

Go 学习 笔记

第 4 版

好好学习 天天向上



前言

本书在不侵犯作者个人权利的情况下，可自由散播。

- 下载：不定期更新，<https://github.com/qyuheng/book>。
- 联系：qyuheng@hotmail.com



雨痕 二〇一四年秋末



更新

- 2012-01-11 开始学习 Go。
- 2012-01-15 第一版, 基于 R60。
- 2012-03-29 升级到 1.0。
- 2012-06-15 升级到 1.0.2。
- 2013-03-26 升级到 1.1。
- 2013-12-12 第二版, 基于 1.2。
- 2014-05-22 第三版, 基于 1.3。
- 2014-12-20 第四版, 基于 1.4。



目录

第一部分 语言	8
第 1 章 类型	9
1.1 变量	9
1.2 常量	10
1.3 基本类型	13
1.4 引用类型	14
1.5 类型转换	14
1.6 字符串	15
1.7 指针	17
1.8 自定义类型	19
第 2 章 表达式	21
2.1 保留字	21
2.2 运算符	21
2.3 初始化	22
2.4 控制流	23
第 3 章 函数	29
3.1 函数定义	29
3.2 变参	30
3.3 返回值	30
3.4 匿名函数	32
3.5 延迟调用	34
3.6 错误处理	35
第 4 章 数据	39
4.1 Array	39
4.2 Slice	40
4.3 Map	45

4.4 Struct	47
第 5 章 方法	53
5.1 方法定义	53
5.2 匿名字段	54
5.3 方法集	56
5.4 表达式	56
第 6 章 接口	60
6.1 接口定义	60
6.2 执行机制	62
6.3 接口转换	63
6.4 接口技巧	65
第 7 章 并发	66
7.1 Goroutine	66
7.2 Channel	68
第 8 章 包	76
8.1 工作空间	76
8.2 源文件	76
8.3 包结构	77
8.4 文档	81
第 9 章 进阶	82
9.1 内存布局	82
9.2 指针陷阱	83
9.3 cgo	86
9.4 Reflect	94
第二部分 源码	109
1. Memory Allocator	110
1.1 初始化	112
1.2 分配流程	117

1.3 释放流程	131
1.4 其他	135
2. Garbage Collector	140
2.1 初始化	140
2.2 垃圾回收	141
2.3 内存释放	155
2.4 状态输出	160
3. Goroutine Scheduler	166
3.1 初始化	166
3.2 创建任务	171
3.3 任务线程	178
3.4 任务执行	184
3.5 连续栈	196
3.6 系统调用	207
3.7 系统监控	211
3.8 状态输出	217
4. Channel	218
4.1 初始化	218
4.2 收发数据	220
4.3 选择模式	227
5. Defer	235
6. Finalizer	241
第三部分 附录	249
A. 工具	250
1. 工具集	250
2. 条件编译	251
3. 跨平台编译	253
4. 预处理	254

B. 调试	255
1. GDB	255
2. Data Race	255
C. 测试	258
1. Test	258
2. Benchmark	260
3. Example	261
4. Cover	261
5. PProf	262

第一部分 语言

第 1 章 类型

1.1 变量

Go 是静态类型语言，不能在运行期改变变量类型。

使用关键字 **var** 定义变量，自动初始化为零值。如果提供初始化值，可省略变量类型，由编译器自动推断。

```
var x int
var f float32 = 1.6
var s = "abc"
```

在函数内部，可用更简略的 **:=** 方式定义变量。

```
func main() {
    x := 123           // 注意检查，是定义新局部变量，还是修改全局变量。该方式容易造成错误。
}
```

可一次定义多个变量。

```
var x, y, z int
var s, n = "abc", 123

var (
    a int
    b float32
)

func main() {
    n, s := 0x1234, "Hello, World!"
    println(x, s, n)
}
```

多变量赋值时，先计算所有相关值，然后再从左到右依次赋值。

```
data, i := [3]int{0, 1, 2}, 0
i, data[i] = 2, 100           // (i = 0) -> (i = 2), (data[0] = 100)
```

