访问者模式

2020年2月23日 14:11

1. 概念

将作用于某种数据结构中的各元素的操作分离出来封装成独立的类，使其在不改变数据结构的前提下可以添加作用于这些元素的新的操作，为数据结构中的每个元素提供多种访问方式。它将对数据的操作与数据结构进行分离，是行为类模式中最复杂的一种模式。

主要解决的情景时 有些集合对象(List/Set/Map)存在多种不同的元素，且每种不同的对象也存在多种不同的访问者和处理方式。

2. UML结构图

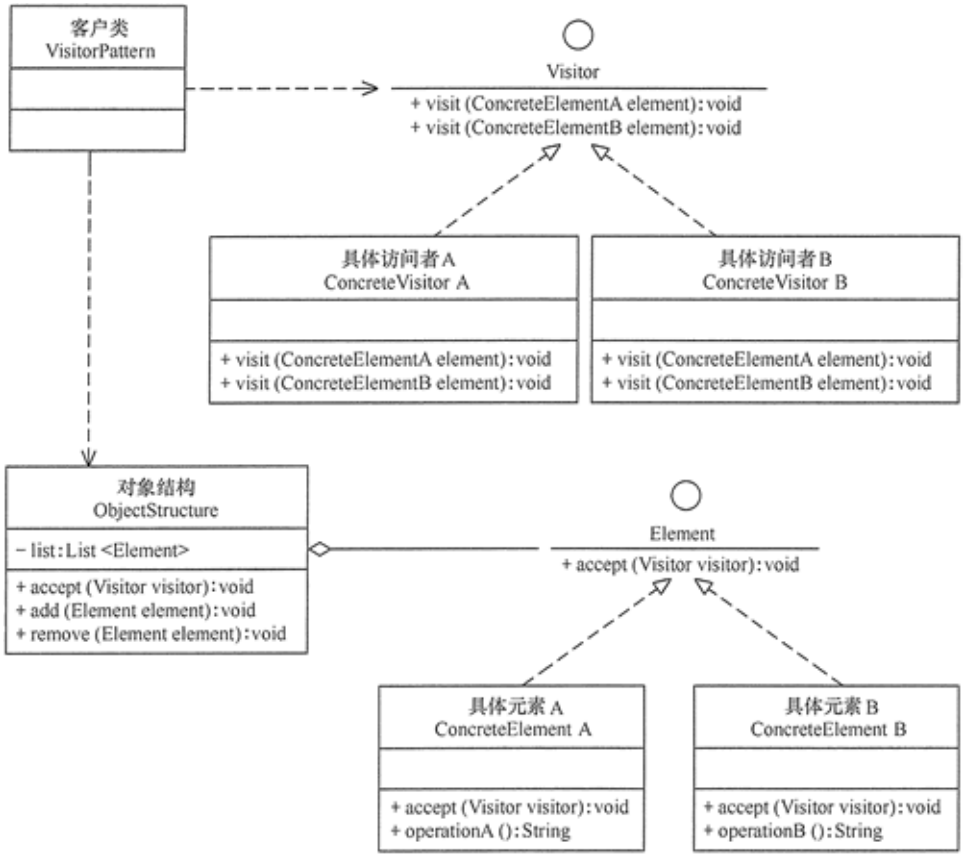

图2.1 访问者模式UML结构图

1. 抽象访问者（Visitor）角色：定义一个访问具体元素的接口，为每个具体元素类对应一个访问操作 visit()，该操作中的参数类型标识了被访问的具体元素。
2. 具体访问者（ConcreteVisitor）角色：实现抽象访问者角色中声明的各个访问操作，确定访问者访问一个元素时该做什么。
3. 抽象元素（Element）角色：声明一个包含接受操作 accept() 的接口，访问者对象作为 accept() 方法的参数。
4. 具体元素（ConcreteElement）角色：实现抽象元素角色提供的 accept() 操作，其方法体通常都是 visitor.visit(this)，另外具体元素中可能还包含本身业务逻辑的相关操作。
5. 对象结构（Object Structure）角色：是一个包含元素角色的容器，提供让访问者对象遍历容器中的所有元素的方法，通常List、Set、Map 等聚合类实现。

3. 访问者模式实现

1. 抽象访问者类

```
interface Visitor
{
    void visit(ConcreteElementA element);
    void visit(ConcreteElementB element);
}
```

2. 具体访问者类

```
class ConcreteVisitorA implements Visitor
{
    public void visit(ConcreteElementA element)
    {
        System.out.println("具体访问者A访问-->"+element.operationA());
    }
    public void visit(ConcreteElementB element)
    {
        System.out.println("具体访问者A访问-->"+element.operationB());
    }
}
```

```
class ConcreteVisitorB implements Visitor
{
    public void visit(ConcreteElementA element)
    {
        System.out.println("具体访问者B访问-->"+element.operationA());
    }
    public void visit(ConcreteElementB element)
    {
        System.out.println("具体访问者B访问-->"+element.operationB());
    }
}
```

3. 抽象元素类

```
interface Element
{
    void accept(Visitor visitor);
}
```

4. 具体元素类

```
class ConcreteElementA implements Element
{
    public void accept(Visitor visitor)
    {
        visitor.visit(this);
    }
    public String operationA()
    {
        return "具体元素A的操作。";
    }
}
```

```
class ConcreteElementB implements Element
{
    public void accept(Visitor visitor)
    {
        visitor.visit(this);
    }
    public String operationB()
    {
        return "具体元素B的操作。";
    }
}
```

