来源：https://blog.csdn.net/liang19890820/article/details/66974516

1. 装饰者模式



* Component（抽象构件）：给出一个抽象接口，以规范准备接收附加责任的对象。
* ConcreteComponent（具体构件）：定义一个将要接收附加责任的类。
* Decorator（抽象装饰类）：持有一个构件（Component）对象的实例，并实现一个与抽象构件接口一致的接口。
* ConcreteDecorator（具体装饰类）：负责给构件对象添加上附加的责任。

1. 桥接模式



* Abstraction（抽象类）：用于定义抽象类的接口，并且维护一个指向 Implementor 实现类的指针。它与 Implementor 之间具有关联关系。
* RefinedAbstraction（扩充抽象类）：扩充由 Abstraction 定义的接口，在 RefinedAbstraction 中可以调用在 Implementor 中定义的业务方法。
* Implementor（实现类接口）：定义实现类的接口，这个接口不一定要与 Abstraction 的接口完全一致，事实上这两个接口可以完全不同。
* ConcreteImplementor（具体实现类）：实现了 Implementor 定义的接口，在不同的 ConcreteImplementor 中提供基本操作的不同实现。在程序运行时，ConcreteImplementor 对象将替换其父类对象，提供给 Abstraction 具体的业务操作方法。

1. 组合模式

安全组合模式



* Component（抽象构件）：为叶子构件和容器构件对象定义接口，可以包含所有子类共有行为的声明和实现。在抽象构件中，声明了访问及管理子构件的接口（例如：Add()、Remove()、GetChild() 等）。
* Leaf（叶子构件）：叶子节点没有子节点。它实现了 Component 中定义的行为，对于访问及管理子构件的接口，可以通过异常等方式进行处理。
* Composite（容器构件）：容器节点包含子节点（可以是叶子构件，也可以是容器构件）。它提供了一个集合用于存储子节点，实现了 Component 中定义的行为，包括访问及管理子构件的接口，在其业务方法中可以递归调用其子节点的业务方法

适用场景：

* 表示对象的“整体-部分”层次结构（树形结构）
* 希望用户忽略组合对象与单个对象的不同，统一地使用组合结构中的所有对象。