 Damage () 方法。简言之，每次维护我们都要修改以前的代码。
- 我们遗忘了一种组合，应该记得，我们的剑是可以镶嵌三个同色宝石的，比如说：三个蓝宝石 或者 三个红宝石，那么上面的设计显然无法完成。当然，我们可以从三种宝石中抽象出一个 Diamond 基类来，而在 Weapon 中添加三个 Diamond 类型的变量。但是，问题依然存在：如果我们需要多添一个装备孔，那么我们又得再次修改 Weapon 类。

## 为对象添加状态和行为

现在假设我们不是一名软件设计者，而是一个游戏玩家，我们要为剑添加一枚红宝石，一枚蓝宝石，那么实际的操作顺序是什么呢？

1. 我们当然首先要有一把剑。(需要先创建一个 Sword 对象，它只是把剑，不含任何宝石)。
2. 我们为剑添加一个红宝石。(我们包装 Sword 对象，给它添加 3 点伤害，并给它火焰效果)。
3. 我们为剑添加一个蓝宝石。(我们包装 包含了一个红宝石的 Sword 对象，给它添加 2 点伤害，并给它冰冻效果。)

用代码来体现应该就是这样的：

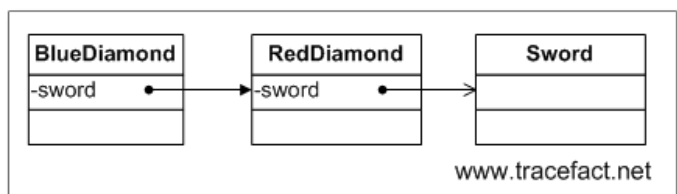
```

Weapon sword = new Sword();           // 创建一把剑
sword = new RedDiamond(sword);        // 给剑添加 红宝石
sword = new BlueDiamond(sword);       // 给剑添加 蓝宝石
  
```

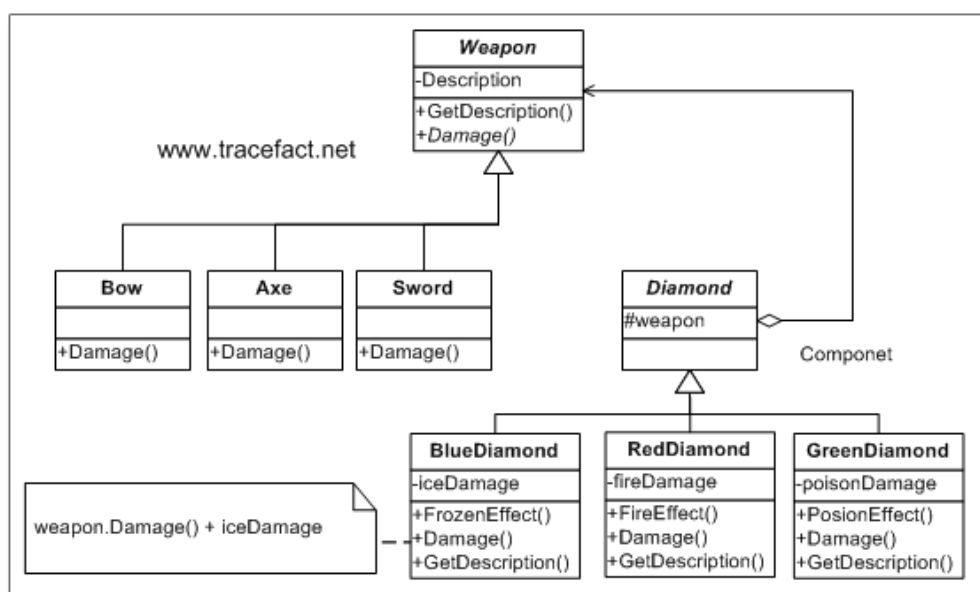
从给剑添加红宝石那句代码，我们发现第一件事：**宝石应该保存一个对剑的引用**。然后我们就可以在宝石类的内部来为 sword 添加行为或状态。

从给剑添加蓝宝石那句代码，我们发现第二件事：添加了红宝石的剑(仅从代码看它属于是宝石)，仍然是剑，所以我们得出：宝石应该和武器是同一个类型(Weapon 基类)的，不然这里将无法再次传递。

