链滴

《Head First 设计模式》：组合模式

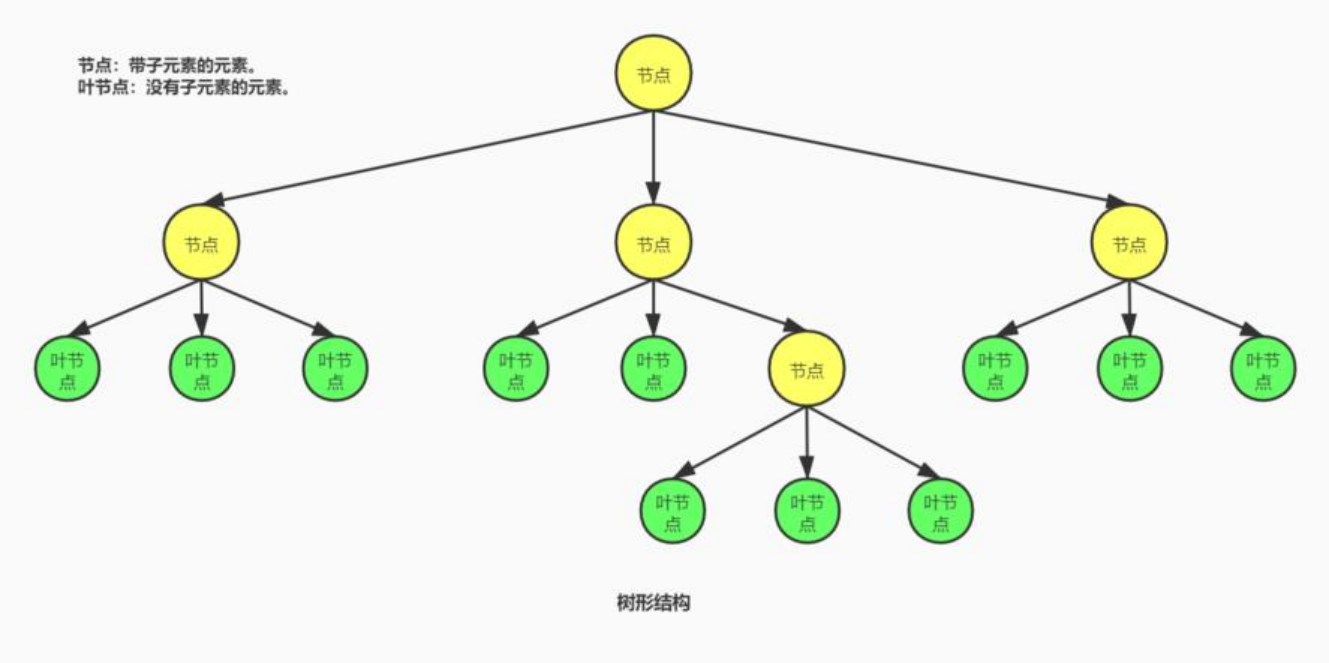
作者：jingqueyimu
原文链接：https://ld246.com/article/1600956198338
来源网站：链滴

正文

一、定义

组合模式允许你将对象合成树形结构来表现“整体/部分”层次结构。组合能让客户以一致的方式处组合对象以及个体对象。

- 组合对象：包含其他组件的组件。
- 个体对象（叶节点对象）：没有包含其他组件的组件。



要点：

- 组合结构内的任意对象称为组件，组件可以是组合，也可以是叶节点。
- 通过将组合对象和个体对象放在树形结构中，我们创建了一个“整体/部分”层次结构。如果将整树形结构视为一个“大组合”的话，那么这个树形结构的每一个“子树形结构”也是一个组合，包括节点也可以被视为一个不包含其他对象的组合。这样一来，我们就有了“以一致的方式处理”的基础。
- 所谓“以一致的方式处理”，是指组合和叶节点具有共同的方法可以调用。这就要求它们必须实现同的接口。
- 组合模式允许客户对组合对象和个体对象一视同仁。换句话说，我们可以把相同的操作应用在组合象和个体对象上。

