Go 学习

常用命令

- go get 获取远程包（需要提前安装 git 或 hg）
- go run 运行程序
- go bulid 测试编译，检查是否有编译错误
- go fmt 测试化源码
- 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

Go 编程基础

- Go 程序是通过 packge 来组织的
- 只有 package 名称为 maind 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 关键字导入其他非 main 包
- 通过 const 关键字进行常量定义
- 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

关键字

- inport
  - 设置包别名：import asd "fmt"
  - 省略导入：import . "fmt"（不建议使用）

既定规则 可见性规则

- 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
- 函数名首字母小写即为 private
- 函数首字母大写即为 public

定义简写

- 声明多个常量

```
const(
  PI      = 3.14
  contst1 = 2
  contst2 = 3
)
```

- 声明多个全局变量

```
var(
  name  = "GSY"
  name1 = 2
  name2 = 3
)
```

- 声明多个一般类型

```
type(
  nameType int
  type1 float32
  type2 string
)
```

> Go 基本数据类型

- 布尔型 bool
  - 长度 1 字节
  - 取值范围：true false
  - 注意事项：不可以用数字代替 true 或 false
- 整形 int/uint
  - 根据平台运行可能为 32 或 64 位
- 8 位整形 int8/uint8
  - 长度：1 字节
  - 取值范围：-128 ~127/0 ~ 255
- 16 位整形 int16/uint16
  - 长度：2 字节
  - 取值范围：-32768 ~32767 / 0 ~ 65535
- 32 位整形 int32(rune 别名)/uint32
  - 长度：4 字节
  - 取值范围：-2^32/2 ~ 2^32/2-1/0 ~ 2^32-1
- 64 位整形 int64/uint64
  - 长度：4 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 位小数
- 字节型 byte (uint 别名)
- 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 unintptr
- 其他值类型
  - array -->数组
  - struct --> 结构
  - string --> 字符串

- 引用类型
  - slice
  - map
  - chan
- 接口类型
  - interFace
- 函数类型
  - func

类型零值

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

```
var a int
fmt.Println(a)
//  a = 0;
var a bool
fmt.Println(a)
//  a = false;
var a string
fmt.Println(a)
//  a = '';
```

设置类型别名

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

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>

```
var name string
```

- 变量的赋值：<变量名称> = <表达式>

```
name  = "GSY"
```

- 声明的同时赋值：var <变量名称> [变量类型] = <表达式>

```
var name string  = "GSY"
```

- 最简声明赋值(系统会根据表达式结果自动推断合适的变量类型)

```
name := "GSY"
```

多个变量的声明与赋值

- 全局变量的声明可使用 var()方式进行简写

```
var(
  aaa = "hello"
  sss,bbb = 1, 2
)
```

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

```
var a,b,c,d int = 1,2,3,4
```

变量类型的转换

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

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

- 转换示例：

```
  var a float32 = 1.34
  // var b int
  b := int(a)
  // b = int(a)
  fmt.Println(b)
```

- int 65 转换为 string 类型

```
var a int = 65
b := string(a)
fmt.Println(b)
// 结果为 A
// 因为string()表示将数据转换为文本格式，因为计算机中存储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用数字65表示的文本A
// ------------------------------------------------------------
// 引入包 strconv
var a int = 65
b := strconv.Itoa(a)
fmt.Println(b)
// 结果为 65
```

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常量表达式
- 常量表达式中函数必须是内置函数
- 声明方式

```
const a int = 1
const b  = "A"
// 同时定义多个变量
const(
  text = "123"
  length = len(text)
  num = b = 20
)
const c,d,e  =  1,2,3
const text2,num3,num4  =  "345",12,false
```

运算符

- Go 中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
  - (*) (/) (%) (<< 左移) (>> 右移) (&) (&^)

- & 运算符
- 计算方式：二进制同位都为 1 才为 1，不然为 0

```
// & 二进制运算
var a = 6
// 6 转换为二进制为：0110
var b = 11
// 11 转换为二进制为：1011
// 0110
// 1011
//   =
// 0010 转为十进制为 2
fmt.Println(a & b)
// 结果 2
```

