# 组合模式

组合模式（Composite Pattern），又叫部分整体模式，是用于把一组相似的对象当作一个单一的对象。组合模式依据树形结构来组合对象，用来表示部分以及整体层次。这种类型的设计模式属于结构型模式，它创建了对象组的树形结构。

这种模式创建了一个包含自己对象组的类。该类提供了修改相同对象组的方式。

## 介绍

**意图：**将对象组合成树形结构以表示"部分-整体"的层次结构。组合模式使得用户对单个对象和组合对象的使用具有一致性。

**主要解决：**它在我们树型结构的问题中，模糊了简单元素和复杂元素的概念，客户程序可以向处理简单元素一样来处理复杂元素，从而使得客户程序与复杂元素的内部结构解耦。

**何时使用：** 1、您想表示对象的部分-整体层次结构（树形结构）。 2、您希望用户忽略组合对象与单个对象的不同，用户将统一地使用组合结构中的所有对象。

**如何解决：**树枝和叶子实现统一接口，树枝内部组合该接口。

**关键代码：**树枝内部组合该接口，并且含有内部属性 List，里面放 Component。

**应用实例：** 1、算术表达式包括操作数、操作符和另一个操作数，其中，另一个操作符也可以是操作树、操作符和另一个操作数。 2、在 JAVA AWT 和 SWING 中，对于 Button 和 Checkbox 是树叶，Container 是树枝。

**优点：** 1、高层模块调用简单。 2、节点自由增加。

**缺点：**在使用组合模式时，其叶子和树枝的声明都是实现类，而不是接口，违反了依赖倒置原则。

**使用场景：**部分、整体场景，如树形菜单，文件、文件夹的管理。

**注意事项：**定义时为具体类。

## 安全性与透明性

组合模式中必须提供对子对象的管理方法，不然无法完成对子对象的添加删除等等操作，也就失去了灵活性和扩展性。但是管理方法是在Component中就声明还是在Composite中声明呢？

一种方式是在Component里面声明所有的用来管理子类对象的方法，以达到Component接口的最大化（如下图所示）。目的就是为了使客户看来在接口层次上树叶和分支没有区别——透明性。但树叶是不存在子类的，因此Component声明的一些方法对于树叶来说是不适用的。这样也就带来了一些安全性问题。

另一种方式就是只在Composite里面声明所有的用来管理子类对象的方法（如下图所示）。这样就避免了上一种方式的安全性问题，但是由于叶子和分支有不同的接口，所以又失去了透明性。

## 优缺点

从上面的举例中可以看到，组合模式有以下优点：

1)         使客户端调用简单，客户端可以一致的使用组合结构或其中单个对象，用户就不必关心自己处理的是单个对象还是整个组合结构，这就简化了客户端代码。

2)       更容易在组合体内加入对象部件. 客户端不必因为加入了新的对象部件而更改代码。这一点符合开闭原则的要求，对系统的二次开发和功能扩展很有利！

当然组合模式也少不了缺点：组合模式不容易限制组合中的构件。