二、实现步骤

1、创建组件抽象类

也可以使用组件接口。

组件中有些方法可能不适合某种对象，此时我们可以抛异常或者提供特定实现。

```java
/**
 * 组件抽象类
 */
public abstract class Component {

    /**
     * 子组件（可以是组合或叶节点）
     */
    protected List<Component> childs = new ArrayList<Component>();

    /**
     * 添加子组件
     */
    public void addChild(Component component) {
        childs.add(component);
    }

    /**
     * 移除子组件
     */
    public void removeChild(Component component) {
        childs.remove(component);
    }

    /**
     * 获取所有子组件
     */
    public List<Component> getChilds() {
        return childs;
    }

    public String getName() {
        // 默认抛异常，由子类决定要不要覆盖
        throw new UnsupportedOperationException();
    }
}
```

2、创建组合及叶节点，并继承组件抽象类

（1）组合

组合可以包含其他组合，也可以包含叶节点。

```java
/**
 * 组合
 */
public class Composite extends Component {

    private String name;

    public Composite(String name) {
        this.name = name;
    }
```

```
    @Override
    public String getName() {
        return name;
    }
}
```

（2）叶节点

叶节点无法添加、删除、获取子节点，因此需要对相应的方法进行特殊处理。

```
/**
 * 叶节点
 */
public class Leaf extends Component {

    private String name;

    public Leaf(String name) {
        this.name = name;
    }

    @Override
    public void addChild(Component component) {
        // 叶节点不能添加子节点，可以抛异常或者空实现
        throw new UnsupportedOperationException();
    }

    @Override
    public void removeChild(Component component) {
        // 叶节点没有子节点可移除，可以抛异常或者空实现
        throw new UnsupportedOperationException();
    }

    @Override
    public List<Component> getChilds() {
        // 叶节点没有子节点，可以抛异常或者返回空集合
        throw new UnsupportedOperationException();
    }

    @Override
    public String getName() {
        return name;
    }
}
```

3、统一使用组件的方法，操作组合及叶节点

由于组合及叶节点都实现了组件接口，因此可以使用组件的方法来操作组合及叶节点。

```
public class Test {

    public static void main(String[] args) {
```

```
        // 组合
        Component composite1 = new Composite("composite1");
        Component composite2 = new Composite("composite2");
        Component composite3 = new Composite("composite3");
        // 叶节点
        Component leaf1 = new Leaf("leaf1");
        Component leaf2 = new Leaf("leaf2");
        Component leaf3 = new Leaf("leaf3");
        Component leaf4 = new Leaf("leaf4");

        // 组合1包含组合2、3
        composite1.addChild(composite2);
        composite1.addChild(composite3);
        // 组合2包含叶节点1、2
        composite2.addChild(leaf1);
        composite2.addChild(leaf2);
        // 组合3包含叶节点3、4
        composite3.addChild(leaf3);
        composite3.addChild(leaf4);

        // 打印组件名称
        System.out.println(composite1.getName());
        for (Component child : composite1.getChilds()) {
            System.out.println(" " + child.getName());
            for (Component leaf : child.getChilds()) {
                System.out.println("  " + leaf.getName());
            }
        }
    }
}
```

三、举个栗子

1、背景

对象村餐厅和对象村煎饼屋合并了，它们合并后的新公司创建了一个 Java 版本的女招待。这个 Java 本的女招待能够打印使用 ArrayList 存储的餐厅菜单和煎饼屋菜单。

现在它们打算加上一份餐后甜点的“子菜单”。也就是说，这个 Java 版本的女招待不仅要支持打印个菜单，还要支持打印菜单中的菜单。

2、实现

由于菜单可能包含菜单和菜单项，因此我们可以创建一个树形结构，这个树形结构由菜单和菜单项组。

通过将菜单和菜单项放在相同的结构中，我们既可以把整个结构视为一个“大菜单”，也可以把这个构的任一部分视为一个“子菜单”，包括菜单项也可以视为一个“我本身就是菜单项因此没必要再包菜单项的菜单”。这样，我们就可以以一致的方式来处理菜单和菜单项了。

