Go编程基础

本文档为视频教程**配套**课件，若单独使用可能导致你**无法理解**相关知识点

任何人**不得以**任何理由借助本课件谋取任何形式的利益，如果你发现有人谋取私利，请立即联系 **http://weibo.com/Obahua**

视频教程首页（包括课堂笔记）
https://github.com/Unknwon/go-fundamental-programming

全套视频下载
http://pan.baidu.com/share/link?shareid=393899&uk=822891499

讲师：无闻

Go编程基础

什么是Go？

Go是一门 **并发支持** 、 **垃圾回收** 的 **编译型** 系统编程语言，旨在创造一门具有在静态编译语言的 **高性能** 和动态语言的 **高效开发** 之间拥有良好平衡点的一门编程语言。

Go的主要特点有哪些？

- **类型安全** 和 **内存安全**
- 以非常直观和极低代价的方案实现 **高并发**
- 高效的垃圾回收机制
- **快速编译**（同时解决C语言中头文件太多的问题）
- 为多核计算机提供性能提升的方案
- UTF-8编码支持

讲师：无闻

Go编程基础

Go存在的价值是什么？

Go在谷歌：以软件工程为目的的语言设计

Go是记事本编程吗？

包括VIM，IDEA，Sublime Text，Eclipse等众多知名IDE均已支持

Go目前有多少实际应用和资源？

- 全球最大视频网站 Youtube（谷歌）
- 七牛云储存以及旗下网盘服务（Q盘）
- 爱好者开发的Go论坛及博客
- 已用Go开发服务端的著名企业：谷歌、盛大、七牛、360
- 其它海量开源项目：go-wiki、GoDoc、Go Language Resources

讲师：无闻

Go编程基础

Go发展成熟了吗？

作为一门2009年才正式发布的编程语言，Go是非常年轻的，因此不能称为一门成熟的编程语言，但开发社区每天都在不断更新其核心代码，给我们这些爱好者给予了很大的学习和开发动力。

Go的爱好者多吗？

以Google Group为主的邮件列表每天都会更新10至20帖，国内的Go爱好者QQ群和论坛每天也在进行大量的讨论，因此可以说目前Go爱好者群体是足够壮大。

Golang相关QQ群

Go编程基础

安装Go语言

- Go源码安装：参考链接
- Go标准包安装：下载地址
- 第三方工具安装

```
C:\Users\Unknown>go env
set GOARCH=amd64
set GOBIN=
set GOCHAR=6
set GOEXE=.exe
set GOGCCFLAGS=-g -O2 -m64 -mthreads
set GOHOSTARCH=amd64
set GOHOSTOS=windows
set GOOS=windows
set GOPATH=E:\Go\Development
set GOROOT=D:\Go
set GOTOOLDIR=D:\Go\pkg\tool\windows_amd64
set CGO_ENABLED=1
```

Go环境变量与工作目录

根据约定，GOPATH下需要建立3个目录：

- bin（存放编译后生成的可执行文件）
- pkg（存放编译后生成的包文件）
- src（存放项目源码）

Go编程基础

Go命令

在命令行或终端输入go即可查看所有支持的命令

Go常用命令简介

- go get：获取远程包（需 **提前安装** git或hg）
- go run：直接运行程序
- go build：测试编译，检查是否有编译错误
- go fmt：格式化源码（部分IDE在保存时自动调用）
- go install：编译包文件并编译整个程序
- go test：运行测试文件
- go doc：查看文档（CHM手册）

讲师：无闻

Go编程基础

程序的整体结构

```
bin/
    mathapp
pkg/
    平台名/ 如: darwin_amd64、linux_amd64
        mymath.a
        github.com/
            astaxie/
                beedb.a
src/
    mathapp
        main.go
      mymath/
        sqrt.go
      github.com/
          astaxie/
              beedb/
                  beedb.go
                  util.go
```

讲师：无闻

Go编程基础

Go开发工具安装及配置

- 本套教程主要使用 Sublime Text
- 其它IDE安装方案：参考链接

Sublime Text

- 下载Sublime Text：官方网站
- 安装gosublime（破解版可能无法安装）：安装指令
- Sublime Text 2 入门及技巧

讲师：无闻

Go编程基础

Go语言版”Hello world!”

```go
package main

import (
    "fmt"
)

func main() {
    fmt.Println("Hello world!你好，世界！")
}
```

