函数

函数定义

Go语言中定义函数使用 `func` 关键字，具体格式如下：

```
func 函数名(参数)(返回值){
    函数体
}
```

其中：

- 函数名：由字母、数字、下划线组成。但函数名的第一个字母不能是数字。在同一个包内，函数名也称不能重名（包的概念详见后文）。
- 参数：参数由参数变量和参数变量的类型组成，多个参数之间使用 `,`分隔。
- 返回值：返回值由返回值变量和其变量类型组成，也可以只写返回值的类型，多个返回值必须用 `()`包裹，并用 `,` 分隔。
- 函数体：实现指定功能的代码块。

```go
func sum(x int, y int) (ret int) {
    return x + y
}
func main() {
    var a, b int
    fmt.Scanf("%d", &a)
    fmt.Scanf("%d", &b)
    fmt.Println(sum(a, b))
}
```

```go
// 可变长参数
func f7(x string, y ...int) {
    fmt.Println(x)
    fmt.Println(y)
}
```

可以传入多个参数，可变长参数比需放在参数最后。

在一个命名的函数中不能够再声明函数

defer

defer语句

Go语言中的 `defer` 语句会将其后面跟随的语句进行延迟处理。在 `defer` 归属的函数即将返回时，将延迟处理的语句按 `defer` 定义的逆序进行执行，也就是说，先被 `defer` 的语句最后被执行，最后被 `defer` 的语句，最先被执行。

```go
func deferdemo() {
    fmt.Println("start")
    defer fmt.Printf("heiheihei")
    fmt.Println("end")
}
func main() {
    deferdemo()
}
```

运行结果：

```
<4 go 设置调用>
start
end
heiheihei
进程 已完成，退出代码为 0
```

发现defer后面的语句被延迟执行

一个函数中有多个defer语句

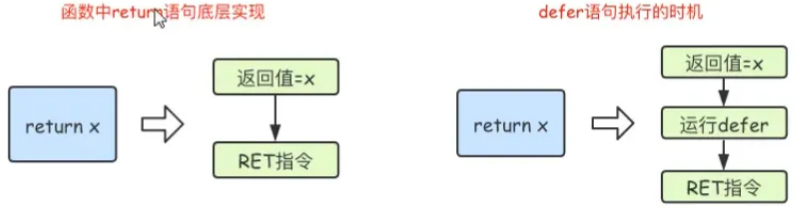
```go
defer fmt.Printf("heiheihei\n")
defer fmt.Printf("hahaha\n")
defer fmt.Printf("hehehe\n")
```

执行顺序;

```
hehehe
hahaha
heiheihei
```

defer的执行时机

在Go语言的函数中 `return` 语句在底层并不是原子操作，它分为给返回值赋值和RET指令两步。而 `defer` 语句执行的时机就在返回值赋值操作后，RET指令执行前。具体如下图所示：



变量的作用域

```go
package main

import "fmt"

var x = 100 //全局变量

func f1() {
    //函数中查找变量的顺序
    //现在函数内部查找
    //找不到就在函数外面找，直到找到
    x := 10
    name := "理想"
    fmt.Println(x, name)
}
func main() {
    f1()
    fmt.Println(name)
}
```

函数类型和变量

函数也可以是一种类型来存在

示例 Go

```go
package main

import "fmt"

func f1() {
    fmt.Println("hello world")
}

func f2() int {
    return 42
}

func main() {
    a := f1
    fmt.Printf("%T\n", a)
    b := f2
    fmt.Printf("%T\n", b)
}
```

```
<4 go 设置调用>
func()
func() int

进程 已完成，退出代码为 0
```

```go
package main

import "fmt"

func f1() {
    fmt.Println("hello world")
}

func f2() int {
    return 42
}
func f3(x func() int) {
    res := x()
    fmt.Println("res:", res)
}

func f4(x, y int) int {
    return x + y
}

// 函数还可以作为返回值
func f5(x func() int) func(int, int) int {
    ret := func(a, b int) int {
        return a * b
    }
    return ret
}
func main() {
    a := f1
    fmt.Printf("%T\n", a)
    b := f2
    fmt.Printf("%T\n", b)
    f3(f2)
    fmt.Printf("%T\n", f3)
    fmt.Printf("%T\n", f4)
    fmt.Println(f5(f2))
    fmt.Printf("%T\n", f5)
}
```