（1）创建菜单组件抽象类

```java
/**
 * 菜单组件抽象类
 */
public abstract class MenuComponent {

    /**
     * 子组件（可以是菜单或菜单项）
     */
    protected List<MenuComponent> childs = new ArrayList<MenuComponent>();

    /**
     * 添加子组件
     */
    public void addChild(MenuComponent component) {
        childs.add(component);
    }

    /**
     * 移除子组件
     */
    public void removeChild(MenuComponent component) {
        childs.remove(component);
    }

    /**
     * 获取所有子组件
     */
    public List<MenuComponent> getChilds() {
        return childs;
    }

    public String getName() {
        throw new UnsupportedOperationException();
    }

    public double getPrice() {
        throw new UnsupportedOperationException();
    }

    public void print() {
        throw new UnsupportedOperationException();
    }
}
```

（2）创建菜单，并继承菜单组件抽象类

```java
/**
 * 菜单（组合）
 */
public class Menu extends MenuComponent {

    private String name;

    public Menu(String name) {
```

```
        this.name = name;
    }

    @Override
    public String getName() {
        return name;
    }

    @Override
    public void print() {
        System.out.println("\n" + getName());;
        System.out.println("---------------------");;
    }
}
```

（3）创建菜单项，并继承菜单组件抽象类

```
/**
 * 菜单项（叶节点）
 */
public class MenuItem extends MenuComponent {

    private String name;
    private double price;

    public MenuItem(String name, double price) {
        this.name = name;
        this.price = price;
    }

    @Override
    public void addChild(MenuComponent component) {
        throw new UnsupportedOperationException();
    }

    @Override
    public void removeChild(MenuComponent component) {
        throw new UnsupportedOperationException();
    }

    @Override
    public List<MenuComponent> getChilds() {
        return childs;
    }

    @Override
    public String getName() {
        return name;
    }

    @Override
    public double getPrice() {
        return price;
    }
```

```
    @Override
    public void print() {
        System.out.println("  " + getName() + ", " + getPrice());;
    }
}
```

（4）创建女招待

```
/**
 * 女招待
 */
public class Waitress {

    MenuComponent allMenus;

    public Waitress(MenuComponent allMenus) {
        this.allMenus = allMenus;
    }

    public void printMenu() {
        print(allMenus);
    }

    private void print(MenuComponent menuComponent) {
        menuComponent.print();
        for (MenuComponent child : menuComponent.getChilds()) {
            print(child);
        }
    }
}
```

（5）使用女招待打印菜单

```
public class Test {

    public static void main(String[] args) {
        // 所有菜单
        MenuComponent allMenus = new Menu("ALL MENUS");
        // 子菜单
        MenuComponent pancakeHouseMenu = new Menu("PANCAKE HOUSE MENU");
        MenuComponent dinerMenu = new Menu("DINER MENU");
        MenuComponent dessertMenu = new Menu("DESSERT MENU");

        // 添加煎饼屋菜单及菜单项
        allMenus.addChild(pancakeHouseMenu);
        pancakeHouseMenu.addChild(new MenuItem("Regular Pancake Breakfast", 2.99));
        pancakeHouseMenu.addChild(new MenuItem("Blueberry Pancakes", 3.49));
        pancakeHouseMenu.addChild(new MenuItem("Waffles", 3.59));
        // 添加餐厅菜单及菜单项
        allMenus.addChild(dinerMenu);
        dinerMenu.addChild(new MenuItem("BLT", 2.99));
        dinerMenu.addChild(new MenuItem("Soup of the day", 3.29));
```

```
        dinerMenu.addChild(new MenuItem("Hotdog", 3.05));
        // 添加甜点菜单及菜单项
        dinerMenu.addChild(dessertMenu);
        dessertMenu.addChild(new MenuItem("Apple Pie", 1.59));
        dessertMenu.addChild(new MenuItem("Cheesecake", 1.99));
        dessertMenu.addChild(new MenuItem("Sorbet", 1.89));

        // 使用女招待打印菜单
        Waitress waitress = new Waitress(allMenus);
        waitress.printMenu();
    }
}
```