输出：**hello.go**

```
[ `go run hello.go` | done: 937.0536ms ]
  Hello world!你好，世界！
```

讲师：无闻

Go编程基础

课堂笔记的使用方法

- 课程大纲给出了知识点讲解的时间点，方便快速定位

```
[00:00] Go基本介绍
[04:10] 安装Go语言
[05:00] Go环境变量与工作目录
[07:27] Go命令简介
[11:06] godoc建立本地文档
[15:40] Sublime Text安装与配置
[18:05] "Hello world!"
```

- 补充说明在教程录制完成后根据反馈进行修正或补充
- 相关链接给出了课件中所有用到的链接，方便在看视频的同时打开

讲师：无闻

Go编程基础

Go内置关键字（25个均为小写）

break	default	func	interface	select
case	defer	go	map	struct
chan	else	goto	package	switch
const	fallthrough	if	range	type
continue	for	import	return	var

Go注释方法

- // ：单行注释
- /* */：多行注释

讲师：无闻

Go编程基础

Go程序的一般结构：**basic_structure.go**

- Go程序是通过 **package** 来组织的（与python类似）
- 只有 **package** 名称为 **main** 的包可以包含 **main** 函数
- 一个可执行程序 **有且仅有** 一个 main 包

- 通过 **import** 关键字来导入其它非 **main** 包
- 通过 **const** 关键字来进行常量的定义
- 通过在函数体外部使用 **var** 关键字来进行全局变量的声明与赋值
- 通过 **type** 关键字来进行结构(**struct**)或接口(**interface**)的声明
- 通过 **func** 关键字来进行函数的声明

Go编程基础

Go导入 package 的格式

```
import "fmt"
import "os"
import "io"
import "time"
import "strings"
```

```
import (
    "fmt"
    "io"
    "os"
    "strings"
    "time"
)
```

- 导入包之后，就可以使用格式<PackageName>.<FuncName>来对包中的函数进行调用
- 如果导入包之后 **未调用** 其中的函数或者类型将会报出编译错误：

```
imported and not used: "io"
```

Go编程基础

package 别名

- 当使用第三方包时，包名可能会非常接近或者相同，此时就可以使用别名来进行区别和调用

```
import (
    io "fmt"
)
```

```
// 使用别名调用包
io.Println("Hello world!")
```

省略调用

- **不建议使用**，易混淆
- 不可以和别名同时使用

```
import (
    . "fmt"
)

func main() {
    // 使用省略调用
    Println("Hello world!")
}
```

讲师：无闻

Go编程基础

可见性规则

- Go语言中，使用 **大小写** 来决定该 常量、变量、类型、接口、结构或函数 是否可以被外部包所调用：

根据约定，函数名首字母 **小写** 即为private

```
func getField(v reflect.Value, i int) refle
    val := v.Field(i)
    if val.Kind() == reflect.Interface && !
        val = val.Elem()
```

函数名首字母 **大写** 即为public

```
func Printf(format string, a ...interface{}) (n int
    return Fprintf(os.Stdout, format, a...)
}
```

讲师：无闻

Go编程基础

课堂作业

既然导入多个包时可以进行简写，那么声明多个 常量、全局变量或一般类型（非接口、非结构）是否也可以用同样的方法呢？

请动手验证。

```
// 常量的定义
const (
    PI     = 3.14
    const1 = "1"
    const2 = 2
    const3 = 3
)
```

```
// 全局变量的声明与赋值
var (
    name  = "gopher"
    name1 = "1"
    name2 = 2
    name3 = 3
)
```

```
// 一般类型声明
type (
    newType int
    type1   float32
    type2   string
    type3   byte
)
```

讲师：无闻

Go编程基础

Go基本类型

- 布尔型：bool
    - 长度：1字节
    - 取值范围：true, false
    - 注意事项：不可以用数字代表true或false
- 整型：int/uint
    - 根据运行平台可能为32或64位
- 8位整型：int8/uint8
    - 长度：1字节
    - 取值范围：-128~127/0~255
- 字节型：byte（uint8别名）

