第一节课：go语言的基础语法（包、变量、函数、分支与循环、容器）

go语言是什么

Golang(又称Go)是Google公司开发的一种静态强类型、编译型、并发型，并具有垃圾回收功能的编程语言。它经常被比喻成21世纪的C语言。**Go**作为Google2009年推出的语言，其被设计成一门应用于搭载 Web 服务器，存储集群或类似用途的巨型中央服务器的系统编程语言。

同时我们网校这边的后端，也是由**Go**去编写的（

接下来我将讲解一些Go的基本语法内容，包括如下部分

- 包
- 变量
- 函数
- 分支与循环
- 容器

Start

Go语言的程序由一个或多个源文件组成，每个源文件都以`.go`作为拓展名。源文件的第一行必须是`package`声明，指明这个文件属于哪个包，一般是为main。

一个简单的go语言程序`Hello World`如下：

```
// hello.go
package main // 声明该文件属于main包

import "fmt" // 导入fmt包，提供输入输出功能

func main() { // 定义main函数，是程序的入口点
    fmt.Println("Hello, world!") // 调用fmt包的Println函数，打印一行文本到标准输出
}
```

要运行这个程序，需要先安装Go语言环境，相信你已经装好了（

然后再terminal中输入以下命令

```
go run hello.go # 编译并运行这个文件
```

当然你如果想持久化，你可以使用build子命令

```
go build hello.go # 编译并生成一个二进制文件
./hello #执行这个二进制文件
```

包

包是Go语言的重要组成单位，包的设计目的是为了简化程序的设计和维护。

每一个包都设定了一个命名空间用于去访问。每个包都是由一个全局唯一的字符串所标识的导入路径单位，并出现在**import**语句中。

正如**上面的Hello World程序**中提到`fmt`就是一个包。这个包是Go语言的标准库包，当然你也可以自己编写一个包。

自己编写一个包

编写一个包有如下的步骤

1. 创建一个目录存放你所编写的源文件，这个目录的名字就是你的包名字（当然也可以使用包的别名，即在调用包的时候赋予别名）

   ```
   import alias "package/path"
   ```

   `alias.`即可作为你的前缀去调用内容
2. 在目录中创建一个或多个文件，每个源文件第一行都必须存在`package`声明，声明这个文件属于哪一个包
3. 在源文件中编写你所需的内容（通常为变量、常量、类型、函数，为了使这些可以导出，其名字应为大写字母开头）
4. 在其他源文件调用这个包，使用`import`去调用包（注意不能**循环调用**）
5. 使用该包的功能时一般为`包名.`作为前缀去调用包的内容

example:

编写一个`circle`包，这个包可以用于计算◯的面积和周长

1. 创建一个名为`circle`的目录
2. 在该目录下创建一个名为`circle.go`的源文件
3. 在源文件中编写如下内容

```
// circle.go
package circle // 声明该文件属于circle包

import "math" // 导入math包

const Pi = math.Pi // 定义一个常量Pi，等于math包中的Pi

func Area(r float64) float64 { // 定义一个函数Area，接受一个float64类型的参数r，表示圆的半径，返回一个float64类型的结果，表示圆的面积
    return Pi * r * r // 返回Pi乘以r的平方
}

func Circumference(r float64) float64 { // 定义一个函数Circumference，接受一个float64类型的参数r，表示圆的半径，返回一个float64类型的结果，表示圆的周长
```

```
    return 2 * Pi * r // 返回2乘以Pi乘以r
}
```

4. 在其他地方导入所创建的包，例如在`main.go`文件中

```
// main.go
package main // 声明该文件属于main包

import (
    "fmt" // 导入fmt包
    "circle" // 导入circle包，注意要指定完整的路径，这里假设circle目录和main.go文件在同一级目录下
)

func main() {
    r := 10.0 // 定义一个变量r，赋值为10.0
    fmt.Println(circle.Area(r)) // 调用circle包的Area函数，传入r作为参数，打印返回值
    fmt.Println(circle.Circumference(r)) // 调用circle包的Circumference函数，传入r作为参数，打印返回值
}
```

包的匿名导入

前面已经说明了包是如何导入的，但是包还有另外一种导入方式， 名为**匿名导入**

即在包的导入路径前添加下划线`_`来引用一个包，但是不直接使用包的内容，一般这种导入方式常用于包的副作用，即在包导入时就完成一些初始化行为以及抑制`unused import`编译错误的出现。