5. 对象结构

```
class ObjectStructure
{
    private List<Element> list=new ArrayList<Element>();
    public void accept(Visitor visitor)
    {
        Iterator<Element> i=list.iterator();
        while(i.hasNext())
        {
            ((Element) i.next()).accept(visitor);
        }
    }
    public void add(Element element)
    {
        list.add(element);
    }
    public void remove(Element element)
    {
        list.remove(element);
    }
```

6. 客户端

```
public static void main(String[] args)
   {
       ObjectStructure os=new ObjectStructure();
       os.add(new ConcreteElementA());
       os.add(new ConcreteElementB());
       Visitor visitor=new ConcreteVisitorA();
       os.accept(visitor);
       System.out.println("------------------------");
       visitor=new ConcreteVisitorB();
       os.accept(visitor);
   }
```

7. 结果

```
具体访问者A访问-->具体元素A的操作。
具体访问者A访问-->具体元素B的操作。
------------------------
具体访问者B访问-->具体元素A的操作。
具体访问者B访问-->具体元素B的操作。
```

4. 优/缺点

1. 优点

- 扩展性好，增加新的操作很容易，即增加一个新的访问者。
- 复用性好。可以通过访问者来定义整个对象结构通用的功能，从而提高系统的复用程度。
- 灵活性好。访问者模式将数据结构与作用于结构上的操作解耦，使得操作集合可相对自由地演化而不影响系统的数据结构。
- 符合单一职责原则。访问者模式把相关的行为封装在一起，构成一个访问者，使每一个访问者的功能都比较单一。

2. 缺点

- 增加新的元素类很困难。在访问者模式中，每增加一个新的元素类，都要在每一个具体访问者类中增加相应的具体操作，这违背了“开闭原则”。
- 破坏了封装性。访问者模式中的具体元素对访问者公布细节，破坏了对象的封装性。
- 违反了依赖倒置原则，访问者模式依赖了具体元素，而没有依赖抽象类。

5. 适用场景

1. 对象结构相对稳定，但其操作算法经常变化的程序。
2. 对象结构中的对象需要提供多种不同且不相关的操作，而且要避免让这些操作的变化影响对象的结构。
3. 对象结构包含很多类型的对象，希望对这些对象实施一些依赖于其具体类型的操作。

6. 扩展

1. 与“迭代器模式”联用。因为访问者模式中的“对象结构”是一个包含元素角色的容器，当访问者遍历容器中的所有元素时，常常要用迭代器。
2. 访问者（Visitor）模式同“组合模式”联用。因为访问者（Visitor）模式中的“元素对象”可能是叶子对象或者是容器对象，如果元素对象包含容器对象，就必须用到组合模式，其结构图如图所示。

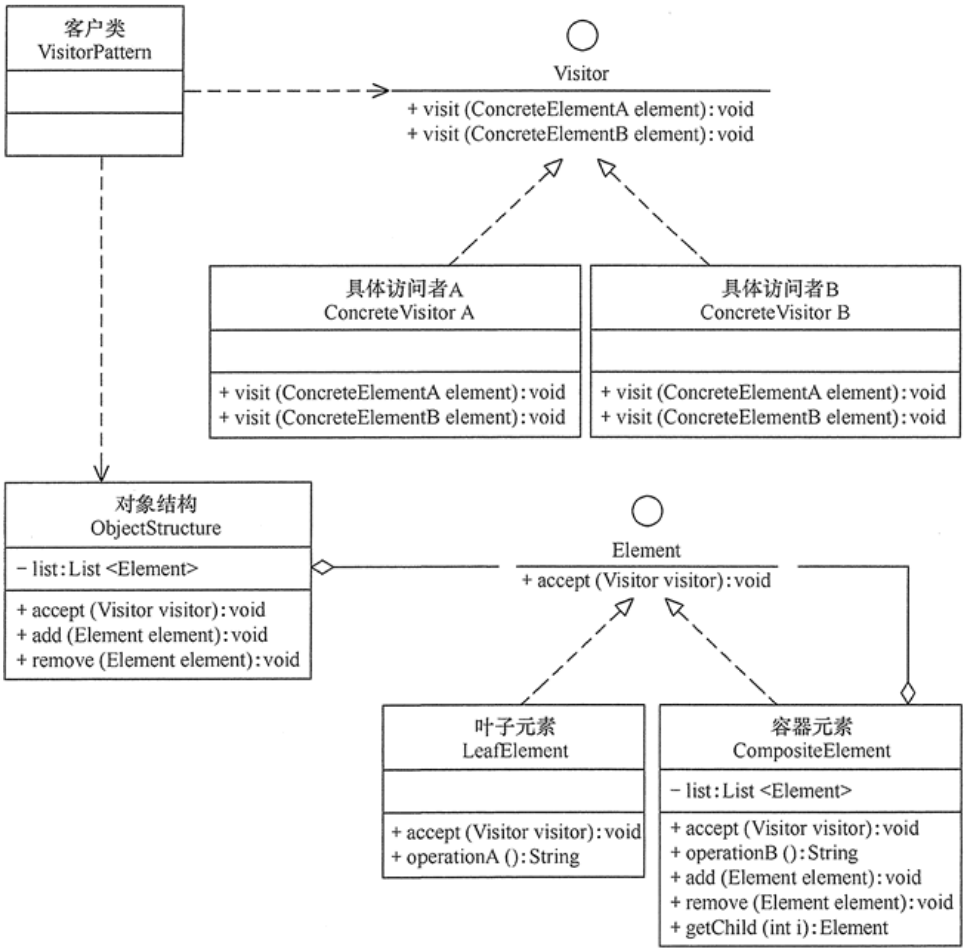

图6.1 访问者模式和组合模式结合使用

7. 参考资料

[1] http://c.biancheng.net/view/1397.html