Go语言学习总结

学习概览

通过系统学习Go语言的核心知识点，我掌握了从基础语法到高级特性的完整知识体系，包括基础类型、复合类型、函数特性以及文本处理等内容。

一、基础类型与语法

学到的核心知识

1. **变量声明的多样性**
   - 掌握了三种变量声明方式：`var`关键字、类型推导、`:=`短声明
   - 理解了匿名变量`_`的使用场景
2. **常量与枚举**
   - 学会使用`const`定义常量
   - 掌握`iota`枚举器的使用规则
3. **基础数据类型体系**
   - 整型（int、uint及其变体）
   - 浮点型（float32、float64）
   - 布尔型、字符型（byte、rune）
   - 字符串和复数类型
4. **运算符与流程控制**
   - 算术、关系、逻辑、位运算符
   - if、switch、for循环结构

- range遍历机制

理解的难点

难点1：iota的重置机制

```
const (
    a = iota  // 0
    b = "B"
    c = iota  // 2（不是1！）
    d = iota  // 3
)
```

理解要点：iota在每个const块中从0开始，每行递增，即使中间有非iota赋值也会继续计数。

难点2：:= 与 var 的区别

- `:=` 只能在函数内使用，必须初始化，且左侧至少有一个新变量
- `var` 可以在全局使用，可以不初始化

难点3：类型转换的严格性

Go不允许隐式类型转换，所有转换必须显式声明：

```
var ch byte = 97
var a int = int(ch)  // 必须显式转换
```

二、复合类型

学到的核心知识

1. **指针操作**
   - `&`取地址，`*`取值
   - 使用`new()`函数创建匿名变量
   - 指针作为函数参数实现引用传递
2. **数组特性**
   - 数组长度是类型的一部分：`[2]int`和`[3]int`是不同类型
   - 数组是值类型，函数传递会完整复制
   - 支持多维数组

理解的难点

难点1：数组的值传递陷阱

```
func modify(array [5]int) {
    array[0] = 10  // 修改的是副本
}

func main() {
    array := [5]int{1, 2, 3, 4, 5}
    modify(array)
    fmt.Println(array)  // 输出 [1 2 3 4 5]，未改变！
}
```

理解要点：数组作为参数传递时会完整复制，要修改原数组需要传递指针`*[5]int`。

难点2：结构体数组中的指针字段

这是一个非常重要的难点示例：

```
type newArray struct {
    userP *user  // 指针字段
    len   int
    cap   int
}

func modifyArray(arr [2]newArray) {
    arr[0].len = 999         // 修改失败（值传递）
    arr[0].userP.name = "new"  // 修改成功（指针引用）
}
```

理解要点：

- 数组本身是值传递，所以`len`和`cap`的修改不会影响原数组
- 但`userP`是指针，指向的内存地址相同，所以通过指针修改的数据会影响原对象
- 这体现了**值传递**和**引用传递**的本质区别

三、函数特性

学到的核心知识

1. **函数定义的灵活性**
    - 支持多返回值

- 命名返回值可以直接return
- 不定参数`...type`的使用

2. **函数作为一等公民**
   - 函数可以作为类型：`type FuncType func(int, int) int`
   - 函数可以作为参数和返回值
3. **匿名函数与闭包**
   - 匿名函数的多种定义方式
   - 闭包捕获外部变量的特性
4. **defer延迟调用**
   - LIFO（后进先出）执行顺序
   - 常用于资源释放

理解的难点

难点1：不定参数的传递

```go
func Test(args ...int) {
    MyFunc01(args...)       // 原样传递
    MyFunc02(args[1:]...)   // 传递部分参数
}
```

理解要点：使用`...`展开切片，可以灵活传递不定参数。

难点2：闭包的变量捕获

```go
func squares() func() int {
    var x int
    return func() int {
```

```
        x++  // 捕获外部变量x
        return x * x
    }
}

f := squares()
fmt.Println(f())  // 1
fmt.Println(f())  // 4  (x被保留了！)
```

理解要点：

- 闭包会"捕获"外部变量，即使外层函数已返回
- 每次调用 `squares()` 会创建新的 `x` 变量
- 变量的生命周期不由作用域决定，而是由闭包引用决定

