Day2 流程控制&容器

tony

目录

1. 字符串详解

2. 流程控制

3. 数组和切片

4. 课后作业

字符串原理解析

1. 字符串底层就是一个byte数组，所以可以和[]byte类型互相转换

2. 字符串之中的字符是不能修改的，那怎么修改呢

3. 字符串是由byte字节组成，所以字符串的长度是byte字节的长度

4. rune类型用来表示utf8字符，一个rune字符由1个或多个byte组成

练习

练习1：写一个程序，对英文字符串进行逆序。

练习2：写一个程序，对包含中文的字符串进行逆序。

练习3：写一个程序，判断一个字符串是否是回文。

时间和日期类型

1. time包

2. time.Time类型，用来表示时间

3. **获取当前时间，** now := time.Now()

4. time.Now().Day()，time.Now().Minute()，time.Now().Month()，time.Now().Year()

5. **格式化，**fmt.Printf(“%02d/%02d%02d %02d:%02d:%02d”, now.Year()…)

时间和日期类型

6. 获取当前时间戳，time.Now().Unix()。

7. 时间戳转Time类型。

8. 定时器的简单使用

时间和日期类型

6. time.Duration用来表示纳秒

7. 一些常量：

```
const (
        Nanosecond  Duration = 1
        Microsecond          = 1000 * Nanosecond
        Millisecond          = 1000 * Microsecond
        Second               = 1000 * Millisecond
        Minute               = 60 * Second
        Hour                 = 60 * Minute
)
```

8. 格式化：

```
now := time.Now()
fmt.Println(now.Format(“02/1/2006 15:04”))
fmt.Println(now.Format(“2006/1/02 15:04”))
fmt.Println(now.Format(“2006/1/02”))
```

时间和日期类型

练习1：写一个程序，获取当前时间，并格式化成 2017/06/15 08:05:00形式

练习2：写一个程序，统计一段代码的执行耗时，单位精确到微秒。

if else语句

1. 基本语法

```
if condition {
  //do something
}
```

```
if condition {
  //do something
} else if condition {
  //do something
} else {
  //do something
}
```

if else语句

2. 练习一

```
package main

import (
    "fmt"
)

func main() {
    num := 10
    if num % 2 == 0 { //checks if number is even
        fmt.Println("the number is even")
    }  else {
        fmt.Println("the number is odd")
    }
}
```

if else语句

3. 基本语法

```
if statement; condition {

}
```

if else语句

4. 练习二

```
package main

import (
    "fmt"
)


func main() {
    if num := 10; num % 2 == 0 { //checks if number is even
        fmt.Println(num,"is even")
    }  else {
        fmt.Println(num,"is odd")
    }
}
```

if else语句

5. 练习三

```
package main

import (
    "fmt"
)

func main() {
    num := 99
    if num <= 50 {
        fmt.Println("number is less than or equal to 50")
    } else if num >= 51 && num <= 100 {
        fmt.Println("number is between 51 and 100")
    } else {
        fmt.Println("number is greater than 100")
    }

}
```

循环

1. Go语言中只有一种循环 for

```
for initialisation; condition; post {

}
```

循环

2. 练习一

```
package main

import (
    "fmt"
)

func main() {
    for i := 1; i <= 10; i++ {
        fmt.Printf(" %d",i)
    }
}
```

循环

3. break，终止循环

```
package main

import (
   "fmt"
)

func main() {
   for i := 1; i <= 10; i++ {
      if i > 5 {
         break //loop is terminated if i > 5
      }
      fmt.Printf("%d ", i)
   }
   fmt.Printf("\nline after for loop")
}
```

循环

4. continue，终止本次循环

```go
package main

import (
    "fmt"
)

func main() {
    for i := 1; i <= 10; i++ {
        if i%2 == 0 {
            continue
        }
        fmt.Printf("%d ", i)
    }
}
```

循环

5. 练习一

```
package main

import (
    "fmt"
)

func main() {
    i := 0
    for ;i <= 10; { // initialisation and post are omitted
        fmt.Printf("%d ", i)
        i += 2
    }
}
```

