函数和map数据类型

tony

Outline

1. 指针类型、函数、递归函数、闭包

3. map数据结构

4. package介绍

5. 课后作业

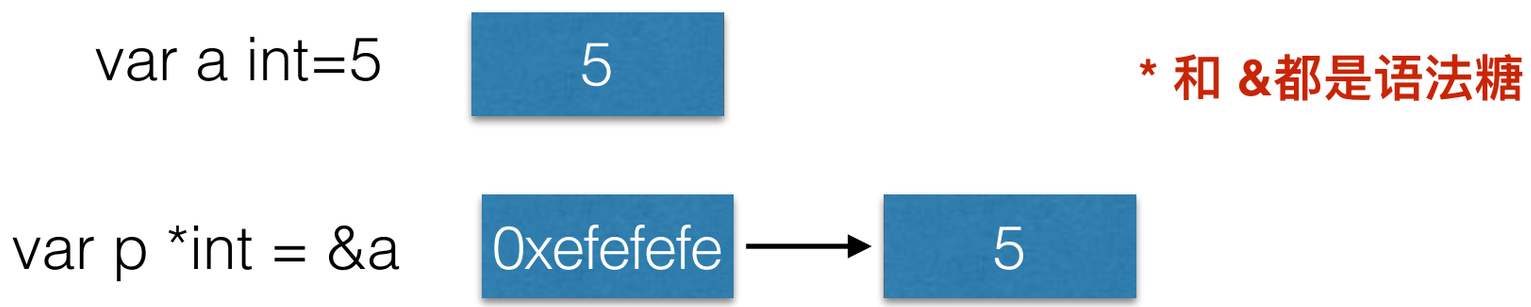
指针类型

1. 普通类型，变量存的就是值，也叫值类型。指针类型存的是地址

2. 获取变量的地址，用&，比如： var a int, 获取a的地址：&a

3. 指针类型，变量存的是一个地址，这个地址存的才是值

4. 获取指针类型所指向的值，使用：*，比如：var *p int, 使用*p获取p指向的值



指针类型

练习1：写一个程序，获取一个变量的地址，并打印到终端。

练习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。

内置函数

1. close：主要用来关闭channel

2. len：用来求长度，比如string、array、slice、map、channel

3. new：用来分配内存，主要用来分配值类型，比如int、struct。返回的是指针

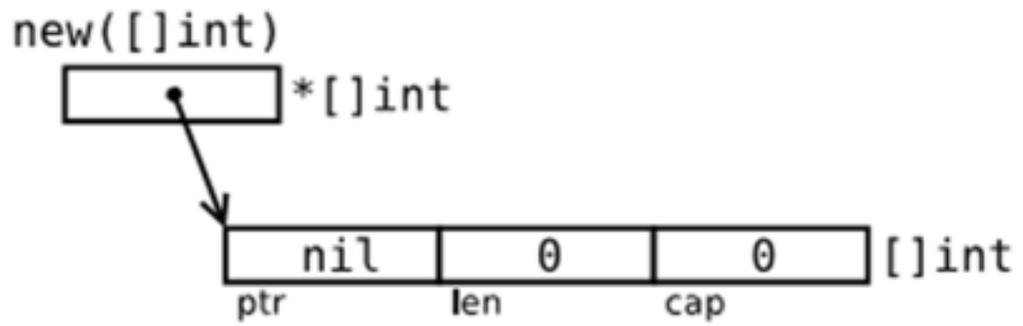
4. make：用来分配内存，主要用来分配引用类型，比如chan、map、slice

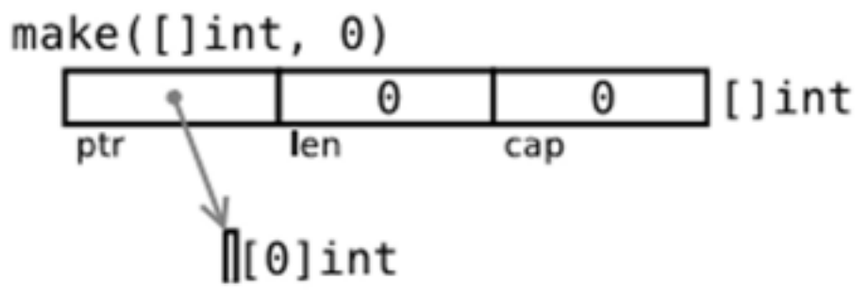
5. append：用来追加元素到数组、slice中

6. panic和recover：用来做错误处理

内置函数

7. new和make的区别





函数

1. 声明语法：func 函数名 (参数列表) [(返回值列表)] {}

```
func add() {
}
```

```
func add(a int, b int) {
}
```

```
func add(a int, b int) int {
}
```

```
func add(a int, b int) (int, int)  {
}
```

```
func add(a, b int) (int, int)  {
}
```

