设计模式-组合模式

参考

- 组合模式

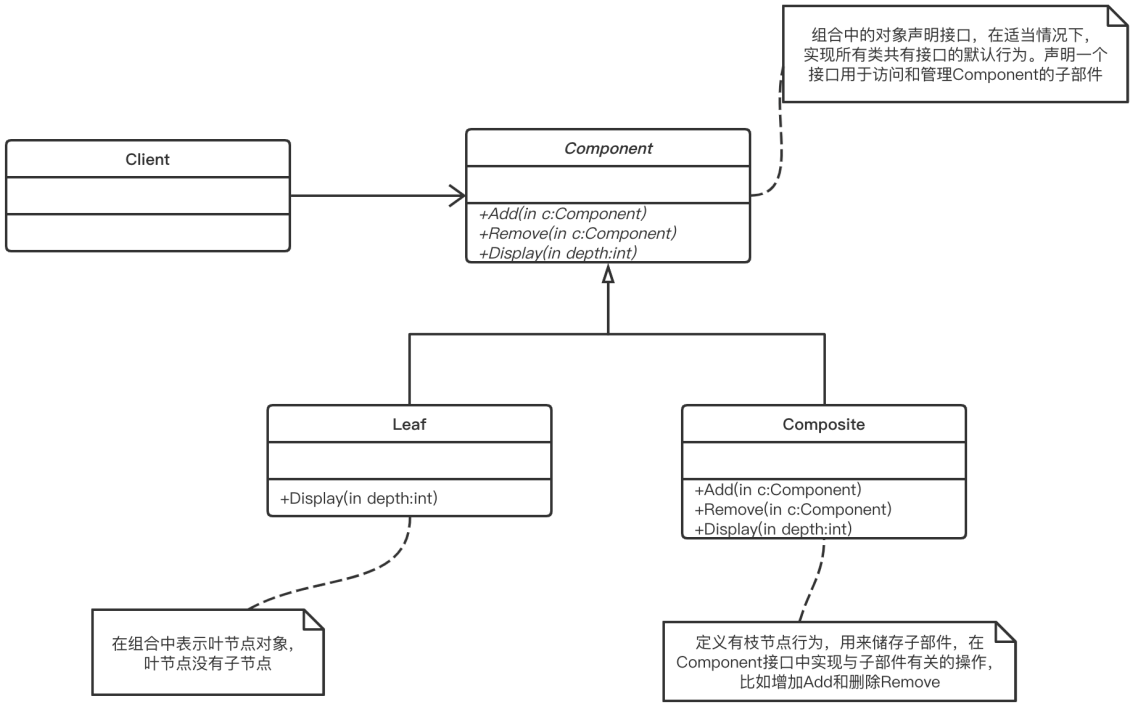
组合模式，又叫部分整体模式，适用于把一组相似的对象当做一个单一的对象。组合模式依据属性结构来组合对象，用来表示部分以及整体层次。这种类型的设计模式属于结构型模式，它创建了对象组的树形结构。

> 要点：
>
> 通过组合模式我们可以提取一组对象中的相似的部分，同时可保持对象的特性。在保持对象特性的同时可以提高代码的复用。

使用场景

- 文件管理、组织管理

Demo分析



> 如上图我门简单制作一个文件管理系统，其中Composite时目录，Leaf时目录下的文件，目录和文件都继承自Component。目录能够增加、删除文件，可以展示目录所在位置，文件只能展示文件所在位置。

按照上述的分析，我们将文件管理系统拆分为两个类：文件类和目录类。这两个类都继承于同一个父类，这样我们可以复用功能的同时消除了两个类调用上的区别。

Go代码实现

```go
package main

import "fmt"

const Separator = "--"

/**
 * @Description: 文件系统的抽象类,文件和文件夹都实现该接口
 */
type FileSystemNode interface {
    Display(Separator string)
}

/**
 * @Description: 文件的通用功能
 */
type FileCommonFunc struct {
    fileName string
}

/**
* @Description: 设置文件名
 */
func (f *FileCommonFunc) SetFileName(fileName string) {
    f.fileName = fileName
```

```
}

/**
 * @Description: 文件类
 */
type FileNode struct {
    FileCommonFunc
}

/**
* @Description: 显示文件内容
 */
func (f *FileNode) Display(Separator string) {
    fmt.Println(Separator + f.fileName + "   文件内容为: Hello world")
}

/**
 * @Description: 目录类
 */
type DirectoryNode struct {
    FileCommonFunc
    nodes []FileSystemNode
}

/**
* @Description: 显示目录中的所有内容
 */
func (d *DirectoryNode) Display(Separator string) {
    fmt.Println(Separator + d.fileName + "   文件夹内容为:")
    for _, node := range d.nodes {
        node.Display(Separator + Separator)
    }
}

/**
 * @Description: 添加目录或者文件
 */
func (d *DirectoryNode) Add(f FileSystemNode) {
    d.nodes = append(d.nodes, f)
}

func test() {
    root := &DirectoryNode{}
    root.SetFileName("root")
    var file1 = &FileNode{}
    file1.SetFileName("file1")
    var file2 = &FileNode{}
    file2.SetFileName("file2")
    root.Add(file1)
    root.Add(file2)
    root.Display(">>")
}

func main() {
    test()
```

```
}
```

输出

```
>>root   文件夹内容为:
>>>>file1   文件内容为: Hello world
>>>>file2   文件内容为: Hello world
```

Python实现

```python
class ComponentBases:
  """
    ## 文件抽象类
  """
  def __init__(self, fileName):
    self.fileName = fileName
    pass
  def setFileName(self, fileName):
    self.fileName = fileName
  def display(self, Separator:str):
    pass

class FileNode(ComponentBases):
  """
    ## 文件类
  """
  def __init__(self, fileName):
    super().__init__(fileName)
  def display(self, Separator:str):
    print(f"{Separator}FileNode: {self.fileName}")

class directoryFileNode(ComponentBases):
  """
    ## 目录类
  """
  def __init__(self, fileName):
    super().__init__(fileName)
    self.children = []
  def add(self, obj):
    self.children.append(obj)
  def display(self, Separator:str):
    print(f"{Separator}{self.fileName}文件内容为:")
    for child in self.children:
      child.display(Separator+Separator)

def test():
  root = directoryFileNode("root")
  root.add(FileNode("file1"))
  root.add(FileNode("file2"))
  file2 = directoryFileNode("root2")
  file2.add(FileNode("file3"))
  root.add(file2)
```

```
  root.display(">>")

if __name__ == '__main__':
  test()
```

输出

```
>>root文件内容为:
>>>>FileNode: file1
>>>>FileNode: file2
>>>>root2文件内容为:
>>>>>>>>FileNode: file3
```

总结

> 比较上面两种语言的写法， python的写法似乎更加易于理解，而go语言中的写法似乎由于写法的松散导致一眼看起来比较复杂。但两者的思想是一致的。