## 基本语法：

1. 只有package为main的包能包含main函数。
2. import “fm” 导入其他包
3. 使用package+别名：当使用第三方包时，包名可能会非常接近或者相同，此时可以使用别名来区分和调用。
4. const PI = 3.14 常量的定义
5. var name = “dadad” 定义全局变量
6. 定义变量的格式：var 变量名字 类型;
7. Go语言不存在隐私类型转化
8. 化简后的方式： b:=5;
9. 全局变量的声明不能省略var关键字
10. 局部变量不能使用var（）方式简写
11. 同时声明几个相同类型的变量并初始化：a, b, c, d := 1, 2, 3, 4
12. 使用空白符号”\_”对赋值进行忽略：a, \_, c, d := 1, 2, 3, 4 // 会忽略2的赋值
13. type newType int // 类型声明
14. type iner interface {
15. } //接口声明 type+接口名字+interface
16. import(“dad”, “fmt”) //使用这种方式一次导入多个包
17. 如果导入的包没有使用，程序就会报错。
18. 可见性规则：Go语言中，使用大小写来决定常量、变量、类型、接口、结构或函数是否可以被外部所调用：
19. 函数名首字母小写即为private
20. 函数名首字母大写即为public
21. Go语言中private和public的区别在于private在包外面不可见，private在包里面也是可见的，public在任何地方可见。
22. 类型转换： A [:]= TypeofA(B);

(1)string(Int) 把整型转化为对应的ASCII文本。

1. bool不能和其他类型进行转化
2. 常量：const
3. 如果声明常量以后不赋值，并且没有填写类型的时候，并且每一行声明的常量的数量是一样的，系统会给常量赋值为上一行的值。
4. 常量表达式（常量的等号右边的表达式），不允许出现运行时才能计算得到的值，必须要是编译时就能确定的值。常量表达式中必须是常量。
5. 枚举：
6. Iota：从常量组第一个常量开始，值为0，然后每增加一个常量，值加1。
7. 类型判断：

（1）stat,ok := sys.(\*syscall.Stat\_t) //判断sys是不是\*syscall.Stat\_t类型的，是的话，返回true和这个类型的变量。否则只返回false；

## Go基本类型：

1. 复数：complex64/complex128，8/16字节
2. 足够保存指针的32位或者64位整数型：uintptr
3. 浮点型：float32/float64，4/8字节，精确到小数点后7/15位
4. 类型零值：变量被声明为某种类型以后的值。

## Go常用包及函数：

1. strconv:

(1)strconv.Itoa(i int); //把i转化为对应的字符串

(2) strconv.Atoi(b string); // 把转化为字符串转化为对应的整数，返回类型为int、error两个

1. sort：

（1）Ints(arr []int); // 对数组排序

## Go常用运算符：

1、& 两个位都为1，结果为1

2、| 两个位有一个为1，结果为1

3、^ 两个位相同为0，不同为1

4、&^ 两个位，上面为1，下面为0，结果才为1，其他为0

0110

1011

-------

0100

5、&& 左右两个条件，如果左边的条件不满足了，不会执行后面的条件

6、& 左右两个条件，如果左边的条件不满足了，会执行后面的条件

7、++、-- 不能作为表达式，只能作为单独的一条语句

## Go语句：

1. if：
2. 条件表达式没有括号
3. for：
4. for {

} // 死循环

1. switch：
2. 不需要写break语句，一旦满足条件，自动终止。
3. 如果使用希望继续执行下一个case，需要使用fallthrougn语句。
4. goto、break配合标签使用的异同：
5. 如果goto跳转到标签处，从标签处开始执行
6. break是跳出与标签相对应的循环，然后继续完后执行。

## 数组：

1. 定义：var arr[n] 类型，如var arr []int = []int{1, 2, 3, 4}
2. 指定数组中其中一个值：var arr = [3]int{2: 3} //把2下标对应的元素设置为3
3. 使用new申请数组空间：var p = new([5]\*int) //申请5个整型指针元素的数组，这里的p是指针数组的指针（数组指针）
4. 定义二维数组：

var arr = [2][3]int{

{1, 2, 3},

{4, 5, 6}} //注意这两个大括号要连在一起

1. 去索引a到b，arr[a:b+1]