常用的标准库

- `time`包：提供了时间和日期的处理功能，例如获取当前时间、格式化时间、计算时间差等。例如：

```
import "time"

func main() {
    now := time.Now() // 获取当前时间
    fmt.Println(now) // 打印当前时间
    fmt.Println(now.Year()) // 打印当前年份
    fmt.Println(now.Month()) // 打印当前月份
    fmt.Println(now.Day()) // 打印当前日期
    fmt.Println(now.Hour()) // 打印当前小时
    fmt.Println(now.Minute()) // 打印当前分钟
    fmt.Println(now.Second()) // 打印当前秒数

    tomorrow := now.Add(24 * time.Hour) // 获取明天的时间，即当前时间加上24小时
    fmt.Println(tomorrow) // 打印明天的时间

    format := "2006-01-02 15:04:05" // 定义一个时间格式，注意要使用特定的数字
    fmt.Println(now.Format(format)) // 将当前时间按照指定的格式输出
}
```

- `strconv`包：提供了字符串和基本数据类型之间的转换功能，例如将字符串转换为整数、浮点数、布尔值等，或者将整数、浮点数、布尔值等转换为字符串。例如：

```go
import "strconv"

func main() {
    s := "10" // 定义一个字符串变量s，赋值为"10"
    i, err := strconv.Atoi(s) // 将字符串s转换为整数i，返回转换结果和可能的错误
    if err != nil { // 如果有错误
        log.Fatal(err) // 处理错误
    }
    fmt.Println(i) // 打印i的值，即10

    f, err := strconv.ParseFloat(s, 64) // 将字符串s转换为64位浮点数f，返回转换结果和可能的错误
    if err != nil { // 如果有错误
        log.Fatal(err) // 处理错误
    }
    fmt.Println(f) // 打印f的值，即10.0

    b, err := strconv.ParseBool(s) // 将字符串s转换为布尔值b，返回转换结果和可能的错误
    if err != nil { // 如果有错误
        log.Fatal(err) // 处理错误
    }
    fmt.Println(b) // 打印b的值，即true

    s = strconv.Itoa(i) // 将整数i转换为字符串s，返回转换结果
    fmt.Println(s) // 打印s的值，即"10"

    s = strconv.FormatFloat(f, 'f', 2, 64) // 将64位浮点数f转换为字符串s，指定格式为'f'（十进制），精度为2（保留两位小数），返回转换结果
    fmt.Println(s) // 打印s的值，即"10.00"

    s = strconv.FormatBool(b) // 将布尔值b转换为字符串s，返回转换结果
    fmt.Println(s) // 打印s的值，即"true"
}
```

- `fmt`包：提供了格式化输入输出的功能，例如打印数据到标准输出、读取数据从标准输入、格式化字符串等。我们已经在前面介绍过`fmt`包中的`Println`和`Printf`函数，它们都可以将数据打印到标准输出，即屏幕上。除此之外，`fmt`包中还有以下几个常用的函数：
    - `Sprint`和`Sprintf`函数：它们与`Print`和`Printf`函数类似，但是不会将数据打印到标准输出，而是返回一个格式化后的字符串。例如：

```go
x := 10
y := 3.14
z := "Hello"
s := fmt.Sprint(x, y, z) // 返回一个由x, y, z组成的字符串，并用空格分隔，结果为"10 3.14 Hello"
fmt.Println(s) // 打印s的值

s = fmt.Sprintf("x = %d, y = %.2f, z = %s", x, y, z) // 返回一个按照指定格式输出x, y, z的字符串，结果为"x = 10, y = 3.14, z = Hello"
fmt.Println(s) // 打印s的值
```

    - `Scan`和`Scanf`函数：它们可以从标准输入，即键盘上，读取数据，并存储到指定的变量中。例如：

```go
var x int
var y float64
var z string
fmt.Scan(&x, &y, &z) // 从标准输入读取三个值，并分别赋给x, y, z，要求用空格分隔输入的值
fmt.Println(x, y, z) // 打印x, y, z的值

fmt.Scanf("%d %f %s", &x, &y, &z) // 从标准输入读取三个值，并按照指定的格式赋给x, y, z，要求用空格分隔输入的值
fmt.Println(x, y, z) // 打印x, y, z的值
```

