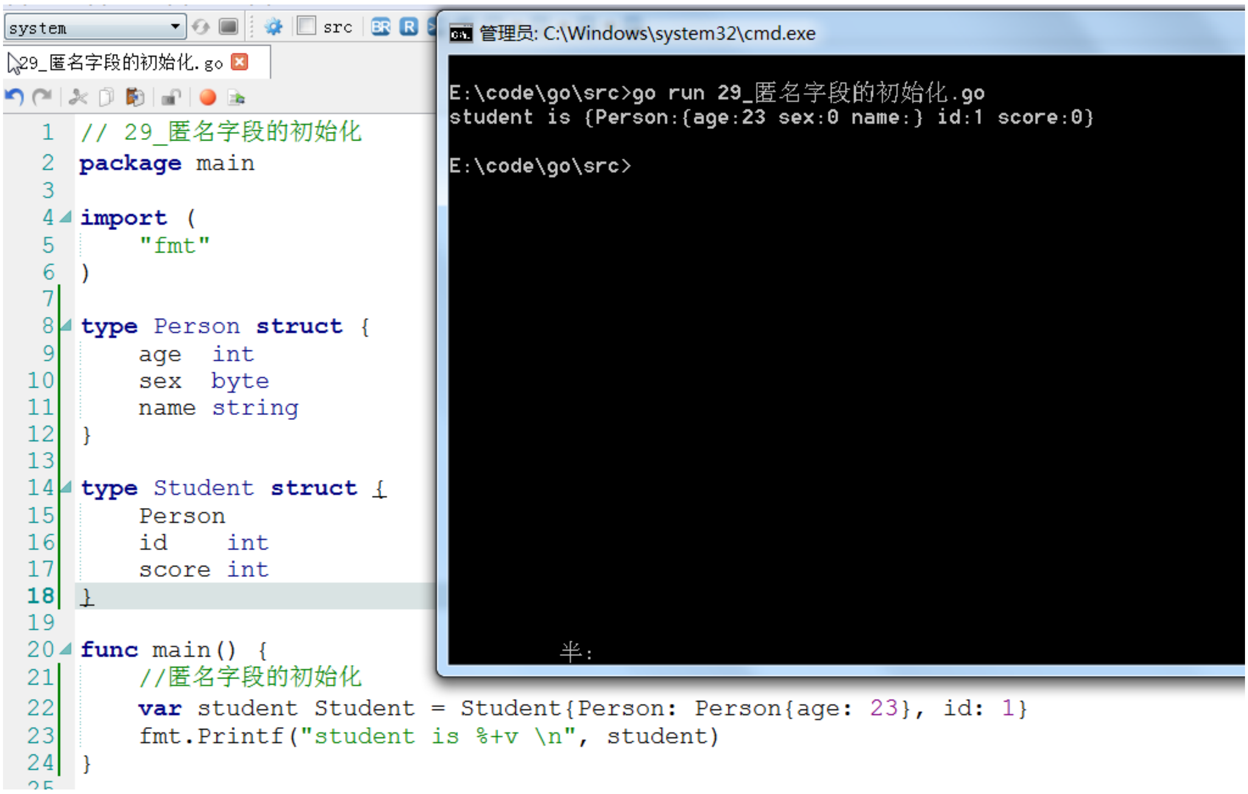
面向对象编程

1.匿名字段的作用

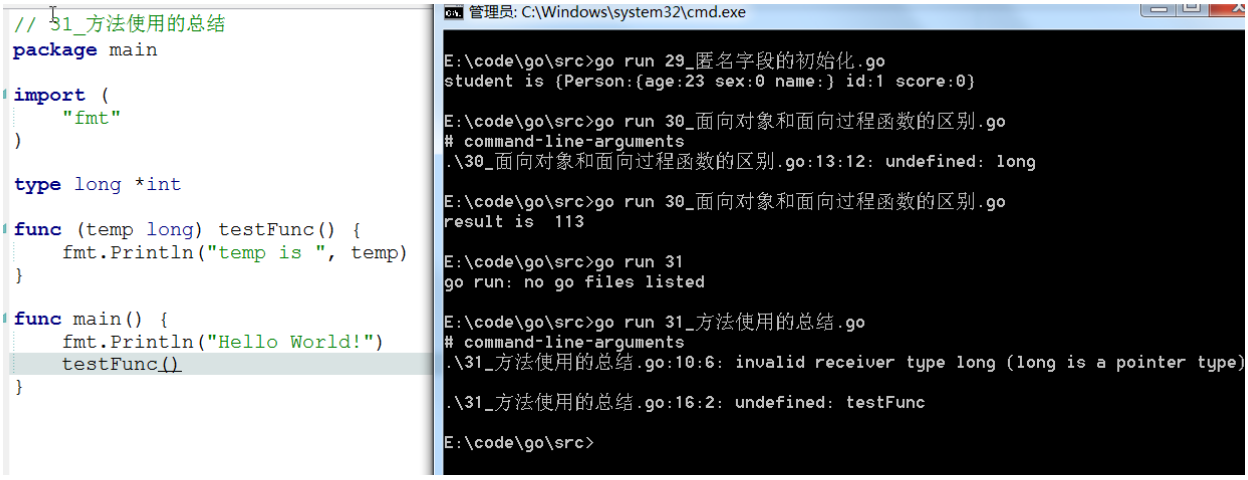


```go
// 29_匿名字段的初始化
package main

import (
	"fmt"
)

type Person struct {
	age  int
	sex  byte
	name string
}

type Student struct {
	Person
	id    int
	score int
}

func main() {
	//匿名字段的初始化
	var student Student = Student{Person: Person{age: 23}, id: 1}
	fmt.Printf("student is %+v \n", student)
}
```

```
E:\code\go\src>go run 29_匿名字段的初始化.go
student is {Person:{age:23 sex:0 name:} id:1 score:0}

E:\code\go\src>
```