函数

2. 错误的写法

```
func add()
{

}
```

函数

2. golang函数特点：

   a. 不支持重载，一个包不能有两个名字一样的函数

   b. 函数是一等公民，函数也是一种类型，一个函数可以赋值给变量

   c. 匿名函数

   d. 多返回值

函数

2. golang函数特点：

```
package main

import "fmt"

func add(a, b int) int {
	return a + b
}

func main() {

	//函数是一等公民，也可以和变量一样进行赋值
	c := add
	fmt.Printf("%p %T %p %T\n", c, add, c, add)

	sum := c(10, 20)
	fmt.Println(sum)

	sum = add(10, 20)
	fmt.Println(sum)
}
```

函数

2. golang函数特点：

```
package main

import "fmt"

type add_func func(int, int) int

func add(a, b int) int {
	return a + b
}

func operator(op add_func, a int, b int) int {
	//使用传进来的函数，进行操作
	return op(a, b)
}

func main() {
	c := add
	fmt.Println(c)
	sum := operator(c, 100, 200)
	fmt.Println(sum)
}
```

函数

2. golang函数特点：

```
package main

import "fmt"

type add_func func(int, int) int

func add(a, b, c int) int {
        return a + b+c
}

func operator(op add_func, a int, b int) int {
        return op(a, b)
}

func main() {
        c := add
        fmt.Println(c)
        sum := operator(c, 100, 200)
        fmt.Println(sum)
}
```

函数

2. golang函数特点：

```
package main

import "fmt"

type add_func func(int, int) int

func sub(a, b int) int {
        return a - b
}

func operator(op add_func, a int, b int) int {
        return op(a, b)
}

func main() {
        c := sub
        fmt.Println(c)
        sum := operator(c, 100, 200)
        fmt.Println(sum)
}
```

函数

3. 函数参数传递方式：

1). 值传递

2). 引用传递

注意1：无论是值传递，还是引用传递，传递给函数的都是变量的副本，不过，值传递是值的拷贝。引用传递是地址的拷贝，一般来说，地址拷贝更为高效。而值拷贝取决于拷贝的对象大小，对象越大，则性能越低。

函数

3. 函数参数传递方式：

注意2：map、slice、chan、指针、interface默认以引用的方式传递

```
package main

import "fmt"

func modify(a int) {
        a = 100
}

func main() {
        a := 8
        fmt.Println(a)
        modify(a)
        fmt.Println(a)
}
```

函数

3. 函数参数传递方式：

练习13：修改上一页的程序，使其功能正确。

函数

4. 命名返回值的名字：

```
func add(a, b int) (c int) {
        c = a + b
        return
}
```

```
func calc(a, b int) (sum int, avg int) {
        sum = a + b
        avg = (a +b)/2
        return
}
```

函数

4. _标识符，用来忽略返回值：

```
func calc(a, b int) (sum int, avg int) {
        sum = a + b
        avg = (a +b)/2
        return
}
func main() {
      sum, _ := calc(100, 200)
}
```

函数

5. 可变参数：

```
func add(arg…int) int {
}
```

0个或多个参数

```
func add(a int, arg…int) int {
}
```

1个或多个参数

```
func add(a int, b int, arg…int) int {
}
```

2个或多个参数

注意：其中arg是一个slice，我们可以通过arg[index]依次访问所有参数

通过len(arg)来判断传递参数的个数

函数

5. 可变参数：

练习14：写一个函数add，支持1个或多个int相加，并返回相加结果

练习15：写一个函数concat，支持1个或多个string相拼接，并返回结果

函数

6. defer用途：

    1. 当函数返回时，执行defer语句。因此，可以用来做资源清理

    2. 多个defer语句，按先进后出的方式执行

    3. defer语句中的变量，在defer声明时就决定了。

函数

6. defer行为特征：

```
func a() {
        i := 0
        defer fmt.Println(i)
        i++
        return
}
```

```
func f() {
        for i := 0; i < 5; i++ {
                defer fmt.Printf(“%d “, i)
        }
}
```

