链滴

GOF 设计模式小白教程之桥接模式

作者：valarchie

原文链接：https://ld246.com/article/1567086813388

来源网站：链滴

桥接模式（Bridge）

定义：

将抽象与实现分离，使它们可以独立变化。它是用组合关系代替继承关系来实现，从而降低了抽象和现这两个可变维度的耦合度。

通俗解释：

假设我们写一个画图程序，要实现画长方形、正方形、圆形。很简单我们只需要抽象出一个Shape抽类，然后长方形、正方形、圆形继承这个Shape类，绘制方法就交给子类实现。但是如果再加上一个求，要绘制红黄蓝三种颜色的图形怎么办？如果再用继承来实现这个需求的话，光长方形就需要红长形，黄长方形，蓝长方形三个子类。依次类推，一共需要继承出九个子类。SOLID原则讲到要尽量用合来替代继承来扩展功能。多层级的继承代码耦合性非常大，而且类不利于重用。这时候就需要桥接式。通过抽象的Shape类当中去持有抽象的Color类，打破原本的继承关系，那么Shape类的扩展和Coor的扩展都可以独立的变化不需要修改原类，这也是开闭原则的一种体现。

代码：

抽象图形类，持有颜色抽象类，还有绘制的方法。（通过组合而不是继承来实现颜色的扩展）

```
public abstract class Shape {
    // 持有颜色实现
    protected Color color;
    // 抽象画图方法
    public abstract void draw();

    public Shape(Color color) {
        this.color = color;
    }
}
```

扩展的抽象图形类：长方形、正方形、圆形

```
public class Rectangle extends Shape{

    public Rectangle(Color color) {
        super(color);
    }
    @Override
    public void draw() {
        System.out.println("我是一个" + color.paint() + "的长方形");
    }
}


public class Square extends Shape{

    public Square(Color color) {
        super(color);
    }
```

```
    @Override
    public void draw() {
        System.out.println("我是一个" + color.paint() + "的正方形");
    }

}

public class Circle extends Shape {

    public Circle(Color color) {
        super(color);
    }

    @Override
    public void draw() {
        System.out.println("我是一个" + color.paint() + "的园形");
    }

}
```

扩展功能接口类Color

```
public interface Color {
    // 填色
    String paint();

}
```

具体功能实现类：红色、黄色、蓝色

```
public class Red implements Color {
    @Override
    public String paint() {
        return "红色";
    }
}

public class Yellow implements Color {
    @Override
    public String paint() {
        return "黄色";
    }
}


public class Blue implements Color {
    @Override
    public String paint() {
        return "蓝色";
    }
}
```

测试桥接模式，通过图形类持有颜色类来扩展绘制图形颜色的能力，而不是通过继承。

```
public class TestBridge {


    public static void main(String[] args) {

        // 三种不同的长方形
        Shape redRectangle = new Rectangle(new Red());
        Shape yellowRectangle = new Rectangle(new Yellow());
        Shape blueRectangle = new Rectangle(new Blue());

        redRectangle.draw();
        yellowRectangle.draw();
        blueRectangle.draw();

        // 三种不同的正方形
        Shape redSquare = new Square(new Red());
        Shape yellowSquare = new Square(new Yellow());
        Shape blueSquare = new Square(new Blue());

        redSquare.draw();
        yellowSquare.draw();
        blueSquare.draw();

        // 三个不同的圆形
        Shape redCircle = new Circle(new Red());
        Shape yellowCircle = new Circle(new Yellow());
        Shape blueCircle = new Circle(new Blue());

        redCircle.draw();
        yellowCircle.draw();
        blueCircle.draw();

    }

}
```

运行结果:

```
我是一个红色的长方形
我是一个黄色的长方形
我是一个蓝色的长方形
我是一个红色的正方形
我是一个黄色的正方形
我是一个蓝色的正方形
我是一个红色的园形
我是一个黄色的园形
我是一个蓝色的园形
```

解析：

1. 由于抽象与实现分离，所以扩展能力强；
2. 其实现细节对客户透明。
3. 由于聚合关系建立在抽象层，要求开发者针对抽象化进行设计与编程， 这增加了系统的理解与设

难度。