

# Software design pattern

## Visitor pattern

Student Name: Yuan Wang

Student ID: A00258427



## Abstract

Software design patterns, also known as design patterns, are solutions to recurring problems in object-oriented design. Design patterns are used to reuse code, making the code easier to understand, ensuring code reliability and reusable programs. The visitor pattern is one of the software design patterns. This paper will focus on the visitor pattern, including an introduction to the visitor pattern, how to use the visitor pattern (programming with the visitor pattern), and the main advantages and disadvantages of this pattern.

## 1. Background and introduction

In software design, laziness is a virtue: it better to reuse than to redo. Design patterns are a good illustration. Patterns, a major advance in software architecture, provide a common vocabulary and a widely known catalog of design solutions addressing frequently encountered situations. [1]

Software design is the software design pattern is a very mature, can be used repeatedly, most people know, classified and cataloged, code design experience summary, which is widely used in software development, to ensure the reliability of the software code, and guarantee reusable the program. In other words, software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation. When we introduce the software design pattern, the first thing to talk about is the four elements of software design, which will help us better understand the software design pattern:

- ❖ The first one is the name of the model. A good schema name can help us think about it. The software design pattern that exists today has its name repeated to make it closer to the pattern itself.
- ❖ The second is the problem. Basically, this element mainly emphasizes the cause and effect of the problem, and may describe a specific design problem, so that we can find or create a solution that is more in line with the problem.
- ❖ The third part is the main factor of the software design pattern, which is the solution to the problem. It mainly describes the components of the design, the relationship between them and their respective responsibilities and ways of cooperation. Because a pattern is like a template that can be applied to many difference situations, a design pattern is not just a solution to a single problem. It is an abstract description that can be used to solve this type or problem.
- ❖ The fourth part is to solve the effect. This part is an evaluation of a software design pattern, so the effect of the model includes its influence on the flexibility

or portability of the system. These evaluations will be used when we choose to use the software design pattern. Is a good reference.

For the software design pattern, as well as other design patterns and development models, there are basic norms and principles. Basically, the software design pattern is proposed to improve the reusability of programming code, so one of the basic principles is to open Closed principles[7], other principles are like interface isolation principles, synthetic reuse principles, abstract classes, and so on. The principle of opening and closing is proposed by "Bertrand Meyer" is that "Software entities should be open for extension but closed for modification" [7]. It means that the module should be extended as much as possible without modifying the original (original, referring to the original code) code.

In software design pattern, basically, have 23 different design patterns and can be divided into three types, namely, the creation mode, the structure mode, and the behavioral model. One of the most controversies discussed patterns is the VISITOR pattern. [2] Visitor pattern belongs to behavior mode type and is a very important part in this type. The visitor pattern is a software design pattern used to separates the algorithm form the object structure. In other words, it's can separate the data structure and data operations. It can be seen from the definition that the structural object is a prerequisite for using the visitor pattern, and this structural object must have a method of traversing its own individual objects. This is similar to the collection concept in the Java language.

In the software construction process, due to changes in requirements, it is often necessary to add new behaviors (methods) in some class hierarchies. If such changes are made directly in the base class, it will bring a heavy burden of change to the sub-classes. And even destroy the original design. The visitor pattern main focus on solving this problem. So a basic assumption of the Visitor pattern is that one knows the classes of all objects to be visited. [3]When the class structure changes, the visitors must be rewritten. For example, if one is writing a grammar and is using JTB or JJTree together with JavaCC, then changes to the grammar imply that the visitors must change as well. [3]This is the case even when the changes are in a part of the grammar which is of no direct interest to a particular visitor. [3] The visitor pattern can be said to be the most complicated mode of behavioral mode. Visitor patter basically have five different objects or classes:

- ❖ The first one is Visitor (Abstract visitor) that can declares a Visit operation for each class of Concrete Element in the object structure. [8] From the name or parameter type of the operation, we can clearly get the type of the specific element. And the specific visitor needs to implement these operation methods and define the access operations on these elements.
- ❖ Second one is Concrete Visitor (vacation visitors). This visitor is primarily