特殊只写变量 **_**，用于忽略值占位。

```
func test() (int, string) {
    return 1, "abc"
}

func main() {
    _, s := test()
    println(s)
}
```

编译器会将未使用的局部变量当做错误。

```
var s string    // 全局变量没问题。

func main() {
    i := 0      // Error: i declared and not used. (可使用 "_ = i" 规避)
}
```

注意重新赋值与定义新同名变量的区别。

```
s := "abc"
println(&s)

s, y := "hello", 20    // 重新赋值：与前 s 在同一层次的代码块中，且有新的变量被定义。
println(&s, y)         // 通常函数多返回值 err 会被重复使用。

{
    s, z := 1000, 30    // 定义新同名变量：不在同一层次代码块。
    println(&s, z)
}
```

输出：

```
0x2210230f30
0x2210230f30 20
0x2210230f18 30
```

1.2 常量

常量值必须是编译期可确定的数字、字符串、布尔值。

```
const x, y int = 1, 2    // 多常量初始化
const s = "Hello, World!" // 类型推断

const (                  // 常量组
    a, b    = 10, 100
    c bool = false
```

```

)

func main() {
    const x = "xxx"           // 未使用局部常量不会引发编译错误。
}

```

不支持 1UL、2LL 这样的类型后缀。

在常量组中，如不提供类型和初始化值，那么视作与上一常量相同。

```

const (
    s    = "abc"
    x           // x = "abc"
)

```

常量值还可以是 len、cap、unsafe.Sizeof 等编译期可确定结果的函数返回值。

```

const (
    a    = "abc"
    b    = len(a)
    c    = unsafe.Sizeof(b)
)

```

如果常量类型足以存储初始化值，那么不会引发溢出错误。

```

const (
    a    byte = 100           // int to byte
    b    int  = 1e20          // float64 to int, overflows
)

```

枚举

关键字 `iota` 定义常量组中从 0 开始按行计数的自增枚举值。

```

const (
    Sunday = iota           // 0
    Monday           // 1, 通常省略后续行表达式。
    Tuesday          // 2
    Wednesday        // 3
    Thursday         // 4
    Friday           // 5
    Saturday         // 6
)

```

```
const (
    _      = iota           // iota = 0
    KB  int64 = 1 << (10 * iota) // iota = 1
    MB                               // 与 KB 表达式相同, 但 iota = 2
    GB
    TB
)
```

在同一常量组中, 可以提供多个 `iota`, 它们各自增长。

```
const (
    A, B = iota, iota << 10 // 0, 0 << 10
    C, D           // 1, 1 << 10
)
```

如果 `iota` 自增被打断, 须显式恢复。

```
const (
    A  = iota // 0
    B      // 1
    C  = "c"  // c
    D      // c, 与上一行相同。
    E  = iota // 4, 显式恢复。注意计数包含了 C、D 两行。
    F      // 5
)
```

可通过自定义类型来实现枚举类型限制。

```
type Color int

const (
    Black Color = iota
    Red
    Blue
)

func test(c Color) {}

func main() {
    c := Black
    test(c)

    x := 1
    test(x) // Error: cannot use x (type int) as type Color in function argument
}
```

```
test(1) // 常量会被编译器自动转换。
}
```

1.3 基本类型

更明确的数字类型命名，支持 Unicode，支持常用数据结构。

类型	长度	默认值	说明
bool	1	false	
byte	1	0	uint8
rune	4	0	Unicode Code Point, int32
int, uint	4 或 8	0	32 或 64 位
int8, uint8	1	0	-128 ~ 127, 0 ~ 255
int16, uint16	2	0	-32768 ~ 32767, 0 ~ 65535
int32, uint32	4	0	-21亿 ~ 21 亿, 0 ~ 42 亿
int64, uint64	8	0	
float32	4	0.0	
float64	8	0.0	
complex64	8		
complex128	16		
uintptr	4 或 8		足以存储指针的 uint32 或 uint64 整数
array			值类型
struct			值类型
string		""	UTF-8 字符串
slice		nil	引用类型
map		nil	引用类型
channel		nil	引用类型
interface		nil	接口
function		nil	函数