## 函数function：

1. Go函数不支持嵌套、重载和默认函数
2. 闭包使用示例：

func **closure**(x int) func(int) int {

return func(y int) int {

return x + y

}

}

func main() {

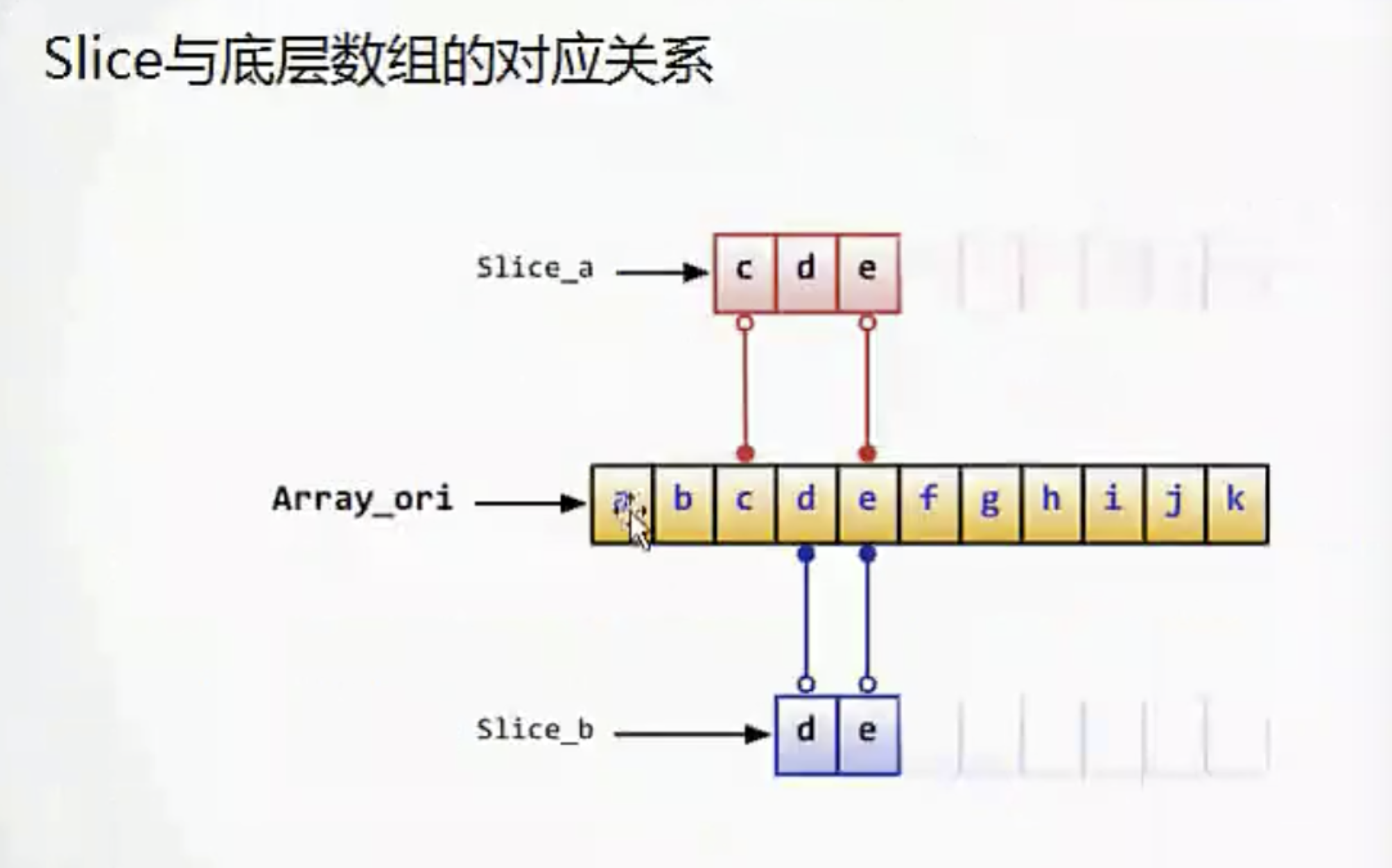
f := closure(10)

fmt.Println(f(1))

}

1. defer：
2. 当一个函数调用前有关键字 **defer** 时, 那么这个函数的执行会推迟到包含这个 defer 语句的函数即将返回前才执行.
3. 如果函数体内某个变量作为defer时匿名函数的参数，则在定义defer时即已经获得了变量拷贝，否则则是引用某个变量的地址。

## 切片slice：



1. var s1 []int // 中括号里面既没有数字也没有3个点，表示这个是slice数组.
2. Slice本质是指向底层数组的位置。
3. Reslice:
4. Reslice时索引以被slice的切片为准
5. 索引越界不会导致底层数组的重新分配而是引发错误。（Slice索引会导致底层数组重新分配）
6. 索引不可以超过声明slice时的cap（）值
7. Append：
8. 可以再slice尾部追加元素
9. 可以将一个slice追加在另一个slice尾部
10. 如果最终长度未超过追加到slice的容量则返回原始slice
11. 如果超过slice的容量则将重新分配数组并拷贝原始数据
12. Copy：第二个slice拷贝到第一个slice中，第一个slice容量不够的话，不会扩充容量

## Range函数：

range函数是个神奇而有趣的内置函数，你可以使用它来遍历数组，切片和字典。

当用于遍历数组和切片的时候，range函数返回索引和元素；

当用于遍历字典的时候，range函数返回字典的键和值

## struct结构体：

1. 定义匿名结构体：

p := &struct {

// 这个大括号内定义结构体

name string

age int

}{

// 这个大括号内赋值

name: "abc",

age: 12,

}

1. 定义普通结构体：

type person struct {

name string

age int

}

func main() {

p := &person{

// 这个大括号内赋值

name: "abc",

age: 12,

}

fmt.Println(p)