难点3：defer与匿名函数的结合

```
func main() {
    a, b := 10, 20
    defer func(x int) {
        fmt.Println(x, b)  // x=10(值传递), b=120(闭包引用)
    }(a)

    a += 10
    b += 100
}
```

理解要点：

- defer的参数在声明时就已经确定（值传递）
- 但闭包引用的变量会在defer执行时才取值

难点4：作用域的就近原则
在不同作用域可以声明同名变量，访问时遵循就近原则：

```
var a int  // 全局

func main() {
    var a uint8  // 局部
    {
        var a float64  // 更局部
        fmt.Printf("%T", a)  // float64
    }
    fmt.Printf("%T", a)  // uint8
}
```

四、文本与数据处理

学到的核心知识

1. **字符串处理（strings包）**
    - Contains、Join、Index、Split等常用函数
    - Trim、Replace字符串操作
    - Fields按空格分割
2. **类型转换（strconv包）**
    - Format系列：其他类型转字符串
    - Parse系列：字符串转其他类型

- Append系列：追加到字节数组

3. **正则表达式（regexp包）**
   - `MustCompile` 编译正则表达式
   - `FindAllStringSubmatch` 查找所有匹配
   - 分组捕获 `(.*?)` 的使用
4. **JSON处理（encoding/json包）**
   - 结构体与JSON的相互转换
   - 编码和解码操作

理解的难点

难点1：正则表达式的贪婪与非贪婪

```
// 非贪婪匹配
exp := regexp.MustCompile(`<div>(.*?)</div>`)

// 跨行匹配需要(?s:)
exp := regexp.MustCompile(`<div>(?s:(.*?))</div>`)
```

理解要点：

- `.*?` 是非贪婪匹配，尽可能少地匹配
- `(?s:)` 标志使 `.` 可以匹配换行符

难点2：字符串不可变性

Go中字符串是不可变的，修改字符串需要转换为 `[]byte` 或 `[]rune`：

```
str := "hello"
// str[0] = 'H'  // 错误!
```

```
bytes := []byte(str)
bytes[0] = 'H'
str = string(bytes)  // 正确
```

五、核心理解与收获

1. Go的设计哲学

- **简洁性**：语法简单，没有复杂的继承体系
- **显式性**：类型转换必须显式，错误处理必须显式
- **实用性**：内置并发支持，丰富的标准库

2. 值类型 vs 引用类型

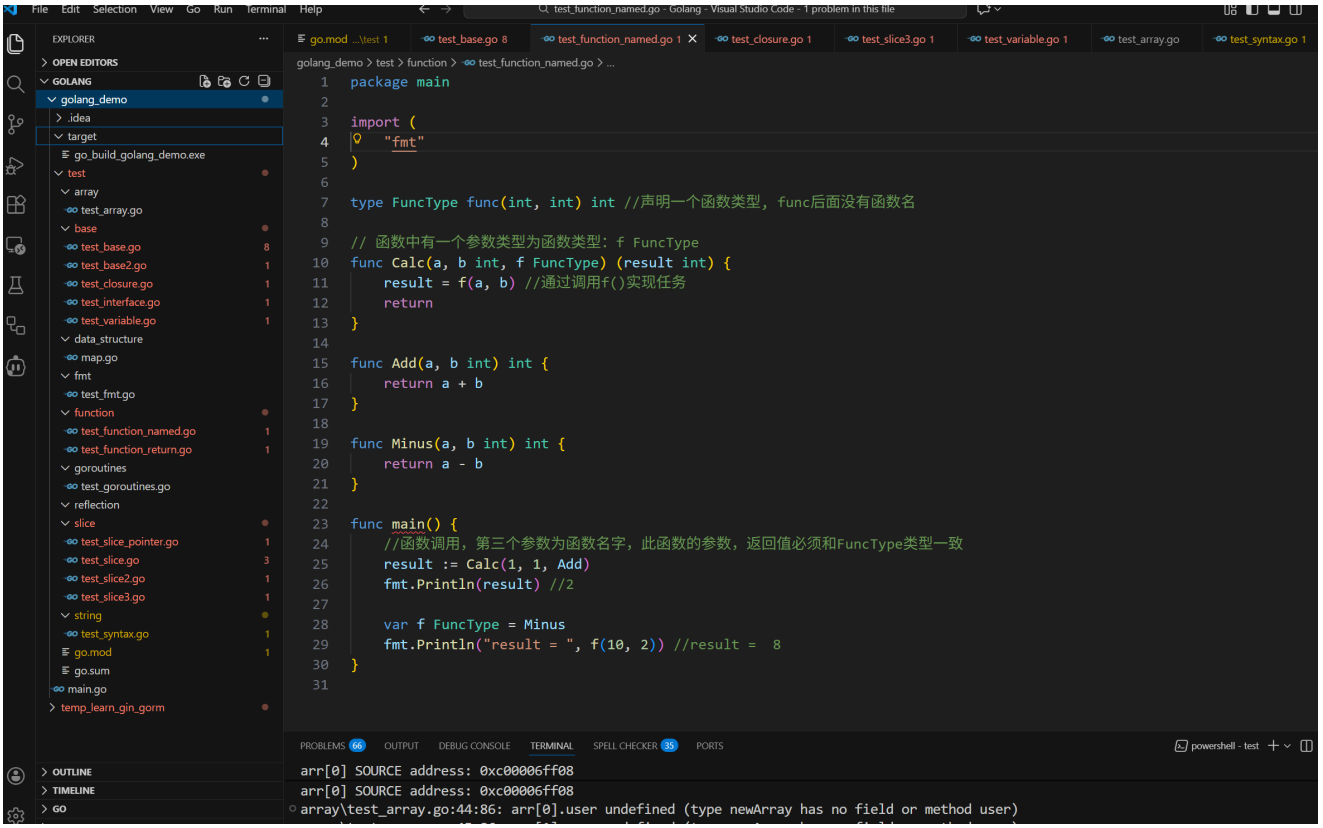
- **值类型**：基本类型、数组、结构体（传递时复制）
- **引用类型**：切片、map、channel、指针（传递时共享）

