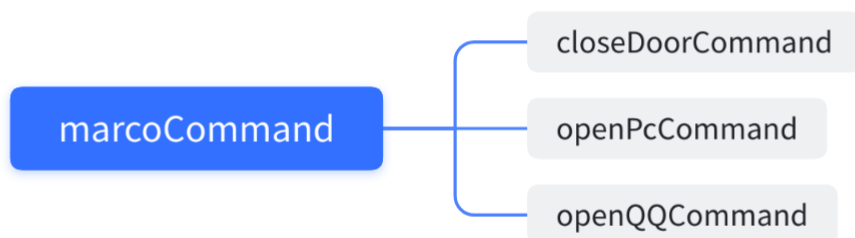


10 组合模式

其实程序设计跟自然一样，也存在“事物是由相似的子事物构成”的思想，例如原子和太阳系的组成一样。组合模式就是借助小的子对象来构建更大的对象

回顾宏命令

之前第九章的宏命令就是一组具体的子命令集合对象，但是不管宏命令自身还是子命令都拥有关键的execute方法，并且如果还有孙命令的话也是同样的道理，最终形成了一级一级的树状结构



marcoCommand虽然变通的像是一个命令，但其实只有内部子命令的代理，只负责传递请求给叶对象

用途

1. 表示树形结构：回顾上面的例子即可，组合模式提供了一种遍历树形结构的方案，例如递归调用叶对象的execute方法来完成抽象操作的实现。非常方便的部署对象部分->整体的层次结构
2. 利用对象多态性统一组合对象和单个对象：利用对象多态性使得客户端可以忽略组合对象和单个对象的不同，对于用户来说透明（上面例子的execute），并不需要关心它究竟是组合对象还是单个对象。例如用户使用遥控器不需要关注它的命令内部实现或者是宏命令还是单个命令，命令只要有execute方法即可嵌入遥控器

更强大的宏命令

这里我们实现一个更强大的遥控器，同时了解请求在复杂组合树中的传递过程：

- 打开空调
- 打开电视和音响
- 关门，开电脑，登录QQ

```
1  /**
2   * 更强大宏命令
3   */
4  let closeDoorCommand = {
5    execute: () => {
6      console.log('关门');
7    }
8  }
9
10 let openPCCommand = {
11   execute: () => {
12     console.log('打开电脑');
13   }
14 }
15
16 let openQQCommand = {
17   execute: () => {
18     console.log('打开QQ');
19   }
20 }
21
22 let openAcCommand = {
23   execute: () => {
24     console.log('开空调');
25   }
26 }
27
28 let openTvCommand = {
29   execute: () => {
30     console.log('打开电视');
31   }
32 }
33
34 let openSoundCommand = {
35   execute: () => {
36     console.log('打开音响');
37   }
38 }
```

```

38 }
39
40 const MacroCommand = () => {
41     return {
42         commandsList: [],
43         add: (command) => {
44             this.commandsList.push(command);
45         },
46         execute: () => {
47             for(let i = 0, command; command = this.commandsList[i++];){
48                 command.execute();
49             }
50         }
51     }
52 }
53
54 const macroCommand1 = MacroCommand();
55 macroCommand1.add(openTvCommand);
56 macroCommand1.add(openSoundCommand);
57
58 const macroCommand2 = MacroCommand();
59 macroCommand.add(closeDoorCommand);
60 macroCommand.add(openPCCCommand);
61 macroCommand.add(openQQCommand);
62 macroCommand.execute();
63
64 const macroCommand = MacroCommand();
65 macroCommand.add(openAcCommand);
66 macroCommand.add(macroCommand1);
67 macroCommand.add(macroCommand2);
68
69 // 给遥控器绑定超级命令
70 var setCommand = (function(command){
71     document.getElementById('button').onclick = function(){
72         command.execute();
73     }
74 })(macroCommand)

```

最终这个复杂的宏命令组合树的运转非常简单，只需要调用最上层的execute即可，创建组合对象的程序员也不需要关心内在的execute细节，同时根据需求往里面更改节点对象也非常方便

抽象类在组合模式的作用

前面提到组合模式最大优点可以一致地对待组合对象和基本对象，只要它是一个命令，它有execute方法就可以。

这种透明性带来的便利，在静态类型语言中尤为明显。例如Java中的关键就是他们必须继承一个抽象类，这个抽象类即代表组合对象又代表叶对象，保证拥有同样名字的方法以及相同的反馈。

```
1 public abstract class Component {
2     public void add(Component child){}
3
4     public void remove(Component child){}
5 }
6 // 具体类
7 public class Composite extends Component {
8     public void add(Component child){}
9
10    public void remove(Component child){}
11 }
12
13 public class Leaf extends Component {
14     public void add(Component child){throw new UnsupportedOperationException()}
15
16     public void remove(Component child){}
17 }
```

然而在JS这种动态语言中对象的多态性是与之俱来的，没有编译器去检查（现在可以通过TS来实现），一般可以用鸭子类型的思想对他们检查

缺点是不严谨，不太安全，但是好处是快速且自由