组合模式

2020年2月11日 15:42

1. 概念

它是一种将对象组合成树形结构以表示"部分 - 整体"的层次结构。 例如目录与目录中的文件，大学中的部门与人员等。

组合模式使得客户端对单个对象和组合对象保持一致的方式处理。

2. UML结构图

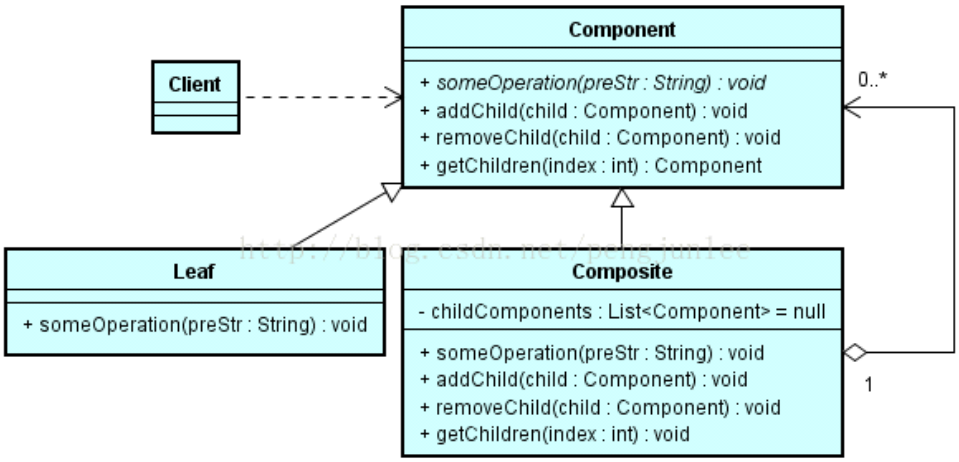

图2.1 组合模式UML结构图

1. 抽象组件(Component)角色：为组合对象和叶子对象声明公共的接口，让客户端可以通过这个接口来访

问和管理整个对象树。

2. 组合对象(Composite)角色：通常会存储子组件(组合对象、叶子对象)，定义属于自己行为，并实现在

抽象组件中定义的与子组件有关的操作，例如子组件的添加(addChild)和删除(removeChild)等。

3. 叶子对象(Leaf)角色：定义和实现叶子对象的行为，并且它不再包含其他的子节点对象。
4. 客户端(Client)角色：通过Component接口来统一操作组合对象和叶子对象，以创建出整个对象树结

构。

3. 组合模式实例

在一个在线慕课网站中，通常目录与课程之间存在组合关系，也可以看作树形关系，通过组合模式，将课程单个对象和目录组合对象保持一致的处理。

1. 抽象组件类

```
public abstract class CatalogComponent {
    public void add(CatalogComponent catalogComponent){
        throw new UnsupportedOperationException("不支持添加操作");
    }
    public void remove(CatalogComponent catalogComponent){
        throw new UnsupportedOperationException("不支持删除操作");
    }
    public String getName(){
        throw new UnsupportedOperationException("不支持获取名称操作");
    }
    public Double getPrice(){
```

```
            throw new UnsupportedOperationException("不支持获取价格操作");
        }
        public void print(){
            throw new UnsupportedOperationException("不支持打印操作");
        }
}
```

2. 叶子对象类

```
public class Course extends CatalogComponent {
    private String name;
    private Double price;
    public Course(String name, Double price) {
        this.name = name;
        this.price = price;
    }

    @Override
    public String getName() {
        return this.name;
    }

    @Override
    public Double getPrice() {
        return this.price;
    }

    @Override
    public void print() {
        System.out.println("打印课程: " + name + ",价格为: " + price);
    }
}
```

3. 组合对象类

```
public class CourseCatalog extends CatalogComponent {
    private List<CatalogComponent> items = new ArrayList<CatalogComponent>();
    private String name;  // 目录名
    private Integer level;

    public CourseCatalog(String name, Integer level) {
        this.name = name;
        this.level = level;
    }

    @Override
    public void add(CatalogComponent catalogComponent) {
        items.add(catalogComponent);
    }

    @Override
    public void remove(CatalogComponent catalogComponent) {
        items.remove(catalogComponent);
    }

    @Override
    public void print() {
        System.out.println(this.name);
        for (CatalogComponent catalogComponent: items){
            if (this.level != null){
                for (int i = 0; i < this.level; i++){
                    System.out.print(" ");
                }
            }
            catalogComponent.print();
        }
    }
}
```

4. 客户端

```
@Test
    public void compositeTest(){
        CatalogComponent linuxCourse = new Course("Linux学习",123.3);
        CatalogComponent pythonCourse = new Course("Python开发",23.3);

        CatalogComponent course = new Course("Java Web开发",23.3);
        CatalogComponent course1 = new Course("Java 并行开发",20.5);
        CatalogComponent course2 = new Course("Java 虚拟机学习",13.3);
        CatalogComponent javaCourseCatalog = new CourseCatalog("Java课程目录",2);
        javaCourseCatalog.add(course);
        javaCourseCatalog.add(course1);
        javaCourseCatalog.add(course2);

        CatalogComponent iMoocCourseCatalog = new CourseCatalog("慕课网课程目录",1);
        iMoocCourseCatalog.add(linuxCourse);
        iMoocCourseCatalog.add(pythonCourse);
```

```
        iMoocCourseCatalog.add(javaCourseCatalog);
        iMoocCourseCatalog.print();

        // linuxCourse 和 iMoocCourseCatalog(组合对象)可以进行一致性处理
        /*linuxCourse.print();
        String linuxName = linuxCourse.getName();*/

    }
```

5. 结果

```
慕课网课程目录
 打印课程: Linux学习,价格为: 123.3
 打印课程: Python开发,价格为: 23.3
 Java课程目录
   打印课程: Java Web开发,价格为: 23.3
   打印课程: Java 并行开发,价格为: 20.5
   打印课程: Java 虚拟机学习,价格为: 13.3
```

4. 优/缺点

1. 优点

- 清楚地定义分层次的复杂对象，表示对象的全部或部分层次
- 让客户端忽略层次的差异，方便对整个层次结构进行控制
- 简化客户端代码
- 符合开闭原则

2. 缺点

- 限制类型时会比较复杂
- 使设计变得更加抽象

5. 适用场景

- 希望客户端可以忽略组合对象和单个对象的差异时
- 处理一个树形结构时