3. 内存管理

- Go有自动垃圾回收，无需手动释放内存
- `new()` 和 `make()` 的区别（虽然文档未详细展开）
- defer机制保证资源正确释放

4. 函数式编程特性

- 函数是一等公民
- 闭包提供了强大的抽象能力
- 匿名函数简化代码结构


File Edit Selection View Go Run Terminal Help
test_function_named.go - Golang - Visual Studio Code - 1 problem in this file
EXPLORER
OPEN EDITORS
GOLANG
golang_demo
.idea
target
go_build_golang_demo.exe
test
array
test_array.go
base
test_base.go 8
test_base2.go 1
test_closure.go 1
test_interface.go 1
test_variable.go 1
data_structure
map.go
fmt
test_fmt.go
function
test_function_named.go 1
test_function_return.go 1
goroutines
test_goroutines.go
reflection
slice
test_slice_pointer.go 1
test_slice.go 3
test_slice2.go 1
test_slice3.go 1
string
test_syntax.go 1
go.mod 1
go.sum
main.go
temp_learn_gin_gorm
OUTLINE
TIMELINE
GO
go.mod ...\test 1
test_base.go 8
test_function_named.go 1
test_closure.go 1
test_slice3.go 1
test_variable.go 1
test_array.go
test_syntax.go 1
golang_demo > test > function > test_function_named.go > ...
1 package main
2
3 import (
4 "fmt"
5)
6
7 type FuncType func(int, int) int //声明一个函数类型, func后面没有函数名
8
9 // 函数中有一个参数类型为函数类型: f FuncType
10 func Calc(a, b int, f FuncType) (result int) {
11 result = f(a, b) //通过调用f()实现任务
12 return
13 }
14
15 func Add(a, b int) int {
16 return a + b
17 }
18
19 func Minus(a, b int) int {
20 return a - b
21 }
22
23 func main() {
24 //函数调用, 第三个参数为函数名字, 此函数的参数, 返回值必须和FuncType类型一致
25 result := Calc(1, 1, Add)
26 fmt.Println(result) //2
27
28 var f FuncType = Minus
29 fmt.Println("result = ", f(10, 2)) //result = 8
30 }
31
PROBLEMS 66
OUTPUT
DEBUG CONSOLE
TERMINAL
SPELL CHECKER 35
PORTS
powershell - test
arr[0] SOURCE address: 0xc00006ff08
arr[0] SOURCE address: 0xc00006ff08
array\test_array.go:44:86: arr[0].user undefined (type newArray has no field or method user)