课程安排，请关注**微信公众平台**或者**官方微博**

编程语言： Golang 与 html5

编程工具： Goland 和 HBuilder

预计平均一周左右更新一或二节课程

授人以鱼，不如授人以渔业。

大家好，
欢迎来到 **字节教育** 课程的学习。
字节教育官网：**www.ByteEdu.Com**
腾讯课堂地址：**Gopher.ke.qq.Com**
技术交流群 ： 221 273 219
微信公众号 ：**Golang 语言社区**
微信服务号 ：**Golang 技术社区**

目录：

字节教育

第一季 Go 语言基础、进阶、提高课

第三节　Go 语言基础语法

1、变量的定义

关键字 var，而类型信息放在变量名之后，变量声明语句不需要使用分号作为结束符。示例如下：

```
var v1 int
var v2 string
var v3 [10]int  // 数组
var v4 []int    // 数组切片
var v5 struct {
    f int
}
var v6 *int // 指针
var v7 map[string]int    // map，key 为 string 类型，value 为 int 类型
var v8 func(a int) int

var (
    v1 int
    v2 string
)
```

变量初始化

```
var v1 int = 10 // 正确的使用方式 1
var v2 = 10     // 正确的使用方式 2，编译器可以自动推导出 v2 的类型
v3 := 10        // 正确的使用方式 3，编译器可以自动推导出 v3 的类型
```

错误写法：

```
var i int
i := 2
```

变量赋值

```
var v10 int
```

字节教育

```
v10 = 123
i, j = j, i //支持变量直接交换
```

匿名变量

```
func GetName() (firstName, lastName, nickName string) {
    return "May", "Chan", "Chibi Maruko"
}
_, _, nickName := GetName() //_接受返回内容，但无法使用
```

2、常量

字面常量：

常量无类型，比如常量-12，它可以赋值给 int、uint、int32、int64、float32、float64、complex64、complex128 等类型的变量。

常量定义：

```
const Pi float64 = 3.14159265358979323846
const zero = 0.0   // 无类型浮点常量
const (
size int64 = 1024
eof = -1           // 无类型整型常量
)
const u, v float32 = 0, 3   // u = 0.0, v = 3.0，常量的多重赋值
const a, b, c = 3, 4, "foo" // a = 3, b = 4, c = "foo", 无类型整型和字符串常量
```

预定义常量 iota

```
const ( // iota 被重设为 0
    c0 = iota // c0 == 0
    c1 = iota // c1 == 1
    c2 = iota // c2 == 2
)
const (
    a = 1 << iota // a == 1 (iota 在每个 const 开头被重设为 0)
    b = 1 << iota // b == 2
    c = 1 << iota // c == 4
)
const (
```

```
    u = iota * 42 // u == 0
    v float64 = iota * 42 // v == 42.0
    w = iota * 42         // w == 84
)
const x = iota // x == 0 (因为 iota 又被重设为 0 了)
const y = iota // y == 0 (同上)
const ( // iota 被重设为 0
    c0 = iota // c0 == 0
    c1 // c1 == 1
    c2 // c2 == 2
)
const (
    a = 1 <<iota // a == 1 (iota 在每个 const 开头被重设为 0)
    b            // b == 2
    c            // c == 4
)
```

枚举

go 语言没有类似 c++的枚举，只有常量，例如：

```
const (
    Sunday = iota
    Monday
    Tuesday
    Wednesday
    Thursday
    Friday
    Saturday
    numberOfDays // 这个常量没有导出
    )
```

游戏开发过程中的枚举定义：

```
const (
    DCYYINIT = iota
    // 游戏server 列表
    C2S_GameList_Proto // C2S_GameList_Proto == 1  游戏server 列表
    S2C_GameList_Proto // S2C_GameList_Proto == 2  游戏server 列表
)

type GameList struct {
    GsID   int
    GsUrl  string
    GsPort int
}

type C2S_GameList struct {
    Protocol  uint32 // 主协议
    Protocol2 uint32 // 子协议
}

type S2C_GameList struct {
    Protocol   uint32                // 主协议
    Protocol2  uint32                // 子协议
    ServerData map[string]*GameList // server list
}
```