Go编程基础

Go基本类型

- 16位整型：int16/uint16
    - 长度：2字节
    - 取值范围：-32768~32767/0~65535
- 32位整型：int32（rune）/uint32
    - 长度：4字节
    - 取值范围：-2^32/2~2^32/2-1/0~2^32-1
- 64位整型：int64/uint64
    - 长度：8字节
    - 取值范围：-2^64/2~2^64/2-1/0~2^64-1
- 浮点型：float32/float64
    - 长度：4/8字节
    - 小数位：精确到7/15小数位

Go编程基础

Go基本类型

- 复数：complex64/complex128
    - 长度：8/16字节
- 足够保存指针的 32 位或 64 位整数型：uintptr

- 其它值类型：
    - array、struct、string
- 引用类型：
    - slice、map、chan

- 接口类型：inteface
- 函数类型：func

Go编程基础

类型零值

零值并不等于空值，而是当变量被声明为某种类型后的默认值，通常情况下值类型的默认值为0，bool为false，string为空字符串

类型别名

```
type (
    byte int8
    rune int32
    文本   string
)
```

```
var b 文本
b = "中文啊亲"
```

Go编程基础

单个变量的声明与赋值

- 变量的声明格式：var <变量名称> <变量类型>
- 变量的赋值格式：<变量名称> = <表达式>
- 声明的同时赋值：var <变量名称> [变量类型] = <表达式>

```
var a int // 变量的声明
a = 123   // 变量的赋值

// 变量声明的同时赋值
var b int = 321
// 上行的格式可省略变量类型，由系统推断
var c = 321
// 变量声明与赋值的最简写法
d := 456
```

讲师：无闻

Go编程基础

多个变量的声明与赋值

- 全局变量的声明可使用 var() 的方式进行简写
- 全局变量的声明不可以省略 var，但可使用并行方式
- 所有变量都可以使用类型推断
- 局部变量不可以使用 var() 的方式简写，只能使用并行方式

```
var (
    // 使用常规方式
    aaa = "hello"
    // 使用并行方式以及类型推断
    sss, bbb = 1, 2
    // ccc := 3 // 不可以省略 var
)
```

```
// 多个变量的声明
var a, b, c, d int
// 多个变量的赋值
a, b, c, d = 1, 2, 3, 4

// 多个变量声明的同时赋值
var e, f, g, h int = 5, 6, 7, 8
// 省略变量类型，由系统推断
var i, j, k, l = 9, 10, 11, 12
// 多个变量声明与赋值的最简写法
i, m, n, o := 13, 14, 15, 16
```

Go编程基础

变量的类型转换

- Go中不存在隐式转换，所有类型转换必须显式声明
- 转换只能发生在两种相互兼容的类型之间
- 类型转换的格式：

  <ValueA> [:]= <TypeOfValueA>(<ValueB>)

```
// 在相互兼容的两种类型之间进行转换
var a float32 = 1.1
b := int(a)

// 以下表达式无法通过编译
var c bool = true
d := int(c)
```

Go编程基础

课堂作业

- 请尝试运行以下代码，看会发生什么，并思考为什么。

```
func main() {
    var a int = 65
    b := string(a)
    fmt.Println(b)
}
```

string() 表示将数据转换成文本格式，因为计算机中存储的任何东西本质上都是数字，因此此函数自然地认为我们需要的是用数字65表示的文本 A。

Go编程基础

常量的定义

- 常量的值在编译时就已经确定
- 常量的定义格式与变量基本相同
- 等号右侧必须是常量或者常量表达式
- 常量表达式中的函数必须是内置函数

```
// 定义单个常量
const a int = 1
const b = 'A'
const (
    text   = "123"
    length = len(text)
    num    = b * 20
)

// 同时定义多个变量
const i, j, k = 1, "2", '3'
const (
    text2, length2, num2 = "456", len(text2), k * 10
)
```