- | 运算符
- 计算方式：二进制同位只要有一个为 1 结果就为 1，不然为 0

```
// | 二进制运算
var a = 6
// 6 转换为二进制为：0110
var b = 11
// 11 转换为二进制为：1011
// 0110
// 1011
//   =
// 1111 转为十进制为 15
fmt.Println(a | b)
// 结果 15
```

- ^ 运算符
- 计算方式：二进制同位只有一个为 1 结果才为 1，不然为 0

```
// ^ 二进制运算
var a = 6
// 6 转换为二进制为：0110
var b = 11
// 11 转换为二进制为：1011
// 0110
// 1011
//   =
// 1101 转为十进制为 13
fmt.Println(a ^ b)
// 结果 13
```

- &^ 运算符

- 计算方式：二进制第二位为 1 结果为 0，不然为 1

```
// &^ 二进制运算
var a = 6
// 6 转换为二进制为：0110
var b = 11
// 11 转换为二进制为：1011
// 0110
// 1011
//   =
// 0100 转为十进制为 4
fmt.Println(a &^ b)
// 结果 4
```

- (+) (-) (|) (^)
- (==) (!=) (<) (<=) (>=) (>)
- (&&)
- 计算方式：与 & 一样，只不过前面不成立后面不会执行

```
a := 0
if a > 0 && (10/a) > 1 {
  fmt.Println("大于")
}
fmt.Println("小于等于0")
```

- (||)

> 结合常量的 itoa 与<<运算符实现计算机存储单位的枚举

```
const (
  B float64 = 1 << (iota * 10)
  KB
  MB
  GB
  TB
  PB
)
func main() {
fmt.Println(B)
fmt.Println(KB)
fmt.Println(MB)
fmt.Println(GB)
fmt.Println(TB)
fmt.Println(PB)
}
```

指针

- Go 虽然保留了指针，但是与其他编程语言不同的是，在 Go 中不支持指针运算以及"->"运算符，而直接采"."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"*"通过指针间接访问对象

```
a := 1
var p *int = &a
fmt.Println(*p)
// 结果为 1
```

- 默认值为 nil 而非 null
- 递增递减语句
  - 在 Go 当中，++与--是作为语句，不是表达式

控制语句

判断语句 if

- 提交表达式没有括号

```
a := 2
if a > 1{}else{}
```

- 支持一个初始化表达式

```
if a,b := 2,4; a > 1{}else{}
// 在if语句中初始化变量，它的作用域仅在if中
```

- 左大括号必须和条件语句或 else 在同一行
- 支持单行模式
- 初始化语句中变量为 block 级别，同时隐藏外部同名变量
- 1.0.3 版本中编译器 BUG

判断语句 for

- Go 中只有 for 一个循环语句关键字，但支持三种形式
- 初始化和步进式可以是多个值
- 条件语句每次循环都会重新检查，因此不建议在条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

for 用法

- 无限循环

```
var a int
for {
  a++
  if a > 10 {
    break
  }
  fmt.Println(a)
}
```

- 先做表达式判断，后循环

```
var a int
for a < 10 {
    a++
    fmt.Println(a)
}
```

- 经典模式

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

> 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,一但条件符合自动终止
- 如希望执行下一个 case，需使用 fallthrough 语句
- 支持一个初始化表达式（可以是并行方式），右侧需跟括号
- 左大括号必须和条件语句在同一行

> > switch 用法

```
a := 2
switch a {
case 0:
    fmt.Println(0)
case 1:
    fmt.Println(1)
default:
    fmt.Println("都不等于")
}
```

> 跳转语句

- goto
- break
- continue
- 三个语句都可配合标签使用
- 标签名区分大小写，若不使用会造成编译错误
- break 和 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 个语句配合标签结果并不相同
- break 使用->跳出指定标签循环（goto、continue 同理）

```
Lable1:
for {
    for a := 0; a < 10; a++ {
        fmt.Println(a)
        if a > 5 {
            break Lable1
        }
    }
}
fmt.Println("结束了")
```

数组 Array