这个过程用图来体现就是这样的：



如果我们将整个过程用 UML 来表示，就构成了下面这幅图：



从图中我们可以看到，通过宝石的扩展，我们可以为剑提供新的能力：额外的伤害加成，以及额外的武器特效(抱歉我不能显示一个华丽的魔法效果，只能在黑底白字的屏幕上输出一句：Additional Effect: Fire !)。

在 Damage() 和 GetDescription() 中，我们先调用基类的相应方法，然后为 Damage() 添加来自宝石的额外的伤害(状态)：iceDamage，以及来自宝石的额外效果(行为)：FrozenEffect()。

## Decorator 设计模式

上面这幅图，就构成了 GOF 的 Decorator 设计模式，我们现在看一下它的官方定义：动态地为对象添加额外的职能。Decorator 模式为通过继承来为类扩展功能这种方式提供了另一种灵活的选择。

# 代码实现与测试

简单起见，我们只实现一种武器：Sword，两种宝石：蓝宝石 和 红宝石。

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Decorator {

    // 定义 Weapon 基类
    public abstract class Weapon {

        protected string description;           // 武器的描述(攻击效果)

        public virtual string GetDescription() {
            return description;
        }

        public abstract int Damage();           // 武器的伤害
    }

    // 定义剑
    public class Sword : Weapon {
        public Sword() {
            description = "One-Hand light Sword";
        }
        public override int Damage() {
            return 15;
        }
    }

    // 定义宝石基类
    public abstract class Diamond : Weapon {
        protected Weapon weapon;               // 保存对武器的引用
    }

    // 定义蓝宝石
    public class BlueDiamond : Diamond {
        private int iceDamage = 2;             // 蓝宝石的额外伤害

        public BlueDiamond(Weapon weapon) {
            this.weapon = weapon;               // 保存引用
        }
    }
}
```

```

    public string IceEffect(){
        return "\nAdditional Effect: Frozen !";
    }

    public override int Damage() {
        return weapon.Damage() + iceDamage;    // 攻击加成
    }

    public override string GetDescription() {
        return weapon.GetDescription() + IceEffect(); // 加入特殊攻击效果
    }
}

// 定义红宝石
public class RedDiamond : Diamond {
    private int fireDamage = 3;    // 蓝宝石的额外伤害

    public RedDiamond(Weapon weapon) {
        this.weapon = weapon;    // 保存引用
    }

    public string FireEffect() {
        return "\nAdditional Effect: Fire !";
    }

    public override int Damage() {
        return weapon.Damage() + fireDamage;    // 攻击加成
    }

    public override string GetDescription() {
        return weapon.GetDescription() + FireEffect(); // 加入特殊攻击效果
    }
}

class Program {
    static void Main(string[] args) {
        Weapon sword = new Sword();    // 创建一把新剑
        // 打印其描述(攻击效果) 和 伤害
        Console.WriteLine(sword.GetDescription() + "\nDamage:" +
sword.Damage());
        Console.WriteLine();

        sword = new BlueDiamond(sword); // 给剑添加一颗蓝宝石
        Console.WriteLine(sword.GetDescription() + "\nDamage:" + word.Damage());
    }
}

```

```
        Console.WriteLine();

        sword = new RedDiamond(sword); // 给剑添加一颗红宝石
        Console.WriteLine(sword.GetDescription() + "\nDamage:" +
sword.Damage());
        Console.WriteLine();
    }
}
}
```

输出为:

```
One-Hand light Sword
Damage:15

One-Hand light Sword
Additional Effect: Frozen !
Damage:17

One-Hand light Sword
Additional Effect: Frozen !
Additional Effect: Fire !
Damage:20
```

## 总结

本文中，我们通过一个常见的给武器(对象)添加宝石(额外的状态和行为)的例子，讨论了 Decorator 设计模式的实现过程。

我们首先提出了要解决的问题：给武器添加宝石，以使它有额外的攻击(伤害)加成和特殊的攻击效果。然后提出了使用继承会遇到的问题：大量的类以及只能通过实例化子类的方式获取行为。随后我们使用复合(Composition)的方式来解决，又遇到新的问题：程序不易维护，每次添加新的宝石或者添加新的物品孔，都需要修改代码。最后，我们使用 Decorator 模式巧妙地解决了这个问题。

希望这篇文章能给你带来帮助!