# 组合模式

## 一、基本概念 **1、定义：**

##### 组合模式（Composite），将对象组合成树状结构图，用此来表示“部分和整体”的结构。组合模式让单个对象和组合对象的使用具有一致性。也就是叶和茎是在同一层的。

##### 优势是可以很好的展现这种整体与部分的结构，并且以树状图的方式进行展现。

### 2、组成环节：

##### 1、Component(组合类):组合里面的抽象声名接口，实现所有类的默认行为，并且访问和管理其子类。

##### 2、Leaf(叶子类)：在组合模式中表示叶节点对象，叶节点没有自己的子节点。

##### 3、Composite(茎类)：可以通过Add和Delete方法来存储或者删除自己的子部件，可以存储茎类和叶类。

## 二、go语言中实现

##### 1、首先定义一个Component类型，并且定义茎和叶所有的行为。

type Component interface {  
 Add(cp Component)  
 Delete(cp Component)  
 Name() string  
 Display()  
}

##### 2、定义叶类，并且对其方法进行实现

type Leaf struct {  
 name string  
}  
  
func (l \*Leaf) Add(cp Component) {} //no mean  
  
func (l \*Leaf) Delete(cp Component) {} //no mean  
  
func (l \*Leaf) Display() {  
 fmt.Printf("it's leaf %s\n", l.name)  
}

func (l \*Leaf) Name() string {  
 return l.name  
}

##### 3、定义茎类，并实现其所有方法

type Composite struct {  
 cs []Component  
 name string  
}  
  
func (c \*Composite) Add(cp Component) {  
 c.cs = append(c.cs, cp)  
}  
  
func (c \*Composite) Delete(cp Component) {  
 for i, com := range c.cs {  
 if com.Name() == cp.Name() {  
 c.cs = append(c.cs[:i], c.cs[i+1:]...)  
 }  
 }  
}  
  
func (c \*Composite) Display() {  
 result := ""  
 for \_, com := range c.cs {  
 result += com.Name()  
 }  
 fmt.Printf("the composite it's contain %s\n", result)  
}  
  
func (c \*Composite) Name() string {  
 return c.name  
}

##### 4、开始进行main方法的调用。

func main() {  
 l1 := &Leaf{name: "l1"}  
 l2 := &Leaf{name: "l2"}  
 c1 := &Composite{name: "c1"}  
 c2 := &Composite{name: "c2"}  
 c1.Add(l1)  
 c1.Add(c2)  
 c2.Add(l2)  
 l1.Display()  
 l2.Display()  
 c1.Display()  
 c2.Display()  
}

##### 5、结果的输出。

it's leaf l1

it's leaf l2

the composite it's contain l1c2

the composite it's contain l2