讲师：无闻

Go编程基础

常量的初始化规则与枚举

- 在定义常量组时，如果不提供初始值，则表示将使用上行的表达式
- 使用相同的表达式不代表具有相同的值
- iota是常量的计数器，从0开始，组中每定义1个常量自动递增1
- 通过初始化规则与iota可以达到枚举的效果
- 每遇到一个const关键字，iota就会重置为0

```
const (
    // a与b都为"A"
    a = "A"
    b
    c = iota
    d // d的值为3
)

const (
    e = iota
    f // f的值为1
)
```

```
// 星期枚举
const (
    // 第一个常量不可省略表达式
    Monday = iota
    Tuesday
    Wednesday
    Thursday
    Friday
    Saturday
    Sunday
)
```

讲师：无闻

Go编程基础

运算符

- Go中的运算符均是从左至右结合

优先级（从高到低）

- ^ !（一元运算符）
- * / % << >> & &^
- + - | ^（二元运算符）
- == != < <= >= >
- <-（专门用于channel）
- &&
- ||

讲师：无闻

Go编程基础

课堂作业

- 请尝试结合常量的iota与<<运算符实现计算机储存单位的枚举

```go
const (
    _           = iota
    KB float64 = 1 << (iota * 10)
    MB
    GB
    TB
    PB
    EB
    ZB
    YB
)
```

```
1024
1.048576e+06
1.073741824e+09
1.099511627776e+12
1.125899906842624e+15
1.152921504606847e+18
1.1805916207174113e+21
1.2089258196146292e+24
```

讲师：无闻

Go编程基础

指针

Go虽然保留了指针，但与其它编程语言不同的是，在Go当中不支持指针运算以及”->”运算符，而直接采用”.”选择符来操作指针目标对象的成员

- 操作符”&”取变量地址，使用”*”通过指针间接访问目标对象
- 默认值为 nil 而非 NULL

递增递减语句

在Go当中，++ 与 -- 是作为语句而并不是作为表达式

Go编程基础

判断语句if

- 条件表达式没有括号
- 支持一个初始化表达式（可以是并行方式）
- 左大括号必须和条件语句或else在同一行
- 支持单行模式
- 初始化语句中的变量为block级别，同时隐藏外部同名变量
- 1.0.3版本中的编译器BUG

```
func main() {
    a := true
    if a, b, c := 1, 2, 3; a+b+c > 6 {
        fmt.Println("大于6")
    } else {
        fmt.Println("小于等于6")
        fmt.Println(a)
    }
    fmt.Println(a)
}
```

讲师：无闻

Go编程基础

循环语句for

- Go只有for一个循环语句关键字，但支持3种形式
- 初始化和步进表达式可以是多个值
- 条件语句每次循环都会被重新检查，因此不建议在条件语句中使用函数，尽量提前计算好条件并以变量或常量代替
- 左大括号必须和条件语句在同一行

```
func main() {
    a := 1
    for {
        a++
        if a > 3 {
            break
        }
    }
    fmt.Println(a)
}
```

```
func main() {
    a := 1
    for a <= 3 {
        a++
    }
    fmt.Println(a)
}
```

```
func main() {
    a := 1
    for i := 0; i < 3; i++ {
        a++
    }
    fmt.Println(a)
}
```

讲师：无闻

Go编程基础

选择语句switch

- 可以使用任何类型或表达式作为条件语句
- 不需要写break，一旦条件符合自动终止
- 如希望继续执行下一个case，需使用fallthrough语句
- 支持一个初始化表达式（可以是并行方式），右侧需跟分号
- 左大括号必须和条件语句在同一行

```go
func main() {
    a := 1
    switch a {
    case 0:
        fmt.Println("a=0")
    case 1:
        fmt.Println("a=1")
    }
    fmt.Println(a)
}
```

```go
func main() {
    a := 1
    switch {
    case a >= 0:
        fmt.Println("a=0")
        fallthrough
    case a >= 1:
        fmt.Println("a=1")
    }
    fmt.Println(a)
}
```

```go
func main() {
    switch a := 1; {
    case a >= 0:
        fmt.Println("a=0")
        fallthrough
    case a >= 1:
        fmt.Println("a=1")
    }
}
```

讲师：无闻

Go编程基础

跳转语句goto, break, continue

- 三个语法都可以配合标签使用
- 标签名区分大小写，若不使用会造成编译错误
- Break与continue配合标签可用于多层循环的跳出
- Goto是调整执行位置，与其它2个语句配合标签的结果并不相同

```go
func main() {
LABEL:
    for {
        for i := 0; i < 10; i++ {
            if i > 2 {
                break LABEL
            } else {
                fmt.Println(i)
            }
        }
    }
}
```

```go
func main() {
LABEL:
    for i := 0; i < 10; i++ {
        for {
            fmt.Println(i)
            continue LABEL
        }
    }
}
```

