

一、函数

1.函数的基本概念

为完成某一功能的程序指令(语句)的集合,称为函数。

在 Go 中,函数分为: 自定义函数、系统函数(查看 Go 编程手册)

2.函数的基本语法

```
func 函数名 (形参列表) (返回值列表) {  
    执行语句...  
    return 返回值列表  
}
```

- 1) 形参列表: 表示函数的输入
- 2) 函数中的语句: 表示为了实现某一功能代码块
- 3) 函数可以有返回值,也可以没有

```
package main  
  
import (  
  
    "awesomeProject/main_one/chapter06/funcdemo01/utils"  
    "fmt"  
)  
//函数  
func cal(n1 float64, n2 float64, operator byte)  
float64 {  
    var res float64  
    switch operator {  
    case '+':  
        res = n1 + n2  
    case '-':  
        res = n1 - n2
```

```

case '*' :
    res = n1 * n2
case '/' :
    res = n1 / n2

default:
    fmt.Println("操作符号错误")
}
return res
}

func main() {

    result := Cal(2, 3, '+')
    //result := cal(1, 2, '-')
    fmt.Println("result=", result)
}

```

3.函数的调用机制

1.函数-调用过程



对于上面这张图我们分析一下

1. 在调用一个函数时，会给该函数分配一个新的空间，编译器会通过自身的处理让这个新的空间和其它的栈的空间区分开来。
2. 在每个函数对应的栈中，数据空间是独立的，不会混淆

3. 当一个函数调用完毕(执行完毕)后，程序会销毁这个函数对应的栈空间。

2.return语句

基本语法和说明

Go函数支持返回多个值，这一点是其它编程语言没有的。[案例演示]

```
func 函数名 (形参列表) (返回值类型列表) {  
    语句...  
    return 返回值列表  
}
```

- 1) 如果返回多个值时，在接收时，希望忽略某个返回值，则使用 _ 符号表示占位忽略
- 2) 如果返回值只有一个，（返回值类型列表）可以不写()

动手敲一敲叭！

```
package main  
  
import "fmt"  
  
func getSum(n1 int, n2 int) int {  
    return n1 + n2  
}  
  
// 给函数类型去一个别名  
type myFunType func(int, int) int //myFun就是  
func(int, int) int类型的  
  
func myFun2(funvar myFunType, num1 int, num2 int)  
int {  
    return funvar(num1, num2)  
}  
  
func getSumAndSub(n1 int, n2 int) (sum int, sub int)  
{  
    sum = n1 + n2  
    sub = n1 - n2  
    return
```

```

}
func main() {
    a := getSum

    fmt.Printf("a的类型是%T，getsum类型是%T\n", a,
getSum)

    result := a(10, 40) //等价于res:=getsum(10,40)
    fmt.Println("result = ", result)

    res2 := myFun2(getSum, 50, 60)
    fmt.Println("res2=", res2)

    type myInt int //给int取一个别名
    var num1 myInt
    var num2 int
    num1 = 40
    num2 = int(num1)

    fmt.Println(num1)
    fmt.Println(num2)

    res3 := myFun2(getSum, 500, 600)
    fmt.Println("res2=", res3)

    a1, b1 := getSumAndSub(1, 2)
    fmt.Printf("a=%v b=%v\n", a1, b1)
}

