Go 语言从入门到项目实战

第2章 基础语法

2.1 声明

- 当一个计算机程序需要调用内存空间时，对内存发出的“占位”指令，称为“声明”；
- 无论是何种类型的数据，只有先声明，才能使用。

2.2.1 常量

- 常量（Constants）在计算机内存中，用于存储值固定不变的数据，并给出名称。
- 格式：const name [type] = value
  - 关键字 const 用于声明；
  - name 表示常量名称；
  - 中括号表示 type 部分可选，它表示常量的类型。没有指定该部分时，常量的类型将根据最后的 value（值）自动推断得出；当程序员显式指定 type 时，该常量将被限定为特定的数据类型；
  - = 为赋值符号，把 value 值赋给常量 name。一旦完成赋值，名为 name 的常量便有了值，且不会改变。
- 常量值类型只能是基本数据类型，即布尔型、数字型（整数型、浮点型和复数）和字符串型。

2.2.1 常量

```
package main
import "fmt"
//单一常量定义使用
func main() {
	//定义值为1的整型常量age
	const age=1
	//定义值为10的整型常量num，int指type
	const num int =10
	//打印输出常量，通过逗号分割可以输出多个常量
	fmt.Println("三酷猫!",age,"岁",num,"只")
}
```

2.2.1 常量

```
const (
        e = 2.7182818                   //数学中的自然常数e
        pi = 3.1415926                  //数学中的七位小数圆周率π
)
fmt.Println("数学里的常量：",e,pi)      //打印输出常量
```

2.2.1 常量

```
const (
        spring=iota          //初始值为0
        summer
        autumn
        winter
)
fmt.Println("一年四季：",spring,summer,autumn,winter)
```

2.2.1 常量

- 变量（Variable），在计算机内存中，用于存储值可变化的数据，并给出名称。
- 格式：var name [type] = [expression]
  - 关键字 var 用于声明变量定义；
  - name 是变量名；
  - type 指定变量的类型， 可以省略掉；
  - expression 是表达式， 可以省略掉。
  - type 和 expression 不能同时省略。
- 变量基本类型包括了数字型（整型、浮点型、复数）、布尔、字符串。

2.2.1 常量

- 显示指定类型

```
var age int                    //定义整型变量age，其默认初始值为0，int为指定的类型
fmt.Println(age)
```

- 类型推断

```
var age0 = 1                   //编译器根据值判断类型
```

- 连续定义同一类型的变量

```
var n1,n2,n3 int               //连续定义同一类型的变量
```

2.2.2 变量

- 批量声明

```
var (  num int              //数量
       age1 int             //年龄
       name stirng          //姓名，字符串类型
)
```

2.2.2 变量

- 短格式声明

  day:=5　　//简短格式定义

2.2.3 作用域

```
package main
import "fmt"
//变量定义
var expA=1                          //全局变量
func main() {                       //主函数
        var expA int = 2            //局部变量
        fmt.Println(expA,expB)
}
var expB=5                          //全局变量
```

2.3.1 整型

类　型	描　述	取值范围
uint	无符号32位或64位	0 ~ 2^{32}-1 或 0 ~ 2^{64}-1
uint8	无符号 8 位整型	0 ~ 2^{8}-1
uint16	无符号 16 位整型	0 ~ 2^{16}-1
uint32	无符号 32 位整型	0 ~ 2^{32}-1
uint64	无符号 64 位整型	0 ~ 2^{64}-1
int	有符号32位或64位整型	-2^{31} ~ 2^{31}-1 或 -2^{63} ~ 2^{63}-1
int8	有符号 8 位整型	-2^{7} ~ 2^{7}-1
int16	有符号 16 位整型	-2^{15} ~ 2^{15}-1
int32	有符号 32 位整型	-2^{31} ~ 2^{31}-1
int64	有符号 64 位整型	-2^{63} ~ 2^{63}-1

2.3.1 整型

```
func main() {
    var number int                                  //标准格式声明变量number
    number=3                                        //给变量赋新值3
    fmt.Println("三酷猫!有",number,"只。") //打印输出
}
```

2.3.2 浮点型

类　型	描　述	取值范围（近似值）
float32	十进制条件下，6位有效数字的浮点型	1.4e-45 ~ 3.4e38
float64	十进制条件下，15位有效数字的浮点型	4.9e-324 ~ 1.8e308

2.3.2 浮点型

```
func main() {
    var float32Number float32                  //标准格式声明变量float32Number
    float32Number=math.MaxFloat32              //给变量赋值
    fmt.Println(float32Number)                 //打印输出
    var float64Number float64                  //标准格式声明变量float64Number
    float64Number=math.MaxFloat64              //给变量赋值
    fmt.Println(float64Number)                 //打印输出
}
```