更多关于这些包和函数的内容，请参考官方文档。

变量

Go语言中，变量是用来存储数据的容器，它有一个名字和一个类型。变量的名字必须遵循标识符的规则，即由字母、数字和下划线组成，且不能以数字开头。变量的类型决定了变量可以存储什么样的数据，以及占用多少内存空间。

Go语言中有25个关键字，它们有特殊的含义，不能作为变量名或其他标识符使用。这些关键字如下：

```
break    default      func    interface    select
case     defer        go      map          struct
chan     else         goto    package      switch
const    fallthrough  if      range        type
continue for          import  return       var
```

Go语言中还有37个预定义的标识符，它们是一些内置的常量、类型和函数。这些标识符可以被重新定义，但不建议这样做。这些标识符如下：

```
// 常量
true  false  iota  nil

// 类型
int  int8  int16  int32  int64
uint uint8 uint16 uint32 uint64 uintptr
float32 float64 complex128 complex64
bool byte rune string error

// 函数
make len cap new append copy close delete
complex real imag
panic recover
```

要声明一个变量，可以使用`var`关键字，后跟变量名和类型。

example

```go
var x int//声明一个整数类型的变量x
var y string //声明一个字符串类型的变量y
```

声明变量时，可以给变量赋值一个初始值，这样无需单独赋值

example

```
ar x int = 10 // 声明一个名为x的整型变量，并赋值为10
var y string = "Hello" // 声明一个名为y的字符串变量，并赋值为"Hello"
```

如果声明变量时给出了初始值，那么可以省略类型，让编译器自动推断类型。

example

```
var x = 10 // 声明一个名为x的变量，并赋值为10，编译器推断其类型为int
var y = "Hello" // 声明一个名为y的变量，并赋值为"Hello"，编译器推断其类型为string
```

果要声明多个变量，可以使用逗号分隔，或者使用括号包裹。

example

```
var x, y, z int // 声明三个整型变量x, y, z
var (
    a string // 声明一个字符串变量a
    b bool // 声明一个布尔变量b
    c float64 // 声明一个浮点数变量c
)
```

如果要给多个变量同时赋值，可以使用平行赋值的语法，即用逗号分隔变量名和值，然后用等号连接。

example

```
x, y, z = 1, 2, 3 // 给三个整型变量x, y, z分别赋值为1, 2, 3
a, b = b, a // 交换两个变量的值
```

Go语言还可以使用短变量声明的语法，即用`:=`代替`var`和类型。这样可以简化代码，但只能用于声明新的变量，不能用于修改已有的变量。

example

```
func main() {
    x := 10 // 等价于 var x int = 10
    y := "Hello" // 等价于 var y string = "Hello"
    x, y := 20, "World" // 等价于 var x int = 20; var y string = "World"
    x := 30 // 错误，不能重新声明已有的变量x
}
```

如果声明了一个变量，但没有给它赋值，那么它会被初始化为零值。零值是每种类型的默认值，通常是该类型的空或无效状态。不同类型的零值如下：

- 数字类型（整数、浮点数、复数）的零值是`0`
- 布尔类型的零值是`false`
- 字符串类型的零值是`""`（空字符串）
- 指针、函数、接口、切片、通道和映射类型的零值是`nil`

Go语言中有多种数字类型，它们分为整数、浮点数和复数三类。每种类型都有不同的长度和范围，如下表所示：

类型	长度（字节）	范围
int8	1	-128 ~ 127
int16	2	-32768 ~ 32767
int32	4	-2147483648 ~ 2147483647
int64	8	-9223372036854775808 ~ 9223372036854775807
uint8（无符号）	1	0 ~ 255
uint16（无符号）	2	0 ~ 65535
uint32（无符号）	4	0 ~ 4294967295
uint64（无符号）	8	0 ~ 18446744073709551615
float32	4	$\pm1.18\times10^{-38}$ ~ $\pm3.4\times10^{38}$
float64	8	$\pm2.23\times10^{-308}$ ~ $\pm1.80\times10^{308}$
byte		类似uint
rune		类似uint32
bool		只能是true(0)或者false(1)
complex64	8	
complex128	16	

复数是由实部和虚部组成的数，例如`3 + 4i`。复数类型的长度是实部和虚部的长度之和，范围是实部和虚部的范围之积。Go语言中有两种复数类型：`complex64`和`complex128`，它们的实部和虚部分别是`float32`和`float64`类型。