```

4.函数的递归调用

1.基本介绍

一个函数在函数体内又调用了本身，我们称为递归调用。

2.小demo

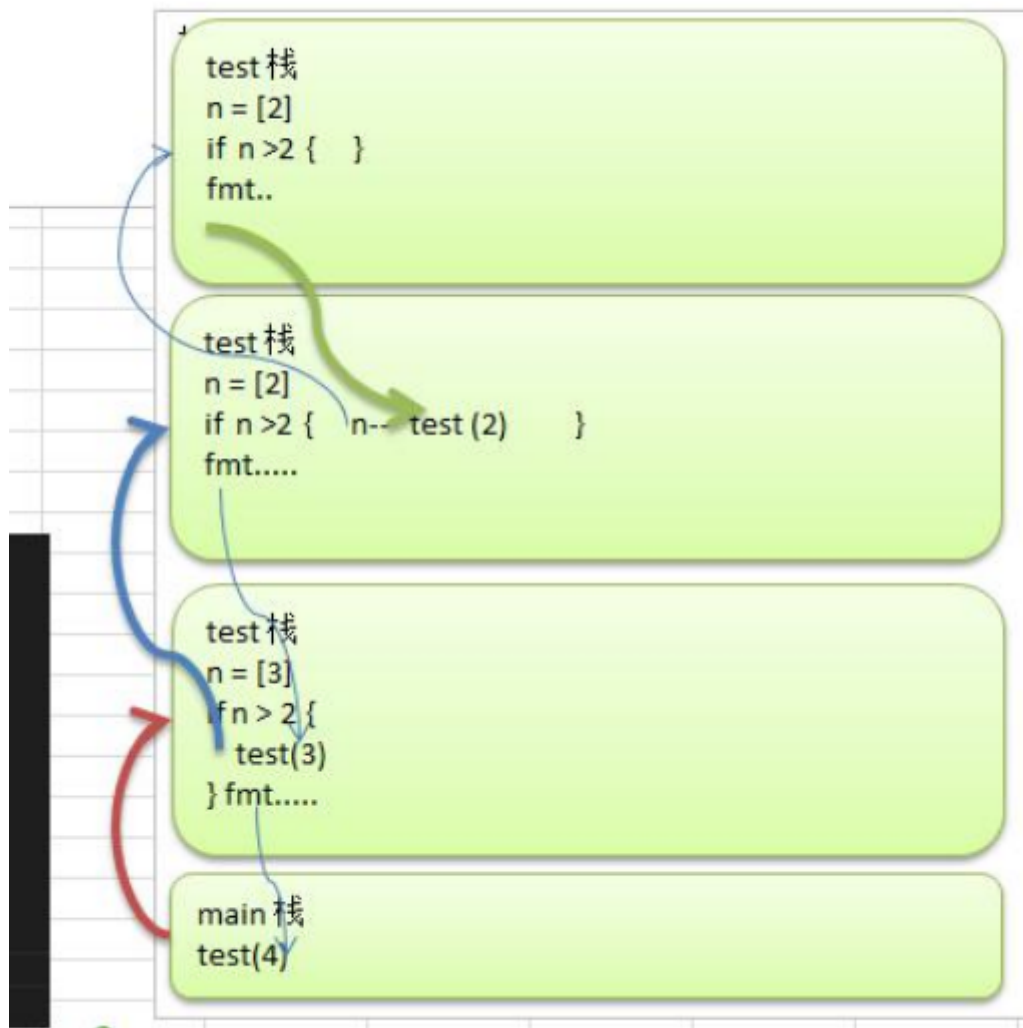
- demo01

```
package main

import "fmt"

func test(n int) {
    if n > 2 {
        n-- //递归必须向退出递归条件逼近，否则就是无限循环调用
        test(n)
    }
    fmt.Println("n= ", n)
}

func main() {
    test(4)
}
```



- demo02

```
package main

import "fmt"

func test2(n int) {
    if n > 2 {
        n--
        test2(n)
    } else {
        fmt.Println("n= ", n)
    }
}

func main() {
```

```
test2(4)
}
```



3.递归调用的总结

1. 执行一个函数时，就创建一个新的受保护的独立空间(新函数栈)
2. 函数的局部变量是独立的，不会相互影响
3. 递归必须向退出递归的条件逼近，否则就是无限递归，死龟了:)
4. 当一个函数执行完毕，或者遇到 `return`，就会返回，遵守谁调用，就将结果返回给谁，同时当函数执行完毕或者返回时，该函数本身也会被系统销毁。

