Day4 指针类型和面向对象

tony

目录

变量和内存地址

1. 每个变量都有内存地址，可以说通过变量来操作对应大小的内存

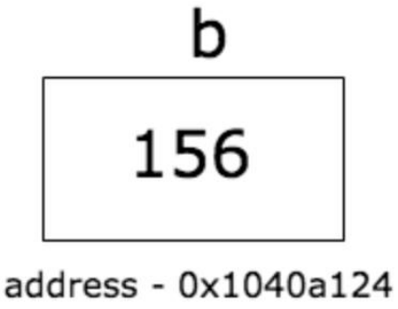
```
var a int32
a = 100
fmt.Printf(“%d\n”, a)
fmt.Printf(“%p\n”, &a)
```

注意：通过&符号可以获取变量的地址

变量和内存地址

2. 普通变量存储的是对应类型的值，这些类型就叫**值类型**

```
var b int32
b = 156
fmt.Printf(“%d\n”, b)
fmt.Printf(“%p\n”, &b)
```



指针类型

3. 指针类型的变量存储的是一个地址，所以又叫**指针类型**或**引用类型**


```
var b int32
b = 156
var a *int32
a = &b
```

指针类型

4. 指针类型定义，var 变量名 *类型

```
package main

import (
    "fmt"
)

func main() {
    b := 255
    var a *int = &b
    fmt.Printf("Type of a is %T\n", a)
    fmt.Println("address of b is", a)
}
```

指针类型

5. 指针类型变量的默认值为nil，也就是空地址

```
package main

import (
    "fmt"
)

func main() {
    a := 25
    var b *int
    if b == nil {
        fmt.Println("b is", b)
        b = &a
        fmt.Println("b after initialization is", b)
    }
}
```

指针类型

6. 如果操作指针变量指向的地址里面的值呢?

```
package main
import (
    "fmt"
)

func main() {
    b := 255
    a := &b
    fmt.Println("address of b is", a)
    fmt.Println("value of b is", *a)
}
```

注意：通过* 符号可以获取指针变量指向的变量

指针类型

7. 通过指针修改变量的值

```
package main
import (
    "fmt"
)

func main() {
    b := 255
    a := &b
    fmt.Println("address of b is", a)
    fmt.Println("value of b is", *a)
    *a++
    fmt.Println("new value of b is", b)
}
```

指针类型

8. 指针变量传参

```
package main
import (
    "fmt"
)
func change(val *int) {
    *val = 55
}
func main() {
    a := 58
    fmt.Println("value of a before function call is",a)
    b := &a
    change(b)
    fmt.Println("value of a after function call is", a)
}
```

指针类型

9. 指针变量传参示例2

```go
package main

import (
    "fmt"
)

func modify(arr *[3]int) {
    (*arr)[0] = 90
}

func main() {
    a := [3]int{89, 90, 91}
    modify(&a)
    fmt.Println(a)
}
```

指针类型

10.切片传参

```go
package main

import (
    "fmt"
)

func modify(sls []int) {
    sls[0] = 90
}

func main() {
    a := [3]int{89, 90, 91}
    modify(a[:])
    fmt.Println(a)
}
```

注意：切片是引用类型！！

指针类型

11. make用来分配引用类型的内存，比如 map、slice以及channel

new用来分配除引用类型的所有其他类型的内存，比如 int、数组等

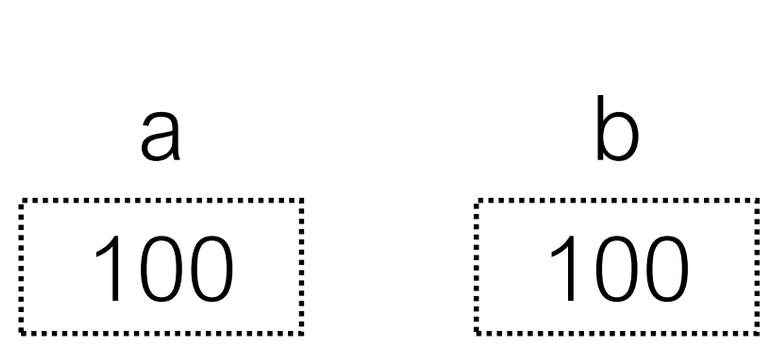
值拷贝和引用拷贝

12. 值拷贝和引用拷贝

```
package main


import (
    "fmt"
)

func main() {
    var a int = 100
    b := a
}
```