2.3.2 浮点型

- 保持精度的浮点型运算

```
func main() {
        var float32Number float32                  //标准格式声明变量float32Number
        float32Number=math.MaxFloat32              //给变量赋值
        fmt.Printf("%9.2e",float32Number)          //打印输出
        fmt.Println()                              //打印空行
        var float64Number float64                  //标准格式声明变量float64Number
        float64Number=math.MaxFloat64              //给变量赋值
        fmt.Printf("%.2e",float64Number)           //打印输出
        fmt.Println()                              //打印空行
        fmt.Printf("%9.2f",123.4567)               //打印输出
}
```

2.3.3 复数型

```
func main() {
    var complexOne complex128          //标准格式声明变量complexOne
    complexOne=complex(5,10)           //给变量赋值
    fmt.Println(complexOne)            //打印输出复数的值
    fmt.Println(real(complexOne))      //打印输出实部
    fmt.Println(imag(complexOne))      //打印输出虚部
}
```

2.3.4 布尔型

- 布尔型（Boolean）通常用作流程控制中的条件判断；
- 只有两个取值，分别是真（true）或假（false）。

2.3.5 字符串型

- 基本使用

```
func main() {
        var text string                 //标准格式声明变量text
        text="Hello，三酷猫"             //给变量赋值
        fmt.Println(text)               //打印输出
}
```

2.3.5 字符串型

- 字符串长度的获取及特定位置字符的获取

```
func main() {
        var text string                 //标准格式声明变量text
        text="Hello，三酷猫"              //给变量赋值
        fmt.Println(len(text))          //打印输出字符串长度
        fmt.Println(text[0])            //打印输出字符串第一个字符
}
```

2.3.5 字符串型

- 截取特定范围的字符串

```
func main() {
    var text string                    //标准格式声明变量text
    text="Hello，三酷猫"                //给变量赋值
    fmt.Println(text[8:])              //打印输出最后三个中文字符
}
```

2.3.5 字符串型

- 转义字符

转义字符	作　用
\a	警告或响铃
\b	退格符
\f	换页符
\n	换行符（直接跳到下一行的同一位置）
\r	回车符（返回行首）
\t	制表符
\v	垂直制表符
\’	单引号（仅用于由单引号包裹的字面量内部）
\”	双引号（仅用于由双引号包裹的字面量内部）
\\	反斜杠
\x	十六进制转义字符
\o	八进制转义字符

2.3.5 字符串型

- 转义字符

```
func main() {
    var text string                    //标准格式声明变量text
    text="唐\t宋\t元\n明\t清\t"          //给变量赋值
    fmt.Println(text)                  //输出变量的值
}
```

2.4.1 算术运算符

算术运算符	含　义
+	相加
-	相减
*	相乘
/	相除
%	求余数
++	自增1
--	自减1

2.4.1 算术运算符

```
func main() {
        var expNumOne=5                     //声明expNumOne变量
        var expNumTwo=6                     //声明expNumTwo变量
        fmt.Println(expNumOne+expNumTwo)    //输出两个数相加的结果
        fmt.Println(expNumOne-expNumTwo)    //输出两个数相减的结果
        fmt.Println(expNumOne*expNumTwo)    //输出两个数相乘的结果
        fmt.Println(expNumTwo/expNumOne)    //输出两个数相除的结果
        fmt.Println(expNumTwo%expNumOne)    //输出两个数相除取余数的结果
        expNumOne++                         //expNumOne变量自加1
        fmt.Println(expNumOne)              //输出运算结果
        expNumTwo--                         //expNumTwo变量自减1
        fmt.Println(expNumTwo)              //输出运算结果
        var uInt8Max uint8=255              //声明uInt8Max变量，类型为uint8，值为该类型最大值
        fmt.Println(uInt8Max+1)             //输出运算结果
        var int8Max int8=127                //声明int8Max变量，类型为uint8，值为该类型最大值
        fmt.Println(int8Max+1)              //输出运算结果
}
```

2.4.2 关系运算符

关系运算符	含　义
==	相等
!=	不相等
<	小于
<=	小于或等于
>	大于
> =	大于或等于

2.4.2 关系运算符

```
func main() {
    //输出100与50+50的值是否相等
    fmt.Println(100==(50+50))
    //输出51+49与50*2的值是否不相等
    fmt.Println((51+49)!=(50*2))
    //声明字符串类型变量text，值为"abcde"
    var text string = "abcde"
    //输出text字符串中首个字符的ASCII码是否与97相等
    fmt.Println(text[0]==97)
}
```

2.4.3 逻辑运算符

逻辑运算符	含　义
&&	逻辑与（AND），当运算符前后两个条件均为true时，运算结果为true
||	逻辑或（OR），当运算符前后两个条件其中有一个为true时，运算结果为true
!	逻辑非（NOT），对运算符后面的条件结果取反，当条件结果为true时，整体运算结果为false；反之则为true。