5.函数使用的注意事项和细节讨论

1. 函数的形参列表可以是多个，返回值列表也可以是多个。
2. 形参列表和返回值列表的数据类型可以是值类型和引用类型。
3. 函数的命名遵循标识符命名规范，首字母不能是数字，首字母大写该函数可以被本包文件和其它包文件使用，类似 public，首字母小写，只能被本包文件使用，其它包文件不能使用，类似 private
4. 函数中的变量是局部的，函数外不生效
5. 基本数据类型和数组默认都是值传递的，即进行值拷贝。在函数内修改，不会影响到原来的值。

```
package main

import "fmt"

func test02(n1 int) {
    n1 = n1 + 10
    fmt.Println("test02 n1= ", n1)
}

func main() {

    num := 20
    test02(num)
    fmt.Println("main num=", num)
}
```

6. 如果希望函数内的变量能修改函数外的变量(指的是默认以值传递的方式的数据类型)，可以传入变量的地址&，函数内以指针的方式操作变量。从效果上看类似引用。

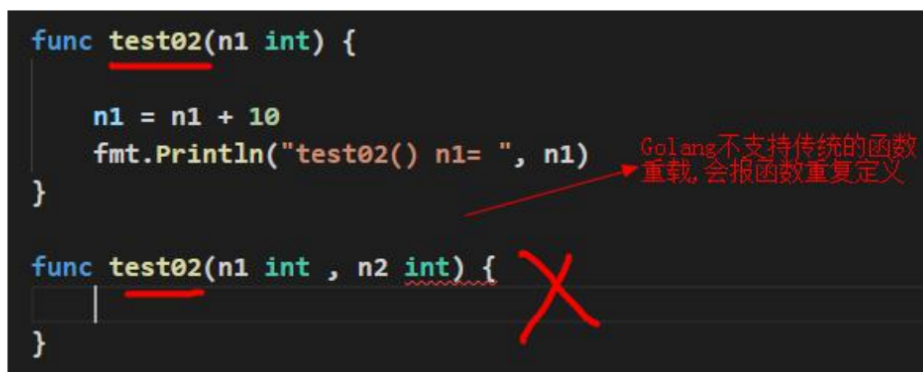
7. `package main`


```
import "fmt"

func test03(n1 *int) {
    *n1 = *n1 + 10
    fmt.Println("test03 n1= ", *n1)
}

func main() {
    num := 20
    test03(&num)
    fmt.Println("main num=", num)
}
```

8. Go 函数不支持函数重载



```
func test02(n1 int) {
    n1 = n1 + 10
    fmt.Println("test02() n1= ", n1)
}

func test02(n1 int, n2 int) {
```

Golang不支持传统的函数重载, 会报函数重复定义

9. 在 Go 中，函数也是一种数据类型，可以赋值给一个变量，则该变量就是一个函数类型的变量了。通过该变量可以对函数调用。

```
package main

import "fmt"

func getSum(n1 int, n2 int) int {
    return n1 + n2
}

func main() {
```

`a := getSum` //在 Go 中，函数也是一种数据类型，可以赋值给一个变量，则该变量就是一个函数类型的变量了。通过该变量可以对函数调用。

```
fmt.Printf("a的类型是%T，getsum类型是%T\n", a,
getSum)

result := a(10, 40) //等价于res:=getsum(10,40)
fmt.Println("result = ", result)

}
```

10. 函数既然是一种数据类型，因此在 Go 中，函数可以作为形参，并且调用。

```
package main

import "fmt"

func myFun(funvar func(int,int)int, num1 int, num2
int) int {
    return funvar(num1, num2)
}

func main() {
    res2 := myFun(getSum, 50, 60)
    fmt.Println("res2=", res2)
}
```

11. 为了简化数据类型定义，Go 支持自定义数据类型

基本语法：type 自定义数据类型名 数据类型 // 理解: 相当于一个别名

案例：type myInt int // 这时 myInt 就等价 int 来使用了.