除了上表中的类型，Go语言还有两种通用的整数类型：`int`和`uint`，它们的长度取决于操作系统的位数，即32位系统为4字节，64位系统为8字节。另外，还有一种特殊的整数类型：`uintptr`，它用于存储指针值，其长度也与操作系统的位数相同。

运算符

运算符用于在程序运行时执行数学或逻辑运算。我们在数学中使用的`+、-、×、÷、=`其实在计算机当中都算作是运算符。

在Go中内置的运算符有以下几类

- 算术运算符
- 关系运算符
- 逻辑运算符
- 位运算符
- 赋值运算符
- 其他运算符

算术运算符

运算符	描述	Example
+	相加	1 + 2 输出结果为 3
-	相减	3 - 2 输出结果为 1
*	相乘	2 * 3 输出结果为 6
/	相除	6 / 2 输出结果为3
%	求余，取模	7 % 2 输出结果为 1
++	自增	i := 1 i++ 输出i的值为2
--	自减	与`++`同理

关系运算符

运算符	描述	Example
==	检查两个值是否相等，如果相等返回`true`否则返回`false`	(1 == 2) 为`false`
!=	检查两个值是否不相等，如果相等返回`true`否则返回`false`	(1 != 2) 为`true`
>	检查左边值是否大于右边值，如果是返回`true`否则返回`false`	(1 > 2) 为`false`
<	检查左边值是否小于右边值，如果是返回`true`否则返回`false`	(1 < 2) 为`true`
>=	检查左边值是否大于等于右边值，如果是返回`true`否则返回`false`	(1 >= 2) 为`false`
<=	检查左边值是否小于等于右边值，如果是返回`true`否则返回`false`	(1 <= 2) 为`true`

逻辑运算符

运算符	描述	Example（A是trueB是false)
&&	逻辑 AND 运算符。 如果两边的操作数都是`true`，则条件`true`，否则为`false`	(A && B) 为`false`
||	逻辑 OR 运算符。 如果两边的操作数有一个`true`，则条件`true`，否则为`false`	(A || B) 为`true`
!	逻辑 NOT 运算符。 如果条件为`true`，则逻辑 NOT 条件`false`，否则为`true`	!(A && B) 为`true`

位运算符

运算符	描述	Example
&	按位与运算符"&"是双目运算符。 其功能是参与运算的两数各对应的二进位相与。	(A & B) 结果为 12, 二进制为 0000 1100
|	按位或运算符"|"是双目运算符。 其功能是参与运算的两数各对应的二进位相或	(A | B) 结果为 61, 二进制为 0011 1101
^	按位异或运算符"^"是双目运算符。 其功能是参与运算的两数各对应的二进位相异或，当两对应的二进位相异时，结果为1。	(A ^ B) 结果为 49, 二进制为 0011 0001
<<	左移运算符"<<"是双目运算符。左移n位就是乘以2的n次方。 其功能把"<<"左边的运算数的各二进位全部左移若干位，由"<<"右边的数指定移动的位数，高位丢弃，低位补0。	A << 2 结果为 240 ，二进制为 1111 0000
>>	右移运算符">>"是双目运算符。右移n位就是除以2的n次方。 其功能是把">>"左边的运算数的各二进位全部右移若干位，">>"右边的数指定移动的位数。	A >> 2 结果为 15 ，二进制为 0000

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

```
假定 A = 60， B = 13， 其二进制数转换为：
A = 0011 1100

B = 0000 1101

-----------------

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001
```

赋值运算符

运算符	描述	Example
=	简单的赋值运算符，将一个表达式的值赋给一个左值	C = A + B 将 A + B 表达式结果赋值给 C
+=	相加后再赋值	C += A 等于 C = C + A
-=	相减后再赋值	C -= A 等于 C = C - A
*=	相乘后再赋值	C = *A* 等于 *C* = *C* A
/=	相除后再赋值	C /= A 等于 C = C / A
%=	求余后再赋值	C %= A 等于 C = C % A
<<=	左移后赋值	C <<= 2 等于 C = C << 2
>>=	右移后赋值	C >>= 2 等于 C = C >> 2
&=	按位与后赋值	C &= 2 等于 C = C & 2
^=	按位异或后赋值	C ^= 2 等于 C = C ^ 2
|=	按位或后赋值	C |= 2 等于 C = C | 2