2.4.3 逻辑运算符

```
func main() {
    //输出true和false的逻辑与结果
    fmt.Println(true&&false)
    //输出true和false的逻辑或结果
    fmt.Println(true||false)
    //输出true和false的逻辑与后的结果的逻辑非的结果
    fmt.Println(!(true&&false))
}
```

2.4.4 位运算符

位运算符	含　义
&	按位与（AND）操作，其结果是运算符前后的两数各对应的二进位相与后的结果。
|	按位或（OR）操作，其结果是运算符前后的两数各对应的二进位相或后的结果。
^	按位异或（XOR）操作，当运算符前后的两数各对应的二进位相等时，返回0；反之，返回1。
&^	按位清空（AND NOT）操作，当运算符右侧某位为1时，运算结果中的相应位值为0；反之，则为运算符左侧相应位的值。
<<	按位左移操作，该操作本质上是将某个数值乘以2的n次方，n即为左移位数。更直观地来看，其结果就是将某个数值的二进制每个位向左移了n个位置。超限的高位丢弃，低位补0。
>>	按位右移操作，该操作本质上是将某个数值除以2的n次方，n即为左移位数。更直观地来看，其结果就是将某个数值的二进制每个位向右移了n个位置。超限的低位丢弃，高位补0。

2.4.4 位运算符

```
func main() {
        var numOne int=0                //声明numOne变量
        var numTwo int=1                //声明numTwo变量
        fmt.Println(numOne&numTwo)      //输出numOne和numTwo的按位与结果
        fmt.Println(numOne|numTwo)      //输出numOne和numTwo的按位或结果
        fmt.Println(numOne^numTwo)      //输出numOne和numTwo的按位异或结果
        fmt.Println(numOne&^numTwo)     //输出numOne和numTwo的按位清空结果
        var numThree int=20             //声明numThree变量
        fmt.Println(numThree<<2)        //输出numThree左移2位后的结果
        fmt.Println(numThree>>2)        //输出numThree右移2位后的结果
}
```

2.4.5 赋值运算符

赋值运算符	含　义
=	直接将运算符后面的值赋给左侧。
+=	先将运算符左侧的值与右侧的值相加，再将相加和赋给左侧。
-=	先将运算符左侧的值与右侧的值相减，再将相减差赋给左侧。
*=	先将运算符左侧的值与右侧的值相乘，再将相乘结果赋给左侧。
/=	先将运算符左侧的值与右侧的值相除，再将相除结果赋给左侧。
%=	先将运算符左侧的值与右侧的值相除取余数，再将余数赋给左侧。
<<=	先将运算符左侧的值按位左移右侧数值个位置，再将位移后的结果赋给左侧。
>>=	先将运算符左侧的值按位右移右侧数值个位置，再将位移后的结果赋给左侧。
&=	先将运算符左侧的值与右侧的值按位与，再将位运算后的结果赋给左侧。
^=	先将运算符左侧的值与右侧的值按位异或，再将位运算后的结果赋给左侧。
|=	先将运算符左侧的值与右侧的值按位或，再将位运算后的结果赋给左侧。

2.4.5 赋值运算符

```
func main() {
        var numOne int=20           //声明numOne变量
        numOne+=20                  //numOne=numOne+20
        numOne-=10                  //numOne=numOne-10
        numOne*=100                 //numOne=numOne*100
        numOne/=20                  //numOne=numOne/20
        numOne%=4                   //numOne=numOne&4
        numOne<<=2                  //numOne=numOne<<2
        numOne>>=3                  //numOne=numOne>>3
        numOne&=0                   //numOne=numOne&0
        numOne^=1                   //numOne=numOne^1
        numOne|=0                   //numOne=numOne|0
}
```

2.4.6 指针类运算符

指针运算符	含　义
&	获取某个变量在内存中的实际地址
*	用于声明一个指针变量

2.4.6 指针类运算符

```
func main() {
    //声明numOne变量，类型为int，值为5
    var numOne int=5
    //声明pointer变量，类型为指针变量，值为numOne变量的内存地址
    var pointer *int=&numOne
    //输出numOne变量的实际内存地址
    fmt.Println(&numOne)
    //输出pointer变量表示的内存地址的变量的值
    fmt.Println(*pointer)
}
```

2.4.7 运算符优先级

优先级顺序	运 算 符
最高	* / % << >> & &^
较高	+ - | ^
一般	== != < <= > >=
较低	&&
最低	||

2.4.6 运算符优先级

```
func main() {
        var numA int=10              //声明numA变量
        var numB int=20              //声明numB变量
        var numC int=30              //声明numC变量
        var numD int=40              //声明numD变量
        fmt.Println((numA+numB)*numC+numD)
        fmt.Println(numA+numB*(numC+numD))
        fmt.Println(numA+numD==numB+numC)
        fmt.Println(numD-numC==numA+numB)
        fmt.Println(numD-numC==(numA+numB))
        fmt.Println(numC==numA+numB)
}
```