12. `package main`

```
import "fmt"
```

```
// 给函数类型去一个别名
```

```
type myFunType func(int, int) int
```

```
//myFun就是func(int, int) int类型的
```

```
func myFun2(funvar myFunType, num1 int, num2 int)
int {
    return funvar(num1, num2)
}
```

```
func main() {
    type myInt int //给int取一个别名
    var num1 myInt
    var num2 int
    num1 = 40
    num2 = int(num1)

    fmt.Println(num1)
    fmt.Println(num2)

    res3 := myFun2(getSum, 500, 600)
    fmt.Println("res3=", res3)
}
```

13. 使用 _ 标识符，忽略返回值

14. Go 支持可变参数

```
//支持0到多个参数
func sum(args... int) sum int {
}
//支持1到多个参数
func sum(n1 int, args... int) sum int {
}
```

说明:

- (1) **args 是slice 切片, 通过 args[index] 可以访问到各个值。**
- (2) 案例演示: 编写一个函数sum, 可以求出 1到多个int的和
- (3) 如果一个函数的形参列表中有可变参数, 则可变参数需要放在形参列表最后。

```
package main

import "fmt"

// go支持可变参数
// 如果一个函数的形参列表有可变参数, 则可变参数需要放在形参列表的最后
func sum(n1 int, vars ...int) int {
    sum := n1
    for i := 0; i < len(vars); i++ {
        sum += vars[i]
    }
    return sum
}

func main() {

    res4 := sum(10, 0, -1, 90, 10, 100)
    fmt.Println(res4)
}
```

6.init函数

7.匿名函数

8.闭包

9.函数的defer

10.函数参数传递方式

11.变量作用域

12.字符串常用的系统函数

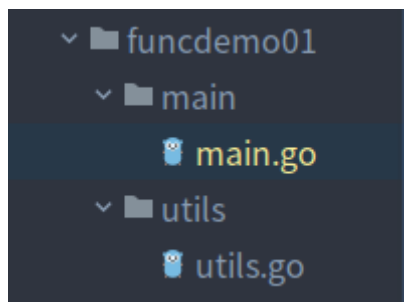
13.内置函数

二、包

1.包的引出

1. 在实际的开发中，我们往往需要在不同的文件中，去调用其它文件的定义的函数。

比如 main.go中，去使用 utils.go 文件中的函数



2.包的原理

1.main.go中的内容

```

package main

import (

    "awesomeProject/main_one/chapter06/funcdemo01/utils"
    "fmt"
)

func main() {

    result := utils.Cal(2, 3, '+')
    fmt.Println("result=", result)
}

```

2.utils.go中的内容

```

package utils

import "fmt"

// 为了让其他包使用，C要大写，该函数可导出
func Cal(n1 float64, n2 float64, operator byte)
float64 {
    var res float64
    switch operator {
    case '+':
        res = n1 + n2
    case '-':
        res = n1 - n2
    case '*':
        res = n1 * n2
    case '/':
        res = n1 / n2

    default:

```

```
        fmt.Println("操作符号错误")
    }
    return res
}
```

3.包的基本概念

go 的每一个文件都是属于一个包的，也就是说 go 是以包的形式来管理文件和项目目录结构的。

4.包的三大作用

- 区分相同名字的函数、变量等标识符
- 当程序文件很多时,可以很好的管理项目
- 控制函数、变量等访问范围，即作用域

5.包的相关说明

打包基本语法

package 包名

引入包的基本语法

import "包的路径"

6.包使用的注意事项和细节讨论

1. 在给一个文件打包时，该包对应一个文件夹，比如这里的 utils 文件夹对应的包名就是 utils,文件的包名通常和文件所在的文件夹名一致，一般为小写字母。
2. 当一个文件要使用其它包函数或变量时，需要先引入对应的包
引入方式 1：import "包名"
引入方式 2：
import (
"包名"

"包名"

)

package 指令在文件第一行，然后是 import 指令。

在 import 包时，路径从 \$GOPATH 的src 下开始，不用带 src，编译器会自动从 src 下开始引入。

3. 为了让其它包的文件，可以访问到本包的函数，则该函数名的首字母需要大写，类似其它语言的 public ,这样才能跨包访问。比如 utils.go 的。
4. 在访问其它包函数，变量时，其语法是 包名.函数名，比如这里的 main.go 文件中。
5. 在同一包下，不能有相同的函数名（也不能有相同的全局变量名），否则报重复定义。
6. 如果你要编译成一个可执行程序文件，就需要将这个包声明为 main，即 package main .这个就是一个语法规范，如果你是写一个库，包名可以自定义。

三、错误处理