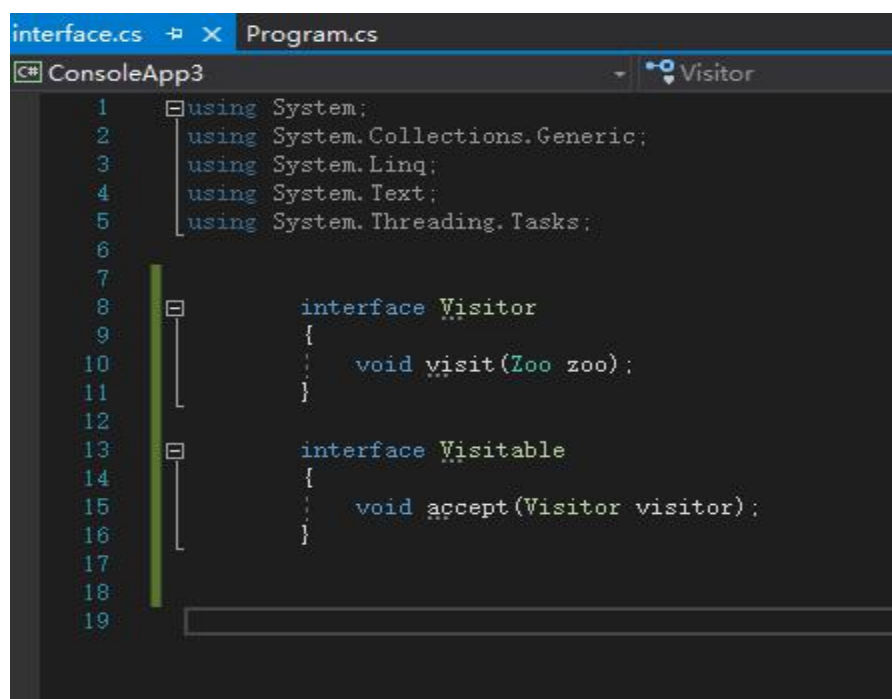
intended to implement each operation declared by an abstract visitor, each operation being used to access a type of element in the object structure.

- ❖ Third one is Element (abstract element) [8]. generally, this character is an abstract class or interface that defines an accept() method that takes an abstract visitor as a parameter.
- ❖ Forth is Concrete Element (specific element) [8]. This concrete element implements the accept() method, which invokes the access method of the visitor in the accept() method to complete the operation on an element.
- ❖ Fifth is Object Structure. The object structure[8] is a collection of elements that hold an element object and provides a way to traverse its internal elements. It can also be a simple collection object, such as a List object or a Set object.

The visitor pattern provides a way of working on the elements of an object structure, allowing us to define new operations on the elements without changing the structure of the elements. In other words, if the data structure of the system is relatively stable, but its operations (algorithms) are easy to change, then using the visitor mode is a good choice.

## 2. Description

### 2.1 Example



```
interface.cs  Program.cs
C# ConsoleApp3  Visitor
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7
8  interface Visitor
9  {
10     void visit(Zoo zoo);
11 }
12
13 interface Visitable
14 {
15     void accept(Visitor visitor);
16 }
17
18
19
```

(Figure1:Visitor Interface)

```
interface.cs  Program.cs  X
ConsoleApp3
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7
8
9
10 class Zoo:Visitable
11 {
12     private String Animal;
13     public Zoo(string animal)
14     {
15         this.Animal = animal;
16     }
17     public void accept(Visitor visitor)
18     {
19         visitor.visit(this);
20     }
21     public void printAnimalName()
22     {
23         Console.WriteLine("The name of this animal is " + Animal);
24     }
25     public string getAnimalName()
26     {
27         return Animal;
28     }
29     public void Hungry()
30     {
31         Console.WriteLine(Animal + " is hungry!!!");
32     }
33 }
34
```

(Figure2:Zoo class)

```
33
34
35 class FeedFunction : Visitor
36 {
37     Zoo extzoo;
38     public void visit(Zoo zoo)
39     {
40         this.extzoo = zoo;
41     }
42     public void feedwithMeat()
43     {
44         String name = extzoo.getAnimalName();
45         Console.WriteLine("we feeding meat to " + name);
46         Console.WriteLine(name + "is so happy");
47     }
48     public void feedwithGrass()
49     {
50         String name = extzoo.getAnimalName();
51         Console.WriteLine("we feeding grass to " + name);
52         Console.WriteLine(name + "is so happy");
53     }
54 }
55
```

(Figure3:Feed function)

```

public class VisitorPattern
{
    public static void Main(string[] args)
    {
        String animal1 = "lion";
        String animal2 = "sheep";
        Zoo z = new Zoo(animal1);
        z.printAnimalName();
        z.Hungry();
        FeedFunction ff = new FeedFunction();
        z.accept(ff);
        ff.feedwithMeat();
        Console.WriteLine("*****");
        Zoo zs = new Zoo(animal2);
        zs.printAnimalName();
        zs.Hungry();
        FeedFunction ffs = new FeedFunction();
        zs.accept(ffs);
        ffs.feedwithGrass();
    }
}

```

(Figure4:Main function)

```

Program.cs
C:\Windows\system32\cmd.exe
The name of this animal is lion
lion is hungry!!!
we feeding meat to lion
lion is so happy
*****
The name of this animal is sheep
sheep is hungry!!!
we feeding grass to sheep
sheep is so happy
请按任意键继续. . .

```

(Figure5: Result )

## 2.2 Explain

This is a simple example to explain the visitor pattern. I created a zoo class. There are many animals in the zoo. I have to feed these animals. Then I need to call different methods, but in the zoo class, I only implement Obtaining the animal's name and obtaining several ways in which the animal is hungry does not achieve a method of feeding the animal.