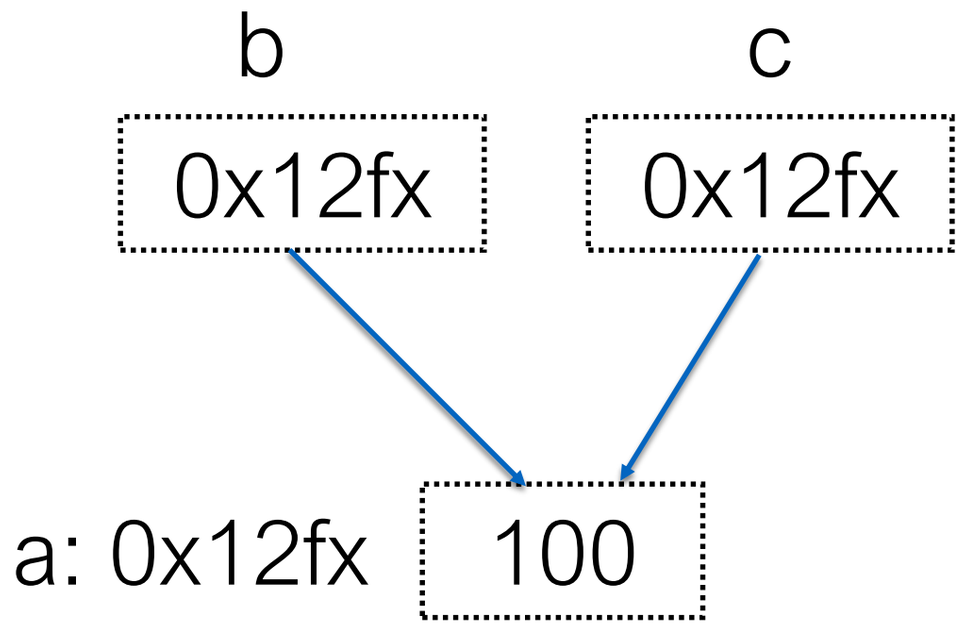
值拷贝和引用拷贝

13. 值拷贝和引用拷贝

```go
package main

import (
    "fmt"
)

func main() {
    var a int = 100
    var b *int = &a
    var c *int = b
    *c = 200
}
```



课后练习

1. 写一个程序，获取一个变量的地址，并打印到终端

2. 写一个函数，传入一个int类型的指针，并在函数中修改所指向的值

目录

方法的定义

1. 和其他语言不一样，Go的方法采用另外一种方式实现。

方法的定义

2. Go的方法是在函数前面加上一个接受者，这样编译器就知道这个方法属于哪个类型了

```
type A struct {

}

func (a A) Test(s string) {

}
```

Test的接受者，因此A这个对象有一个Test方法

通过a来访问A的实例中的成员变量，也就是struct中的字段

方法的定义

3. 可以为当前包内定义的任何类型增加方法

```
type int Integer    //Integer是int的别名

func (a Integer) Test(s string) {

}
```

通过a来访问Integer的实例中的成员变量，也就是int

Test的接受者，因此Integer这个对象有一个Test方法

函数和方法的区别

4. 函数不属于任何类型，方法属于特定的类型

值类型和指针类型

5. 指针类型作为接受者

```
type A struct {

}

func (a *A) Test(s string) {

}
```

Test的接受者，因此A这个对象有一个Test方法

通过a来访问A的实例中的成员变量，也就是struct中的字段

值类型和指针类型

6. 指针类型和值类型作为接受者的区别

值类型和指针类型

7. 什么时候用值类型/指针类型作为接受者？

A. 需要修改接受者中的值的时候

B. 接受者是大对象的时候，副本拷贝代价比较大

C. 一般来说，通常使用指针类型作为接受者

匿名字段与继承

8. 匿名结构体与继承

```
type Animal struct {
    Name        string
}
```

```
type People struct {
    Sex       string
    Age       int
    Animal    //or *Animal
}
```

匿名字段与继承

9. 多重继承与冲突解决

```
type Mother struct {
    Name        string
}
```

```
type Father struct {
    Name        string
}
```

```
type People struct {
    Sex       string
    Age       int
    *Mother
    *Father
}
```

结构体与json序列化

10.结构体序列化：结构体转成json数据格式

```
type People struct {
    Sex       string
    Age       int
    *Mother
    *Father
}
```

结构体与json序列化

11.结构体反序列化：json数据格式转成结构体

课后练习

1. 实现一个图书管理系统，具有以下功能：

   a. 书籍录入功能，书籍信息包括书名、副本数、作者、出版日期

   b. 书籍查询功能，按照书名、作者、出版日期等条件检索

   c. 学生信息管理功能，管理每个学生的姓名、年级、身份证、性别、借了什么书等信息

   d. 借书功能，学生可以查询想要的书籍，进行借出

   e. 书籍管理功能，可以看到每种书被哪些人借出了

QA