链滴

GOF 设计模式小白教程之访问者模式

作者：valarchie

原文链接：https://ld246.com/article/1567441031680

来源网站：链滴

访问者模式（Visitor）

定义：

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。它将对数据的操作数据结构进行分离，是行为类模式中最复杂的一种模式。

通俗解释：

访问者模式的确挺不好理解的。我们举一个非常简单的例子，例如年底服装公司要盘点库存了。仓库服也有裤子，小A负责盘点衣服，小B负责盘点裤子。如果将这两个盘点都写在仓库类当中的话，一旦品类（比如鞋子）出现的话，又需要在仓库类中修改代码了，所以为了遵守开闭原则，我们将员工盘货品的方式抽离出来，进行解耦。小A和小B其实就是访问者。而裤子和衣服就是元素。仓库类是某种据结构。

代码：

抽象元素类：接收访问者，来访问自己

```
public interface Element {
    // 接收访问者
    void accept(Visitor visitor);
}
```

具体元素类：衣服和裤子

```
public class ShirtElement implements Element {

    @Override
    public void accept(Visitor visitor) {
        visitor.visite(this);
    }

}

public class PantsElement implements Element {

    @Override
    public void accept(Visitor visitor) {
        visitor.visite(this);
    }

}
```

抽象访问者：定义了盘点衣服和裤子的方法

```
public interface Visitor {
```

```
    void visite(ShirtElement element);

    void visite(PantsElement element);

}
```

具体访问者：衣服盘点类和裤子盘点类

```
public class ShirtVisitor implements Visitor {

    @Override
    public void visite(ShirtElement element) {
        System.out.println("盘点衣服！");
    }

    @Override
    public void visite(PantsElement element) {
        System.out.println("这是裤子，不盘点！");
    }
}

public class PantsVisitor implements Visitor {

    @Override
    public void visite(ShirtElement element) {
        System.out.println("这是衣服，不盘点！");
    }

    @Override
    public void visite(PantsElement element) {
        System.out.println("盘点裤子！");
    }
}
```

数据结构类：服装仓库，接收盘点者，然后依次让元素接收盘点者。

```
public class WareHouse {

    private List<Element> clothes = new ArrayList();

    public void add(Element element) {
        clothes.add(element);
    }

    public void accept(Visitor visitor) {

        for (Element clothe : clothes) {
            clothe.accept(visitor);
        }

    }

}
```

测试访问者模式

```
public class TestVisitor {

    public static void main(String[] args) {

        WareHouse wareHouse = new WareHouse();
        // 加入三件衣服 一件裤子
        wareHouse.add(new ShirtElement());
        wareHouse.add(new ShirtElement());
        wareHouse.add(new PantsElement());
        wareHouse.add(new ShirtElement());
        // 盘点衣服
        wareHouse.accept(new ShirtVisitor());
        System.out.println("========================");
        // 盘点裤子
        wareHouse.accept(new PantsVisitor());

    }

}
```

运行结果：

```
盘点衣服!
盘点衣服!
这是裤子，不盘点!
盘点衣服!
========================
这是衣服，不盘点!
这是衣服，不盘点!
盘点裤子!
这是衣服，不盘点!
```

解析：

1. 扩展性好。能够在不修改对象结构中的元素的情况下，为对象结构中的元素添加新的功能。

2. 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。

3. 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影响系统的数据结构。

4. 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都比较单一。