Because it is implemented using the visitor pattern, the first thing I need to do is that create a Visitor interface like figure1. In this interface class, I create two interfaces first one is called “ Visitor ” which is the abstract visitor character and inside had created a visit method which is used to get the Concrete Visitor, the second interface is called “visitable” and inside create an accept method which used to get the visitor object.

The next step is to create the zoo class, as shown in Figure 2. In the Zoo class, we need to implement this "accessible" interface, because the visitor object can only be obtained when we implement the accept method in this interface class. I create a constructor that takes animal variables. After that, three methods PrintAnimalName(), getAnimalName(), and Hungry() are created to output the animal's name on the screen, get the animal's name, and get the animal's hungry state.

As shown in Figure 3, after we have created the zoo class and implemented the accept method in the “visitable” interface, we can create the feed class to implement the method of feeding the animal. This class needs to implement the visitor method which in the Visitor interface. And we need to declare the Concrete Visitor first. Next to implement the visit() method. Then create two feed method, feedwithMeat() and feedwithGrass(). In both of these methods, we need to call the method of getAnimalName() in the previous zoo class. That's why we need to declare the zoo object at the beginning of this class. In other words, in this way we can get our Concrete Visitor object and call it's methods. This is also the most essential thing in the visitor pattern.

Finally as shown in figure 4, we can test visitor pattern, create two animals lion and sheep, then declare zoo object to call zoo method, output animal's name and animal's hungry state. Next we want call the feed method in feed class so we create feed class object. We need to call the accept() method and send this object, to tell program this is a visitor and we want to use the visitor's method. For the first animal lion, we want feed meat so we call method feedwithMeat(), for the second animal sheep we want feed grass so we call method feedwithGrass(), Then run this program and can get result like figure5.

## 3. Summary and Conclusion

### 3.1 Summary

In summary, we can find a good explanation of the visitor pattern programming in the example. The visitor mode makes the program more simple to add functions and makes the structure of the program clearer and simpler. As a software design pattern,

it is particularly suitable for developing stable data structures program. For example, in the code example, because the animal class, not just a single animal, can be a Lion, can be a sheep, in the case where the visitor mode is not applicable, the program will not be correctly recognized, resulting in a compilation error. So when we call the method, we pass the parameters and let it act as a visitor role to make it more convenient after the specific element.

If an object structure is complex and the structure is stable and not easy to change, but you need to define new operations on this structure frequently, it is very suitable to use the visitor pattern, such as complex collection objects etc.

### 3.2 Conclusion

As a software design pattern, the visitor mode is very good. It has good advantages, such as adding new access operations is very convenient, in line with the principle of opening and closing[7]; the access behavior of related element objects is concentrated into one in the visitor object, rather than being scattered among the individual element classes, the responsibilities of the class are clearer and conform to the single responsibility principle.[4]

At the same time, however, there are some shortcomings and limitations in the visitor pattern. For example, it is very difficult to add new element classes. element objects sometimes must Exposing some of its own internal operations and state, otherwise it is not accessible to visitors, which undermines the encapsulation of elements. Moreover, the visitor mode also requires that the program data structure designed using this mode must be stable. If the data structure is easy to change, the visitor mode is not suitable.

Through this practice of the visitor pattern, I have a deeper understanding of the software design pattern. At the same time, this exercise also gave me a deeper and better understanding of the C# development of using Visual Studio and the high-level language of C#.

## 4. Reference

- [1] Martin, R. (2001). The visitor pattern. Estados Unidos: R y D Publications.
- [2] Anon,(2018).[online]Availableat:[https://www.researchgate.net/publication/221389779\\_Digging\\_into\\_the\\_Visitor\\_Pattern](https://www.researchgate.net/publication/221389779_Digging_into_the_Visitor_Pattern) [Accessed 14 Nov. 2018].
- [3] Web.cs.ucla.edu. (2018). [online] Available at: <http://web.cs.ucla.edu/~palsberg/paper/compsac98.pdf> [Accessed 14 Nov. 2018].
- [4] Houben, R. and Vandenbruwaene, W. (n.d.). *The single resolution mechanism*.
- [5] Rouson, D., Xia, J. and Xu, X. (2011). *Scientific software design*. Cambridge: Cambridge University



Press.

- [6] www.tutorialspoint.com. (2018). *Software Design Basics*. [online] Available at: [https://www.tutorialspoint.com/software\\_engineering/software\\_design\\_basics.htm](https://www.tutorialspoint.com/software_engineering/software_design_basics.htm) [Accessed 28 Nov. 2018].
- [7] JoelAbrahamsson.com. (2018). *A simple example of the Open/Closed Principle*. [online] Available at: <http://joelabrahamsson.com/a-simple-example-of-the-enclosed-principle/> [Accessed 28 Nov. 2018].
- [8] www.tutorialspoint.com. (2018). *Design Patterns Visitor Pattern*. [online] Available at: [https://www.tutorialspoint.com/design\\_pattern/visitor\\_pattern.htm](https://www.tutorialspoint.com/design_pattern/visitor_pattern.htm) [Accessed 28 Nov. 2018].