3、类型

基础类型

布尔类型：bool。
整型：int8、byte、int16、int、uint、uintptr 等。
浮点类型：float32、float64。
复数类型：complex64、complex128。
字符串：string。
字符类型：rune。
错误类型：error。

结构类型

指针（pointer）
数组（array）
切片（slice）
字典（map）
通道（chan）
结构体（struct）
接口（interface）

布尔类型：不能接受其他类型的赋值，不支持自动或强制的类型转换：

```
var b bool
b = 1       // 编译错误
b = bool(1) // 编译错误
```

整形：分为 int ， int8，int16，int32，int64 和无符号的 uint ， uint8，uint16，uint32，uint64 int 和 int32 在 Go 语言里被认为是两种不同的类型,不能相互赋值，不能相互比较，但和常量可以赋值和比较：

```
var value2 int32
value1 := 64            // value1 将会被自动推导为 int 类型
value2 = value1         // 编译错误
value2 = int32(value1) // 通过强制转换后可以赋值，编译通过

var i int32
var j int64
i, j = 1, 2
if i == j {             // 编译错误
    fmt.Println("i and j are equal.")
}
if i == 1 || j == 2 { // 编译通过
    fmt.Println("i and j are equal.")
}
```

位运算：Go 语言支持位运算。左移，右移，异或，与，或，取反

x << y 左移 等于乘 2

x >> y 右移 等于除以 2

浮点型：Go 语言定义了两个类型 float32 和 float64。默认不带小数点的数值被自动推断为整形，带小数点的数被自动推导为 float64.

浮点数不是一种精确的表达方式，不能直接==比较，推荐的替代方案：

```
import "math"
// p 为用户自定义的比较精度，比如 0.00001
func IsEqual(f1, f2, p float64) bool {
    return math.Fdim(f1, f2) < p
}
```

复数实际上由两个实数（在计算机中用浮点数表示）构成，一个表示实部（real），一个表示虚部（imag）。对于一个复数 z = complex(x, y)，就可以通过 Go 语言内置函数 real(z)获得该复数的实部，也就是 x，通过 imag(z)获得该复数的虚部，也就是 y。

字符串

字符串的内容可以用类似于数组下标的方式获取，但与数组不同，字符串的内容不能在初始化后被修改：

```
var str string // 声明一个字符串变量
str = "Hello world" // 字符串赋值
ch := str[0] // 取字符串的第一个字符
fmt.Printf("The length of \"%s\" is %d \n", str, len(str))
```

```
fmt.Printf("The first character of \"%s\" is %c.\n", str, ch)
str := "Hello world" // 字符串也支持声明时进行初始化的做法
str[0] = 'X' // 编译错误
```

字符串操作有：x + y 字符串连接，len(s) 字符串长度，s[i] 取字符
Go 语言支持两种方式遍历字符串。一种是以字节数组的方式遍历，一种是是以 Unicode 字符遍历

```
str := "Hello,世界"
n := len(str)
for i := 0; i < n; i++ {
    ch := str[i]            // 依据下标取字符串中的字符，类型为 byte
    fmt.Println(i, ch)
}                           //该方法获取的字符数量是 13 个

str := "Hello,世界"
    for i, ch := range str {
    fmt.Println(i, ch)          //ch 的类型为 rune
}                               //该方法获取的字符数量是 9 个
```

在 Go 语言中支持两个字符类型，一个是 byte（实际上是 uint8 的别名），代表 UTF-8 字符串的单个字节的值；另一个是 rune，代表单个 Unicode 字符。

数组：

```
//数组定义
[32]byte                    // 长度为 32 的数组，每个元素为一个字节
[2*N] struct { x, y int32 } // 复杂类型数组
[1000]*float64              // 指针数组
[3][5]int                   // 二维数组
[2][2][2]float64            // 等同于[2]([2]([2]float64))

//数组的两种遍历方法
for i := 0; i < len(array); i++ {
    fmt.Println("Element", i, "of array is", array[i])
}
for i, v := range array {
    fmt.Println("Array element[", i, "]=", v)
}
```

数组是一个值类型（value type），所有的值类型变量在赋值和作为参数传递时都将产生一次复制动作。因此，在函数体中无法修改传入的数组的内容，因为函数内操作的只是所传入数组的一个副本。