其他运算符

运算符	描述	Example
&	返回变量存储地址	&*a*; 将给出变量的实际地址。
*	指针变量。	*a; 是一个指针变量

运算符的优先级

优先级	运算符
5	`*` `/` `%` `<<` `>>` `&` `&^`
4	`+` `-` `|` `^`
3	`==` `!` `=` `<` `<=` `>` `>=`
2	`&&`
1	`||`

函数

函数的定义，使用`func`定义

具体如下

```
func 函数名(参数列表)(返回值列表){
    //Todo
    函数体
}
```

example

```
func add(x int, y int) int {
    return x + y
}
```

分支与循环

分支

Go的语句中有两个分支，分别是`if`和`switch`

if/if-else

example 1

```
x := 10
if x > 0 { // 如果x大于0
    fmt.Println("x is positive") // 打印"x is positive"
}
```

example 2

```
x := -10
if x > 0 { // 如果x大于0
    fmt.Println("x is positive") // 打印"x is positive"
} else { // 否则
    fmt.Println("x is negative") // 打印"x is negative"
}
```

switch-case

example

```
x := 10
switch x { // 根据x的值选择执行
case 1, 2, 3: // 如果x等于1或2或3
    fmt.Println("x is small") // 打印"x is small"
case 4, 5, 6: // 如果x等于4或5或6
    fmt.Println("x is medium") // 打印"x is medium"
case 7, 8, 9: // 如果x等于7或8或9
    fmt.Println("x is large") // 打印"x is large"
default: // 如果x不等于上述任何值
    fmt.Println("x is unknown") // 打印"x is unknown"
}
```

循环

Go的循环只有两个分别是`for`和`range`,没有`while`和`do`循环

for

虽然Go没有这些循环但是go可以通过for循环去实现

正常形式

example

```
for i := 0; i < 10; i++ { // 初始化i为0，每次循环后i加一，直到i等于10时停止
    fmt.Println(i) // 打印i的值
}
```

无限循环

example

```
for { // 永远不会停止的循环
    fmt.Println("Hello") // 打印"Hello"

```

类while循环

example

```
for i < 5 {
        fmt.Println(i)
        i++
    }
```

类似C语言的

```
while (i < 5) {
        printf("%d\n", i);
        i++;
    }
```

类do-while循环

example

```
    for {
        fmt.Println(i)
        i++
        if i >= 5 {
            break
        }
    }

```

类似C语言的

```
do {
        printf("%d\n", i);
        i++;
    } while (i < 5);
```

for-range循环

常用来遍历

example

```
for index, value := range array {
    fmt.Println(index, value)
}
```

容器

Go 语言的容器主要是指一些数据结构，它们可以存储和管理多个元素。Go 原生提供了一些常见的容器类型，包括数组（array）、切片（slice）、映射（map）、通道（channel）等。

数组

固定长度的容器，定义后长度不可改变

example

```
var arr [5]int  // 创建长度为5的整数数组
    arr[0] = 10     // 给第一个元素赋值
```

切片

可以理解为动态数组，而且切片是引用类型，数组是值类型，切片使用的场景相较于数组更多

长度可变，比较灵活

example

```
slice := []int{1, 2, 3, 4, 5}
```

映射（字典）

键值对的集合，键和值可以是不同类型

为哈希表结构

example

```
m := map[string]int{
        "apple":  1,
        "banana": 2,
        "cherry": 3,
    }
```

管道

通道是 Go 语言中的一种数据结构，用于在不同 goroutine 之间传递数据。

example

```
ch := make(chan string)
```

参考资料

1. Go语言之旅
2. Go语言圣经（The Go Programming Language）
3. 【尚硅谷】Golang入门到实战教程丨一套精通GO语言

第一次课的作业

请大家尽自己最大的努力去完成，将作业项目打包压缩成压缩包，并作为附件邮箱发到`yangziqiang@redrock.team`中，邮箱的主题为`学号-姓名-后端第几次作业-lv几`（做到哪一级发几）