循环

6. 练习二

```
package main

import (
    "fmt"
)

func main() {
    i := 0
    for i <= 10 { // initialisation and post are omitted
        fmt.Printf("%d ", i)
        i += 2
    }
}
```

循环

7. 练习三

```
package main

import (
    "fmt"
)

func main() {
    for no, i := 10, 1; i <= 10 && no <= 19; i, no = i+1, no+1 {
        fmt.Printf("%d * %d = %d\n", no, i, no*i)
    }

}
```

循环

8. 无限循环

```
package main

import (
    "fmt"
)

func main() {
    for {
        fmt.Printf("hello")
    }

}
```

switch语句

1. switch

```
package main

import (
    "fmt"
)

func main() {
    finger := 4
    switch finger {
    case 1:
        fmt.Println("Thumb")
    case 2:
        fmt.Println("Index")
    case 3:
        fmt.Println("Middle")
    case 4:
        fmt.Println("Ring")
    case 5:
        fmt.Println("Pinky")

    }
}
```

switch语句

2. Switch default

```
package main

import (
    "fmt"
)

func main() {
    switch finger := 8; finger {
    case 1:
        fmt.Println("Thumb")
    case 2:
        fmt.Println("Index")
    case 3:
        fmt.Println("Middle")
    case 4:
        fmt.Println("Ring")
    case 5:
        fmt.Println("Pinky")
    default: //default case
        fmt.Println("incorrect finger number")
    }
}
```

switch语句

3. Switch

```
package main

import (
    "fmt"
)

func main() {
    letter := "i"
    switch letter {
    case "a", "e", "i", "o", "u":
        fmt.Println("vowel")
    default:
        fmt.Println("not a vowel")
    }
}
```

switch语句

4. Switch case 条件判断

```
package main

import (
    "fmt"
)

func main() {
    num := 75
    switch { // expression is omitted
    case num >= 0 && num <= 50:
        fmt.Println("num is greater than 0 and less than 50")
    case num >= 51 && num <= 100:
        fmt.Println("num is greater than 51 and less than 100")
    case num >= 101:
        fmt.Println("num is greater than 100")
    }

}
```

switch语句

5. Switch fallthrough

```
package main
import (
    "fmt"
)
func number() int {
        num := 15 * 5
        return num
}
func main() {
    switch num := number(); { //num is not a constant
    case num < 50:
        fmt.Printf("%d is lesser than 50\n", num)
        fallthrough
    case num < 100:
        fmt.Printf("%d is lesser than 100\n", num)
        fallthrough
    case num < 200:
        fmt.Printf("%d is lesser than 200", num)
    }
}
```

数组定义

1. 数组是同一类型的元素集合。

```
var a [3]int
        //定义一个数组
```

Go中数组下标从0开始，因此长度为n的数组下标范围：[0,n-1]

整数数组中的元素默认初始化为0，字符串数组中的元素默认初始化为""

数组定义

2. 数组初始化

```
var a [3]int
a[0] = 10
a[1] = 20
a[2] = 30
    //数组初始化
```

```
var a [3]int = [3]int{10, 20, 30}
    //定义时数组初始化
```

```
a := [3]int{10, 20, 30}
    //定义时数组初始化
```

```
a := [...]int{10, 20, 30}
    //定义时数组初始化
```

```
a := [3]int{10}
    //定义时数组初始化
```

```
a := [3]int{2:10}
    //定义时数组初始化
```

数组定义

3. 数组长度是类型的一部分

```
var a [3]int
a[0] = 10
a[1] = 20
a[2] = 30

var b [5]int
b = a
        //a、b是不同类型的数组，不能赋值
```

数组定义

4. len内置函数

```
var a [3]int
a[0] = 10
a[1] = 20
a[2] = 30

fmt.Printf(“len:%d\n”, len(a))
```

数组定义

5. 数组遍历

```
var a [3]int
a[0] = 10
a[1] = 20
a[2] = 30

for i := 0; i < len(a); i++ {
}
```

//a、b是不同类型的数组，不能赋值

数组定义

6. 数组遍历

```
var a [3]int
a[0] = 10
a[1] = 20
a[2] = 30

for index, val :=  range a {
}
```