数组切片的创建方法主要有两种——基于数组和直接创建。
基于数组的创建：

```
package main
```

```
import "fmt"
func main() {
    // 先定义一个数组
    var myArray [10]int = [10]int{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
    // 基于数组创建一个数组切片
    var mySlice []int = myArray[:5]
    fmt.Println("Elements of myArray: ")
    for _, v := range myArray {
        fmt.Print(v, " ")
    }
    fmt.Println("\nElements of mySlice: ")
    for _, v := range mySlice {
        fmt.Print(v, " ")
    }
    fmt.Println()
}
```

直接创建：

```
mySlice1 := make([]int, 5) //创建一个初始元素个数为 5 的数组切片，元素初始值为 0

mySlice2 := make([]int, 5, 10)//创建一个初始元素个数为 5 的数组切片，元素初始值为 0，并预留 10 个元素的存储空间

mySlice3 := []int{1, 2, 3, 4, 5}//直接创建并初始化包含 5 个元素的数组切片
```

事实上还会有一个匿名数组被创建出来了。

数组切片可以动态增减元素。

数组切片支持 Go 语言内置的 cap()函数和 len()函数，append()函数，copy()函数

map 是一堆键值对的未排序集合。

```
var myMap map[string] PersonInfo //map 声明
myMap = make(map[string] PersonInfo) //map 创建
myMap = make(map[string] PersonInfo, 100) //map 创建时指定初始存储能力
//创建并初始化
myMap = map[string] PersonInfo{
“1234”: PersonInfo{“1”, “Jack”, “Room 101,…”},
}
myMap[“1234”] = PersonInfo{“1”, “Jack”, “Room 101,…”} //元素赋值
delete(myMap, “1234”) //元素删除.如果“1234”这个键不存在，那么这个调用将什么都不发生，也不会有什么副作用

value, ok := myMap[“1234”]//元素查找
if ok { // 找到了
```

// 处理找到的 value

}

4、流程控制

条件语句

关于条件语句，需要注意以下几点：

θ 条件语句不需要使用括号将条件包含起来()；

θ 无论语句体内有几条语句，花括号{}都是必须存在的；

θ 左花括号{必须与 if 或者 else 处于同一行；

θ 在 if 之后，条件语句之前，可以添加变量初始化语句，使用;间隔；

θ 在有返回值的函数中，不允许将“最终的”return 语句包含在 if…else…结构中，否则会编译失败：

选择语句

```
switch i {
    case 0:
        fmt.Printf("0")
    case 1:
        fmt.Printf("1")
    case 2:
        fallthrough//执行接下来的条件下的内容（这个例子中执行 case 3 中的内容）
    case 3:
        fmt.Printf("3")
    case 4, 5, 6:
        fmt.Printf("4, 5, 6")
    default:
        fmt.Printf("Default")
}
switch {
    case 0 <= Num && Num <= 3:
        fmt.Printf("0-3")
    case 4 <= Num && Num <= 6:
        fmt.Printf("4-6")
    case 7 <= Num && Num <= 9:
        fmt.Printf("7-9")
}
```

循环语句

Go 语言中的循环语句只支持 for 关键字

```
sum := 0
for i := 0; i < 10; i++ {
    sum += i
}

sum := 0
for {
    sum++
    if sum > 100 {
        break
    }
}

a := []int{1, 2, 3, 4, 5, 6}
for i, j := 0, len(a) - 1; i < j; i, j = i + 1, j - 1 {
    a[i], a[j] = a[j], a[i]
}
```