Go编程基础

课堂作业

- 将下图中的continue替换成goto，程序运行的结果还一样吗？
- 请尝试并思考为什么。

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

Goto是调整执行位置

讲师：无闻

Go编程基础

数组Array

- 定义数组的格式：var <varName> [n]<type>，n>=0
- 数组长度也是类型的一部分，因此具有不同长度的数组为不同类型
- 注意区分指向数组的指针和指针数组
- 数组在Go中为值类型
- 数组之间可以使用==或!=进行比较，但不可以使用<或>
- 可以使用new来创建数组，此方法返回一个指向数组的指针
- Go支持多维数组

Go语言版冒泡排序

Go编程基础

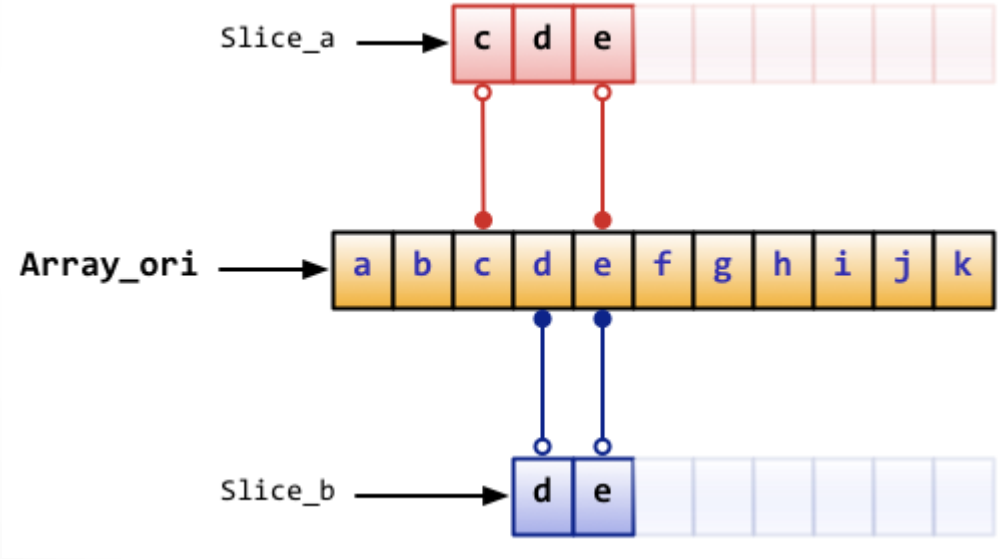
切片Slice

- 其本身并不是数组，它指向底层的数组
- 作为变长数组的替代方案，可以关联底层数组的局部或全部
- 为引用类型
- 可以直接创建或从底层数组获取生成
- 使用len()获取元素个数，cap()获取容量
- 一般使用make()创建
- 如果多个slice指向相同底层数组，其中一个的值改变会影响全部

- make([]T, len, cap)
- 其中cap可以省略，则和len的值相同
- len表示存数的元素个数，cap表示容量

Go编程基础

Slice与底层数组的对应关系



本图来源《Go Web编程》

讲师：无闻

Go编程基础

Reslice

- Reslice时索引以被slice的切片为准
- 索引不可以超过被slice的切片的容量cap()值
- 索引越界不会导致底层数组的重新分配而是引发错误

Append

- 可以在slice尾部追加元素
- 可以将一个slice追加在另一个slice尾部
- 如果最终长度未超过追加到slice的容量则返回原始slice
- 如果超过追加到的slice的容量则将重新分配数组并拷贝原始数据

Copy

Go编程基础

map

- 类似其它语言中的哈希表或者字典，以key-value形式存储数据
- Key必须是支持==或!=比较运算的类型，不可以是函数、map或slice
- Map查找比线性搜索快很多，但比使用索引访问数据的类型慢100倍
- Map使用make()创建，支持 := 这种简写方式