//a、b是不同类型的数组，不能赋值

二维数组

6. 二维数组

```
var a [3][2]int
a[0][0] = 10
a[0][1] = 20
a[1][0] = 30
a[1][1] = 30
a[2][0] = 30
a[2][1] = 30
for index, val :=  range a {
}
```

```go
package main
import (
    "fmt"
)
func printarray(a [3][2]string) {
    for _, v1 := range a {
        for _, v2 := range v1 {
            fmt.Printf("%s ", v2)
        }
        fmt.Printf("\n")
    }
}
func main() {
    a := [3][2]string{
        {"lion", "tiger"},
        {"cat", "dog"},
        {"pigeon", "peacock"},
    }
    printarray(a)
    var b [3][2]string
    b[0][0] = "apple"
    b[0][1] = "samsung"
    b[1][0] = "microsoft"
    b[1][1] = "google"
    b[2][0] = "AT&T"
    b[2][1] = "T-Mobile"
    fmt.Printf("\n")
    printarray(b)
}
```

数组拷贝和传参

1. 数组是值类型

```
var a [3]int
a[0] = 10
a[1] = 20
a[2] = 30

b := a
//b拷贝了数组a中所有元素
b[0] = 1000

fmt.Println(a, b)
```

数组拷贝和传参

2. 数组是值类型，函数传参也会拷贝

```
func main() {
        var a [3]int
        a[0] = 10
        a[1] = 20
        a[2] = 30
        modify(a)
        fmt.Println(a)
}

func modify(b [3]int) {
   b[0] = 1000
        return
}
```

切片定义

1. 切片是基于数组类型做的一层封装。它非常灵活，可以自动扩容。

```
var a []int
        //定义一个int类型的空切片
```

切片定义

2. **切片初始化**, a[start:end]**创建一个包括从**start**到**end-1**的切片。**

```
package main


import (
    "fmt"
)

func main() {
    a := [5]int{76, 77, 78, 79, 80}
    var b []int = a[1:4] //基于数组a创建一个切片，包括元素a[1] a[2] a[3]
    fmt.Println(b)
}
```

切片定义

3. 切片初始化方法2。

```
package main

import (
    "fmt"
)

func main() {
    c := []int{6, 7, 8} //创建一个数组并返回一个切片
    fmt.Println(c)
}
```

切片基本操作

5. 数组切片的基本操作

a) arr[start:end]：包括start到end-1(包括end-1)之间的所有元素

b) arr[start:]：包括start到arr最后一个元素(包括最后一个元素)之间的所有元素

c) arr[:end]：包括0到end-1(包括end-1）之间的所有元素

d) arr[:]：包括整个数组的所有元素

切片基本操作

4. 切片修改

```
package main

import (
    "fmt"
)

func main() {
    //创建一个数组，其中[...]是编译器确定数组的长度,darr的长度是9
    darr := [...]int{57, 89, 90, 82, 100, 78, 67, 69, 59}
    //基于darr创建一个切片dslice,包括darr[2],darr[3],darr[4]三个元素
    dslice := darr[2:5]
    fmt.Println("array before",darr)
    for i := range dslice {
        //对于dslice中每个元素进行+1，其实修改是darr[2],darr[3],darr[4]
        dslice[i]++
    }
    fmt.Println("array after",darr)
}
```

切片基本操作

6. 切片修改

```
package main

import (
    "fmt"
)

func main() {
    numa := [3]int{78, 79 ,80}
    //创建一个切片，包含整个数组的所有元素
    nums1 := numa[:]
    nums2 := numa[:]
    fmt.Println("array before change 1",numa)
    nums1[0] = 100
    fmt.Println("array after modification to slice nums1", numa)
    nums2[1] = 101
    fmt.Println("array after modification to slice nums2", numa)
}
```

切片基本操作

7. 使用make创建切片

```go
package main


import (
    "fmt"
)

func main() {
    //[]中没有长度
    i := make([]int, 5, 5)
    fmt.Println(i)
}
```

切片基本操作

8. 切片的长度和容量

```
package main

import (
   "fmt"
)

func main() {
   fruitarray := [...]string{
                  "apple", "orange", "grape",
                  "mango", "water melon",
                  "pine apple", "chikoo"}
   fruitslice := fruitarray[1:3]
   //长度是2，容量is 6
   fmt.Printf("length of slice %d capacity %d",
       len(fruitslice), cap(fruitslice))
}
```