example：2024213xxx-卷娘-后端第一次作业-lv2

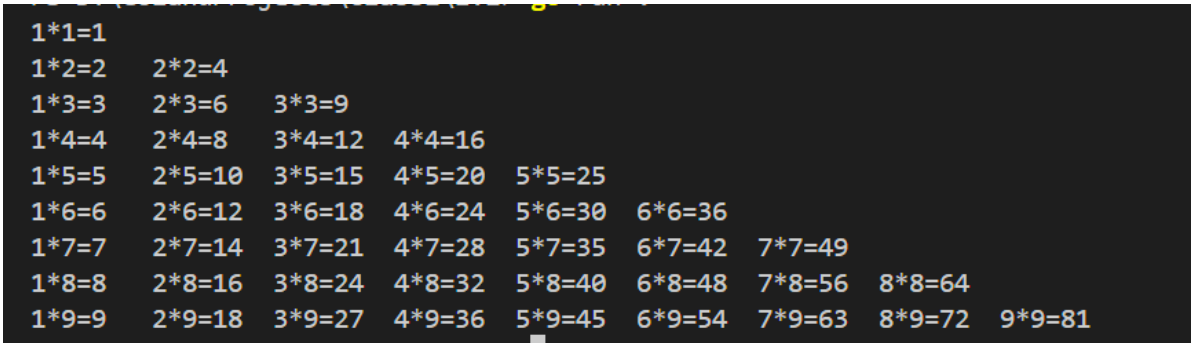
lv0：复习本节课内容并预习下一节课的内容

这个作业不难，但是希望可以大家可以认真去做一下，因为我们对语法知识不会讲的特别细致，需要同学们自己课下去学，最好是把每个代码都敲一遍。

lv1：使用代码实现九九乘法表

> **请不要使用面向结果编程，即使用`fmt.Println`把九九乘法表输出出来**
>
> 你可能会使用到for

效果图



lv2：实现一个简单四则运算的计算器

要求：1. 用户输入一个数，接着输入一个运算符，然后是最后一个数字，输出结果

2.询问用户是否继续使用

3. `exit`退出程序

> 可能会使用到 for，if switch

效果图

```
欢迎使用Go语言计算器！
请输入两个整数和一个操作符，进行四则运算
输入exit退出程序
请输入第一个整数：1
请输入操作符：+
请输入第二个整数：1
1 + 1 = 2
是否继续？(exit退出)：y
请输入第一个整数：114515
请输入操作符：-
请输入第二个整数：1
114515 - 1 = 114514
是否继续？(exit退出)：exit
感谢使用！再见！
```

计算器ProMax版（学有余力的同学可以尝试一下）

可以输入一整个算式进行运算，并支持小数、四则混合运算、括号等功能

> 可能会使用的知识：切片、映射、栈、中缀与后缀表达式、字符串操作

效果图

```
欢迎使用Go语言计算器！
请输入一个合法的算术表达式，例如：(3.14+2.71)*2/5
输入exit退出程序
请输入：8.1+4.4*(7+1.1)-2.2/2
结果是：42.640000
请输入：(3.14+2.71)*2/5
结果是：2.340000
请输入：exit
感谢使用！再见！
```

lv3：罗马数字转整数

罗马数字包含以下七种字符: `I`， `V`， `X`， `L`，`C`，`D` 和 `M`。

字符	数值
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

例如， 罗马数字 `2` 写做 `II` ，即为两个并列的 1 。`12` 写做 `XII` ，即为 `X` + `II` 。 `27` 写做 `XXVII`, 即为 `XX` + `V` + `II` 。

通常情况下，罗马数字中小的数字在大的数字的右边。但也存在特例，例如 4 不写做 `IIII`，而是 `IV`。数字 1 在数字 5 的左边，所表示的数等于大数 5 减小数 1 得到的数值 4 。同样地，数字 9 表示为 `IX`。这个特殊的规则只适用于以下六种情况：

- `I` 可以放在 `V` (5) 和 `X` (10) 的左边，来表示 4 和 9。
- `X` 可以放在 `L` (50) 和 `C` (100) 的左边，来表示 40 和 90。
- `C` 可以放在 `D` (500) 和 `M` (1000) 的左边，来表示 400 和 900。

给定一个罗马数字，将其转换成整数。

> 可能会使用到map, []int, for， if-else
>
> 提示：1. 试着使用哈希表map，将字符的值存储下来
>
> 2.比这个罗马数字小的在它左边的要做什么，在右边又要做什么

效果图 （注释为该问题的答案）

```go
func main() {
    fmt.Println(romanToInt("III")) // 3
    fmt.Println(romanToInt("MCMXCIV")) //1994
    fmt.Println(romanToInt("LVIII")) //58
    fmt.Println(romanToInt("IX")) //9
}
```