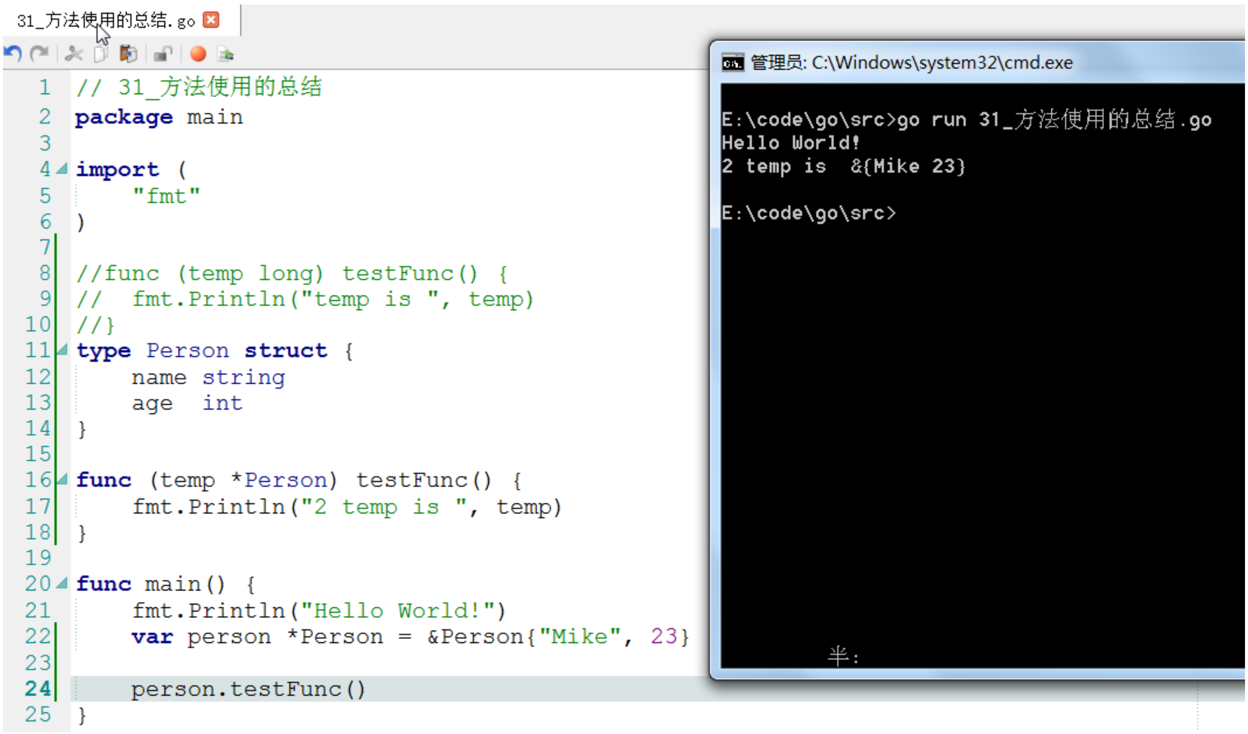
2.面向过程和面向对象函数的区别

3.带有接收者的函数叫做方法



```go
// 31_方法使用的总结
package main

import (
	"fmt"
)

type long *int

func (temp long) testFunc() {
	fmt.Println("temp is ", temp)
}

func main() {
	fmt.Println("Hello World!")
	testFunc()
}
```

```
E:\code\go\src>go run 29_匿名字段的初始化.go
student is {Person:{age:23 sex:0 name:} id:1 score:0}

E:\code\go\src>go run 30_面向对象和面向过程函数的区别.go
# command-line-arguments
.\30_面向对象和面向过程函数的区别.go:13:12: undefined: long

E:\code\go\src>go run 30_面向对象和面向过程函数的区别.go
result is  113

E:\code\go\src>go run 31
go run: no go files listed

E:\code\go\src>go run 31_方法使用的总结.go
# command-line-arguments
.\31_方法使用的总结.go:10:6: invalid receiver type long (long is a pointer type)

.\31_方法使用的总结.go:16:2: undefined: testFunc

E:\code\go\src>
```



```go
// 31_方法使用的总结
package main

import (
	"fmt"
)

//func (temp long) testFunc() {
//	fmt.Println("temp is ", temp)
//}
type Person struct {
	name string
	age  int
}

func (temp *Person) testFunc() {
	fmt.Println("2 temp is ", temp)
}

func main() {
	fmt.Println("Hello World!")
	var person *Person = &Person{"Mike", 23}

	person.testFunc()
}
```

```
E:\code\go\src>go run 31_方法使用的总结.go
Hello World!
2 temp is  &{Mike 23}

E:\code\go\src>
```

4.方法的继承

```go
type Person struct {
    age  int
    name string
    sex  byte
}

func (p *Person) printPerson() {
    fmt.Printf("person is %+v\n", p)
}

type Student struct {
    Person
    id    int
    score int
}

func main() {
    //验证方法的继承
    var student *Student = &Student{Person{23, "Mike", 'm'}, 666, 87}
    student.printPerson()
}
```

5.方法的重写

6.接口

不关心类型,只关心行为

接口名通常以er结尾

7.定义空接口切片

```
func main() {
    slice := make([]interface{}, 3)
    slice[0] = 1
    slice[1] = "haha"
    slice[2] = Student{"Mike", 90}

    for i, data := range slice {
        switch value := data.(type) { //得到对应的切片元素的类型
        case int:
            fmt.Printf("slice[%d] i is int,type is %v\n", i, value)
            break
        case string:
            fmt.Printf("slice[%d] i is string,type is %v\n", i, value)
            break
        case Student:
            fmt.Printf("slice[%d] i is Student,type is %v\n", i, value)
            break
        }
    }
}
```

8.超集可以转换成子集,反之不可以