- make([keyType]valueType, cap)，cap表示容量，可省略
- 超出容量时会自动扩容，但尽量提供一个合理的初始值
- 使用len()获取元素个数

- 键值对不存在时自动添加，使用delete()删除某键值对
- 使用 for range 对map和slice进行迭代操作

Go编程基础

课堂作业

- 根据在 for range 部分讲解的知识，尝试将类型为map[int]string的键和值进行交换，变成类型map[string]int
- 程序正确运行后应输出如下结果：

```
[ `go run temp.go` | done: 900.0515ms ]
  m1 map[0:j 8:h 3:c 9:i 5:e 7:g 6:f 4:d 2:b 1:a]
  m2 map[b:2 i:9 h:8 d:4 e:5 g:7 a:1 j:0 f:6 c:3]
```

讲师：无闻

Go编程基础

函数function

- Go 函数 **不支持** 嵌套、重载和默认参数
- 但支持以下特性：

  无需声明原型、不定长度变参、多返回值、命名返回值参数
  匿名函数、闭包

- 定义函数使用关键字 func，且左大括号不能另起一行
- 函数也可以作为一种类型使用

Go编程基础

defer

- 的执行方式类似其它语言中的析构函数，在函数体执行结束后按照调用顺序的相反顺序逐个执行
- 即使函数发生严重错误也会执行
- 支持匿名函数的调用
- 常用于资源清理、文件关闭、解锁以及记录时间等操作
- 通过与匿名函数配合可在return之后修改函数计算结果
- 如果函数体内某个变量作为defer时匿名函数的参数，则在定义defer时即已经获得了拷贝，否则则是引用某个变量的地址

- Go 没有异常机制，但有 panic/recover 模式来处理错误
- Panic 可以在任何地方引发，但recover**只有**在defer调用的函数中有效

Go编程基础

课堂作业

- 运行以下程序并分析输出结果。

```go
func main() {
    var fs = [4]func(){}

    for i := 0; i < 4; i++ {
        defer fmt.Println("defer i = ", i)
        defer func() { fmt.Println("defer_closure i = ", i) }()
        fs[i] = func() { fmt.Println("closure i = ", i) }
    }

    for _, f := range fs {
        f()
    }
}
```

讲师：无闻

Go编程基础

结构struct

- Go 中的struct与C中的struct非常相似，并且Go没有class
- 使用 type <Name> struct{} 定义结构，名称遵循可见性规则
- 支持指向自身的指针类型成员
- 支持匿名结构，可用作成员或定义成员变量
- 匿名结构也可以用于map的值
- 可以使用字面值对结构进行初始化
- 允许直接通过指针来读写结构成员
- 相同类型的成员可进行直接拷贝赋值
- 支持 == 与 !=比较运算符，但不支持 > 或 <
- 支持匿名字段，本质上是定义了以某个类型名为名称的字段
- 嵌入结构作为匿名字段看起来像继承，但不是继承
- 可以使用匿名字段指针

Go编程基础

课堂作业

- 如果匿名字段和外层结构有同名字段，应该如何进行操作？
- 请思考并尝试。

Go编程基础

方法method

- Go 中虽没有class，但依旧有method
- 通过显示说明receiver来实现与某个类型的组合
- 只能为同一个包中的类型定义方法
- Receiver 可以是类型的值或者指针
- 不存在方法重载
- 可以使用值或指针来调用方法，编译器会自动完成转换
- 从某种意义上来说，方法是函数的语法糖，因为receiver其实就是方法所接收的第1个参数（Method Value vs. Method Expression）
- 如果外部结构和嵌入结构存在同名方法，则优先调用外部结构的方法
- 类型别名不会拥有底层类型所附带的方法
- 方法可以调用结构中的非公开字段

Go编程基础

课堂作业

- 根据为结构增加方法的知识，尝试声明一个底层类型为int的类型，并实现调用某个方法就递增100。

如：a:=0，调用a.Increase()之后，a从0变成100。

相关资源

Go Web编程交流QQ群：259316004
Golang中文社区：http://mygolang.com
Go语言学习园地：http://studygolang.com/
Golang中国：http://golang.tc

感谢 **优才网** 对本套教程的大力支持！
http://www.ucai.cn/course/show/69