函数

6. defer用途：

    1. 关闭文件句柄

```
func read() {
    file := open(filename)
    defer file.Close()

    //文件操作
}
```

函数

6. defer用途：

2. 锁资源释放

```
func read() {
	mc.Lock()
	defer mc.Unlock()
	//其他操作
}
```

函数

6. defer用途：

3. 数据库连接释放

```
func read() {
    conn := openDatabase()
    defer conn.Close()
    //其他操作
}
```

递归函数

1. 一个函数调用自己，就叫做递归。

```
package main

import (
	"fmt"
)

func calc(n int) int {
	if n == 1 {
		return 1
	}

	return test(n-1) * n
}

func main() {
	n := calc(5)
	fmt.Println(n)
}
```

递归函数

2.斐波那契数

```
package main

import "fmt"

func fab(n int) int {
	if n <= 1 {
		return 1
	}

	return fab(n-1) + fab(n-2)
}

func main() {
	for i := 0; i < 10; i++ {
		n := fab(i)
		fmt.Println(n)
	}
}
```

递归函数

2.斐波那契数

```
package main

import "fmt"

func fab(n int) int {
	if n <= 1 {
		return 1
	}

	return fab(n-1) + fab(n-2)
}

func main() {
	for i := 0; i < 10; i++ {
		n := fab(i)
		fmt.Println(n)
	}
}
```

递归函数

3. 递归的设计原则

1）一个大的问题能够分解成相似的小问题

2）定义好出口条件

闭包

1. 闭包：一个函数和与其相关的引用环境组合而成的实体

```
package main

import “fmt”

func main() {
        var f = Adder()
        fmt.Print(f(1),” - “)
        fmt.Print(f(20),” - “)
        fmt.Print(f(300))
}

func Adder() func(int) int {
        var x int
        return func(delta int) int {
                x += delta
        }
}
```

闭包

2. 闭包的例子

```
package main

import (
	"fmt"
	"strings"
)
func makeSuffixFunc(suffix string) func(string) string {
	return func(name string) string {
		if !strings.HasSuffix(name, suffix) {
			return name + suffix
		}
		return name
	}
}
func main() {
	func1 := makeSuffixFunc(".bmp")
	func2 := makeSuffixFunc(".jpg")
	fmt.Println(func1("test"))
	fmt.Println(func2("test"))
}
```

数组与切片

11. 排序和查找操作

排序操作主要都在 sort包中，导入就可以使用了

import(“sort”)

sort.Ints对整数进行排序， sort.Strings对字符串进行排序, sort.Float64s对浮点数进行排序.

sort.SearchInts(a []int, b int) 从数组a中查找b，前提是a必须有序

sort.SearchFloats(a []float64, b float64) 从数组a中查找b，前提是a必须有序

sort.SearchStrings(a []string, b string) 从数组a中查找b，前提是a必须有序

map数据结构

1. map简介

key-value的数据结构，又叫字典或关联数组

a. 声明

var map1 map[keytype]valuetype

var a map[string]string

var a map[string]int

var a map[int]string

var a map[string]map[string]string

声明是不会分配内存的，初始化需要make

map数据结构

2. map相关操作

```
var a map[string]string = map[string]string{“hello”, “world”}

a = make(map[string]string, 10)
```

```
a[“hello”] = “world”
```

插入和更新

```
Val, ok:= a[“hello”]
```

查找

```
for k, v := range a {
    fmt.Println(k,v)
}
```

遍历

```
delete(a, “hello”)
```

删除

```
len(a)
```

长度

map数据结构

3. map是引用类型

```
func modify(a map[string]int) {
    a[“one”] = 134
}
```

map数据结构

4. slice of map

```
Items := make([]map[int][int], 5)
For I := 0; I < 5; i++ {
        items[i] = make(map[int][int])
}
```

map数据结构

5. map排序

   a. 先获取所有key，把key进行排序

   b. 按照排序好的key，进行遍历

map数据结构

6. Map反转

a. 初始化另外一个map，把key、value互换即可

包

1. golang中的包

   a. golang目前有150个标准的包，覆盖了几乎所有的基础库

   b. golang.org有所有包的文档，没事都翻翻

包

2. 线程同步

a. import(“sync”)

b. 互斥锁, var mu sync.Mutex

c. 读写锁, var mu sync.RWMutex

包

3. go get安装第三方包

课后工作

1. 实现一个冒泡排序

2. 实现一个选择排序

3. 实现一个插入排序

4. 实现一个快速排序