切片基本操作

9. 切片的再切片

```
package main

import (
    "fmt"
)
func main() {
    fruitarray := [...]string{
          "apple", "orange", "grape", "mango",
          "water melon", "pine apple", "chikoo"}
    fruitslice := fruitarray[1:3]
    //长度是2, 容量是6
    fmt.Printf("length of slice %d capacity %d\n",
         len(fruitslice), cap(fruitslice))
    //再重新进行切片，不能大于数组fruitarray的长度，否则越界
    fruitslice = fruitslice[:cap(fruitslice)]
    fmt.Println("After re-slicing length is”,
        len(fruitslice), "and capacity is",cap(fruitslice))
}
```

切片基本操作

10.append操作

```
package main

import (
    "fmt"
)

func main() {
    cars := []string{"Ferrari", "Honda", “Ford"}
    //长度和容量都等于3
    fmt.Println("cars:", cars, "has old length",
        len(cars), "and capacity", cap(cars))
    cars = append(cars, “Toyota")
    //容量等于6
    fmt.Println("cars:", cars, "has new length",
        len(cars), "and capacity", cap(cars))
}
```

切片基本操作

11. 空切片

```
package main
import (
    "fmt"
)

func main() {
    //定义names是一个空切片，长度和容量都等于0
    //不能对空切片进行访问，否则panic
    var names []string
    if names == nil {
        fmt.Println("slice is nil going to append")
        names = append(names, "John", "Sebastian", "Vinay")
        fmt.Println("names contents:",names)
    }
}
```

切片基本操作

12. append一个切片

```
package main


import (
    "fmt"
)

func main() {
    veggies := []string{"potatoes","tomatoes","brinjal"}
    fruits := []string{“oranges","apples"}
    //fruits后面的3个点表示展开fruits切片成一个个元素
    food := append(veggies, fruits...)
    fmt.Println("food:",food)
}
```

切片传参

13. 切片传参

```
package main

import (
    "fmt"
)
//在函数内部修改numbers切片的值
func subtactOne(numbers []int) {
    for i := range numbers {
        numbers[i] -= 2
    }
}
func main() {
    nos := []int{8, 7, 6}
    fmt.Println("slice before function call", nos)
    subtactOne(nos)
    //nos修改生效了，说明切片是引用类型
    fmt.Println("slice after function call", nos)
}
```

切片定义

14. 切片拷贝

```
package main

import (
    "fmt"
)

func main() {
    veggies := []string{"potatoes","tomatoes","brinjal"}
    fruits := []string{“oranges","apples"}
    copy(veggies, fruits)
    fmt.Println(veggies, fruits)
}
```

切片遍历

15.切片遍历

```
var a [3]int
a[0] = 10
a[1] = 20
a[2] = 30
B := a[:]

for index, val :=  range b {
}
      //和数组遍历是一样的
```

make和new区别

16. make为内建类型slice、map和channel分配内存。

```
//初始化一个切片
s := make([]int, 10, 30)
```

17. new用于各种类型的内存分配，new返回是一个指针。

课后作业

1. 求数组所有元素之和

2. 找出数组中和为给定值的两个元素的下标，比如数组:[1,3,5,8,7]，找出两个元素之和等于8的下标分别是(0, 4)和(1,2)。

课后作业

1. 下列程序输出什么？

```
func main() {
    var sa = make ([]string,5,10);

    for i:=0;i<10;i++{
        sa=append(sa, fmt.Sprintf("%v",i))
    }
    fmt.Println(sa);
}
```

2. 使用golang标准包 “sort”对数组进行排序

3. 实现一个密码生成工具，支持以下功能：

提示：可以用标准包 “flag”解析命令行参数

a) 用户可以通过-l指定生成密码的长度

b) 用户可以通过-t指定生成密码的字符集，比如-t num生成全数字的密码
-t char 生成包含全英文字符的密码，-t mix包含生成数字和英文的密码，
-t advance 生成包含数字、英文以及特殊字符的密码

QA