5、课后作业—难度：★ ★ ★ ★ ☆

实现一个类游戏协议处理,通过用户 init 初始化协议，实现协议处理，功能函数循环打印数据：

<1> 使用 const 定义主协议，子协议

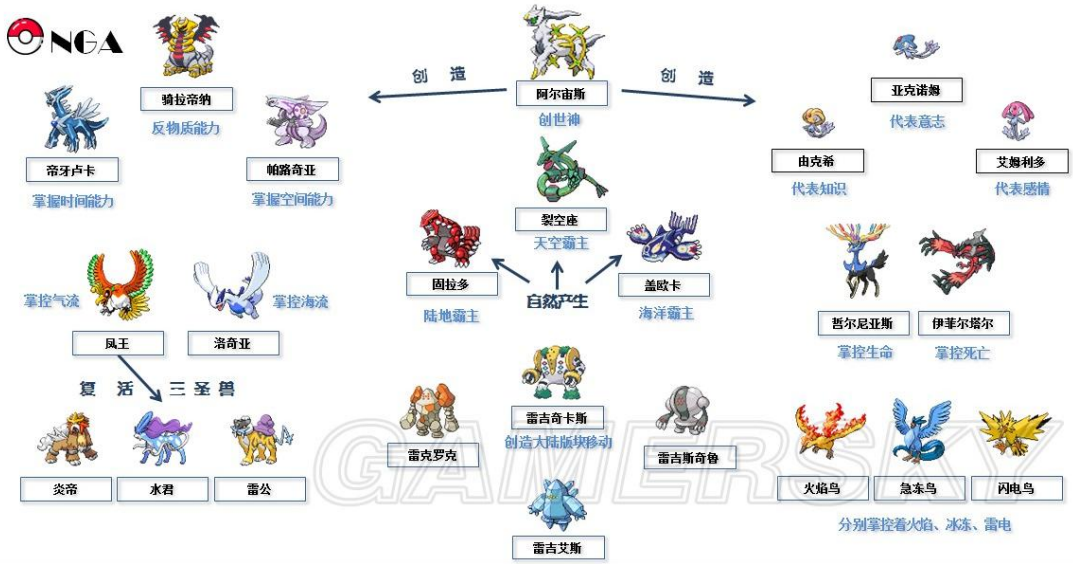
<2> 处理协议函数实现，使用 switch 实现处理

<3> 功能函数用 for 循环打印 100 组数据

6、游戏开发拓展知识—游戏名词解释

RPG：角色扮演游戏（Role-playing game）

《口袋妖怪》：由 Game Freak 和 Creatures 株式会社开发，由任天堂发行的一系列游戏



口袋妖怪世界观

口袋妖怪世界次元空间里，阿尔宙斯从虚无之地的蛋中诞生，是第一只存在于世界的口袋妖怪。它创造了口袋妖怪世界以及整个宇宙，也被称之为创世神。

阿尔宙斯分别创造了掌控着时间、空间、反物质能力的帝牙卢卡、帕路奇亚、骑拉帝纳。以及创造了分别代表共存于现代的人类和口袋妖怪之中的意志、知识与感情的亚克诺姆、由克希、和艾姆利多。

随着时间发展，固拉多、盖欧卡、裂空坐诞生了。固拉多依靠高温蒸发水分扩张大陆。盖欧卡利用下雨的能力扩张海洋。烈空坐则形成于臭氧层的物质中，以大气层中的水和颗粒为食，生活在臭氧层中。

雷吉奇卡斯拖动了用绳捆绑的大陆，这就解释了为什么口袋妖怪大陆划分成了不同的区域。

这些传说神兽，构建并维持着口袋妖怪、人类世界的和谐。其他传说口袋妖怪，也拥有着特殊的能力，保护着自己的领域。

所有的NDS平台的游戏都可以在任天堂3DS上运行，与之前向下兼容不同，3DS上运行的NDS游戏也可以进行联机。

			
GB/GBC世代	GBA世代	NDS世代	3DS世代
第一世代、第二世代	第一世代、第二世代、第三世代	第三世代、第四世代、第五世代	第一世代至第七世代

Gen3•口袋妖怪第三世代

				
红宝石	蓝宝石	绿宝石	火红	叶绿

掌机游戏

- 第一世代（Game Boy）
  - 精灵宝可梦红·绿（1996-02-27）
  - 精灵宝可梦蓝（1996-10-15）
  - 精灵宝可梦红·蓝（1998-09-28）
  - 精灵宝可梦黄（1998-09-12）
  - 精灵宝可梦卡片 GB（1998-12-18）
- 第二世代（Game Boy Color）
  - 精灵宝可梦金·银（1999-11-21）
  - 精灵宝可梦方块（2000-09-21）

