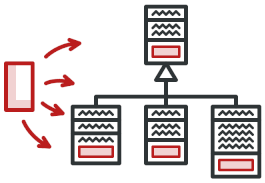

Посетитель на Go

Посетитель — это поведенческий паттерн, который позволяет добавить новую операцию для целой иерархии классов, не изменяя код этих классов.

Подробней о том, почему Посетитель нельзя заменить простой перегрузкой методов читайте в статье **Посетитель и Double Dispatch**.

Подробней о паттерне Посетитель →

Навигация

Интро

Концептуальный пример
shape
square
circle
rectangle
visitor
areaCalculator
middleCoordinates
main
output

Концептуальный пример

Паттерн Посетитель позволяет вам добавлять поведение в структуру без ее изменения. Представим, что вы разработчик библиотеки, которая содержит структуры разных фигур:

- Квадрат
- Круг
- Треугольник

Структуры каждой из вышеназванных фигур реализуют общий интерфейс фигуры.

Как только сотрудники в вашей компании начали использовать вашу замечательную библиотеку, вас засыпали просьбами добавить тот или иной функционал. Рассмотрим один из простейших вариантов: команда попросила вас добавить в структуру функцию `getArea`, возвращающую площадь фигуры.

Существует много решений этой проблемы.

Первое, что приходит в голову – добавить метод `getArea` напрямую в интерфейс фигуры, и затем реализовать его в каждой структуре. Это кажется самым правильным решением, но у него есть свои минусы. Как разработчик библиотеки, вы рискуете сломать ваш драгоценный

код каждый раз, когда добавляете новое поведение. Несмотря на это, вы хотите дать другим командам возможность каким-то образом расширять вашу библиотеку.

Второй вариант – команда, которая запрашивает добавление новой функции, может сама реализовать ее в своем проекте. Однако, это не всегда является возможным, так как новый функционал может полагаться на скрытый код.

Третий возможный вариант — решить вышеуказанную проблему благодаря использованию паттерна Посетитель. Сперва мы определяем интерфейс посетителя следующим способом:

```
type visitor interface {
   visitForSquare(square)
   visitForCircle(circle)
   visitForTriangle(triangle)
}
```

Функции `visitForSquare(square)`, `visitForCircle(circle)`, `visitForTriangle(triangle)` позволят нам добавлять функционал для квадратов, кругов и треугольников соответственно.

Не понимаете, почему мы не можем оставить только один метод `visit(shape)` в интерфейсе посетителя? Это невозможно из-за того, что язык Go не поддерживает перегрузку методов, поэтому вы не можете иметь методы с одинаковыми именами, но разными параметрами.

Второй, не менее важный, этап – добавление метода `accept` в интерфейс фигуры.

```
func accept(v visitor)
```

Все структуры фигур должны определять этот метод похожим способом:

```
func (obj *square) accept(v visitor){
    v.visitForSquare(obj)
}
```

Погодите, разве я не заявлял чуть ранее, что не хочу менять существующие структуры фигур? К сожалению, во время использования паттерна Посетителя нам придется вносить изменения в структуры, но лишь единожды.

В случае добавления другого функционала, например `getNumSides` или `getMiddleCoordinates`, мы будем использовать все тот же метод `accept(v visitor)` без новых изменений структур фигур.

В конечном итоге, структуры нужно изменить лишь единожды, и все будущие запросы нового функционала можно будет реализовать с помощью функции `accept(v visitor)`. Если команда запросит поведение `getArea`, мы можем просто определить явную реализацию интерфейса посетителя и прописать логику вычисления площади в этой конкретной имплементации.

shape.go: Элемент

```
package main

type Shape interface {
        getType() string
        accept(Visitor)
}
```

square.go: Конкретный элемент

```
package main

type Square struct {
        side int
}

func (s *Square) accept(v Visitor) {
        v.visitForSquare(s)
}

func (s *Square) getType() string {
        return "Square"
}
```

circle.go: Конкретный элемент

```
package main

type Circle struct {
        radius int
}

func (c *Circle) accept(v Visitor) {
        v.visitForCircle(c)
}

func (c *Circle) getType() string {
        return "Circle"
}
```

rectangle.go: Конкретный элемент

```
package main

type Rectangle struct {
        l int
        b int
}

func (t *Rectangle) accept(v Visitor) {
        v.visitForrectangle(t)
}

func (t *Rectangle) getType() string {
        return "rectangle"
}
```

visitor.go: Посетитель

```
package main

type Visitor interface {
        visitForSquare(*Square)
        visitForCircle(*Circle)
        visitForrectangle(*Rectangle)
}
```

areaCalculator.go: Конкретный посетитель

```
package main

import (
        "fmt"
)

type AreaCalculator struct {
        area int
}

func (a *AreaCalculator) visitForSquare(s *Square) {
        // Calculate area for square.
        // Then assign in to the area instance variable.
        fmt.Println("Calculating area for square")
}

func (a *AreaCalculator) visitForCircle(s *Circle) {
        fmt.Println("Calculating area for circle")
}
func (a *AreaCalculator) visitForrectangle(s *Rectangle) {
        fmt.Println("Calculating area for rectangle")
}
```

middleCoordinates.go: Конкретный посетитель

```
package main

import "fmt"

type MiddleCoordinates struct {
```

```
        x int
        y int
}

func (a *MiddleCoordinates) visitForSquare(s *Square) {
        // Calculate middle point coordinates for square.
        // Then assign in to the x and y instance variable.
        fmt.Println("Calculating middle point coordinates for square")
}

func (a *MiddleCoordinates) visitForCircle(c *Circle) {
        fmt.Println("Calculating middle point coordinates for circle")
}
func (a *MiddleCoordinates) visitForrectangle(t *Rectangle) {
        fmt.Println("Calculating middle point coordinates for rectangle")
}
```

main.go: Клиентский код

```
package main

import "fmt"

func main() {
        square := &Square{side: 2}
        circle := &Circle{radius: 3}
        rectangle := &Rectangle{l: 2, b: 3}

        areaCalculator := &AreaCalculator{}

        square.accept(areaCalculator)
        circle.accept(areaCalculator)
        rectangle.accept(areaCalculator)

        fmt.Println()
        middleCoordinates := &MiddleCoordinates{}
        square.accept(middleCoordinates)
        circle.accept(middleCoordinates)
        rectangle.accept(middleCoordinates)
}
```

output.txt: Результат выполнения

```
Calculating area for square
Calculating area for circle
Calculating area for rectangle

Calculating middle point coordinates for square
Calculating middle point coordinates for circle
Calculating middle point coordinates for rectangle
```

По материалам: ***Golang By Example***

ВЕРНУТЬСЯ НАЗАД

← Шаблонный метод на Go

ЧИТАЕМ ДАЛЬШЕ

Паттерны проектирования на Java →

Посетитель на других языках программирования