- 精灵宝可梦水晶（2000-12-14）
- 精灵宝可梦卡片 GB2（2001-03-28）

- 第三世代
- （Game Boy Advance）
  - 精灵宝可梦红宝石·蓝宝石（2002-11-21）
  - 精灵宝可梦火红·叶绿（2004-01-29）
  - 精灵宝可梦绿宝石（2004-09-16）
  - 精灵宝可梦不可思议的迷宫：赤之救助队（2005-11-17）
- （nds）
  - 精灵宝可梦不可思议的迷宫：青之救助队（2005-11-17）
  - 精灵宝可梦冲刺赛（2004-12-02）
  - 精灵宝可梦益智方块（2005-10-20）
  - 精灵宝可梦保育家（2006-03-23）
- 第四世代
  - 精灵宝可梦钻石·珍珠（2006-09-28）
  - 精灵宝可梦不可思议的迷宫：时之探险队·暗之探险队（2007-09-13）
  - 精灵宝可梦保育家：巴特那吉（2008-03-20）
  - 精灵宝可梦白金（2008-09-13）
  - 精灵宝可梦不可思议的迷宫：空之探险队（2009-04-18）
  - 精灵宝可梦心金·魂银（2009-09-12）
  - 精灵宝可梦保育家：光的轨迹（2010-03-06）
- 第五世代
  - 精灵宝可梦黑·白（2010-09-18）
  - 对战与收服！精灵宝可梦打字练习 DS（2011-04-21）
  - 精灵宝可梦+信长的野望（2012-03-17）
  - 精灵宝可梦黑 2·白 2（2012-06-13）
- （任天堂3DS）
  - 超级精灵宝可梦大纷争（2011-08-11）
  - 精灵宝可梦不可思议的迷宫：极大之门与无限迷宫（2012-11-23）
- 第六世代
  - 精灵宝可梦 X·Y（2013-10-12）
  - 精灵宝可梦终极红宝石·始源蓝宝石（2014-11-21）
  - 大家的精灵宝可梦大纷争（2015-04-08）
  - 精灵宝可梦：超不可思议的迷宫 （2015-09-07）
  - 名侦探皮卡丘：新搭档诞生（2016-02-03）
- 第七世代
  - 精灵宝可梦太阳·月亮（2016-11-18）
  - 精灵宝可梦 GO（旁支非正统手机游戏）
  - 精灵宝可梦究极太阳·究极月亮（2017-11-17）

主机游戏

- 第一世代（任天堂64）
  - 精灵宝可梦竞技场（1998-08-01）
  - 精灵宝可梦竞技场 2（1999-04-30）
  - 精灵宝可梦快照（1999-03-21）

- 精灵宝可梦方块联盟（2000-09-01）
- 皮卡丘你好吗（1998-12-12）

- **第二世代**（任天堂64）
  - 精灵宝可梦竞技场金银（2000-12-14）
- **第三世代**（任天堂GameCube）
  - 精灵宝可梦圆形竞技场（2003-11-21）
  - 精灵宝可梦整理箱：红宝石·蓝宝石（2003-05-30）
  - 精灵宝可梦频道（2003-07-18）
  - 精灵宝可梦 XD：暗之旋风 黑暗洛奇亚（2005-08-04）
- **第四世代**（Wii）
  - 精灵宝可梦对战革命（2006-12-14）
  - 大家的精灵宝可梦牧场（2008-03-25）
  - 乱战！精灵宝可梦大纷争（2009-06-16）
  - 精灵宝可梦乐园 Wii 皮卡丘的大冒险（2009-12-05）
- **第五世代**（Wii&Wii U）
  - 精灵宝可梦乐园 2：超越世界（2011-11-12）
  - 精灵宝可梦大纷争 U（2013-04-24）
- **第六世代**（Wii U）
  - 宝可拳（2016-03-18）

做游戏，目前现状就是抄来抄去，策划完全只是别人家游戏的搬运工；完全把策划定义为工作职位，而不是用心去想，用心去做，请问那么这样的策划还有和意义？所以说好游戏策划真的难遇，有想法好策划的都去给自己当老板了。

7、微信公众平台及服务号

Golang 语言社区

Golang 技术社区