}

3、

## 方法method:

1. Go 中虽没有class，但依旧有method
2. 通过显示说明receiver来实现与某个类型的组合
3. 只能为同一个包中的类型定义方法
4. Receiver 可以是类型的值或者指针
5. 不存在方法重载
6. 可以使用值或指针来调用方法，编译器会自动完成转换
7. 从某种意义上来说，方法是函数的语法糖，因为receiver其实就是

方法所接收的第1个参数（Method Value vs. Method Expression）

1. 如果外部结构和嵌入结构存在同名方法，则优先调用外部结构的方法
2. 类型别名不会拥有底层类型所附带的方法
3. 方法可以调用结构中的非公开字段
4. 格式：

func (t \*TF) incre() {

(\*t) += 100

}

// 通过(t \*TF)和结构TF或者基本类型TF绑定在一起

## 接口interface：

1、package main

import (

"fmt"

)

type USB interface {

name() string

connect()

}

type sUSB struct {

n string

}

func disconnect(usb USB) {

// if \_, ok := usb.(sUSB); ok {

// fmt.Println("disconnect")

// }

switch v := usb.(type) {

case USB:

fmt.Println(v.name() + " " + "disconnect")

default:

}

}

func (s sUSB) name() string {

return s.n

}

func (s sUSB) connect() {

fmt.Println("connect")

}

func main() {

var s USB = sUSB{

n: "sUSB"}

s.connect()

fmt.Println(s.name())

disconnect(s)

}

2、接口是一个或多个方法签名的集合

3、只要某个类型拥有该接口的所有方法签名，即算实现该接口，无需显示

声明实现了哪个接口，这称为 Structural Typing

1. 接口只有方法声明，没有实现，没有数据字段
2. 接口可以匿名嵌入其它接口，或嵌入到结构中
3. 将对象赋值给接口时，会发生拷贝，而接口内部存储的是指向这个

复制品的指针，既无法修改复制品的状态，也无法获取指针

1. 只有当接口存储的类型和对象都为nil时，接口变量才等于nil
2. 接口调用不会做receiver的自动转换
3. 接口同样支持匿名字段方法
4. 接口也可实现类似OOP中的多态
5. 空接口可以作为任何类型数据的容器
6. 类型判断：

switch v := usb.(type) {

case USB:

fmt.Println(v.name() + " " + "disconnect")

default:

}

## 反射：

1. 如果结构体的字段是私有的，则反射机制无法无法反射出其字段，会报错。cannot return value obtained from unexported field or method
2. 获取类型，查看是否是结构体：

if k := t.Kind(); k != reflect.Struct {

return

}

1. t := reflect.TypeOf(s) // 相当于获取java的Class对象，java的Class对象代表.class文件
2. f := t.Field(i) // 获取第一个字段
3. v := reflect.ValueOf(s); val := v.Field(i).Interface() // 获取第i个字段的值
4. 取出结构中的匿名字段：通过序号

(1)t.FieldByIndex([]int{0: 1}) // 通过传入一个slice，0是相对最外层的结构的字段顺序，1是相对内层的字段序号

(2) v.FieldByName("Name") // 通过名称，推荐这种方法

7、 通过指针改变变量的值。

// x := 123

v := reflect.ValueOf(&x)

fmt.Printf("%v -- %p", v, &x)

v.Elem().SetInt(999)

8、调用方法: u := User{1, "OK", 12}

v := reflect.ValueOf(u)

mv := v.MethodByName("Hello")

args := []reflect.Value{reflect.ValueOf("jk")}

mv.Call(args)

## 并发：

1. goroutine：
2. goroutine :协程（轻量级线程，创建、销毁的代价很小）,协程在逻辑上上顺序执行
3. 每个实例 4-5KB 的栈内存占用和由于实现机制而大幅减少的创建和销毁开销，是制造 Go 号称的高并发的根本原因。另外，goroutine 的简单易用，也在语言层面上给予了开发者巨大的便利
4. 并发主要由切换时间片来实现“同时”运行，在并行则是直接利用

多核实现多线程的运行，但 Go 可以设置使用核数，以发挥多核计算机

的能力

1. Chanel：
2. Chanel是goroutine沟通的桥梁，大都是阻塞同步的。
3. 通过make创建，close关闭
4. Chanel是引用类型
5. 可以使用单向或双向通道
6. 可以设置缓存大小，有缓存是异步的，不会发生阻塞；无缓存是同步的，会发生阻塞。
7. 使用实例1：

c := make(chan bool)

go func() {

fmt.Println("GO")

c <- true //匿名函数结束后，写入一个值

}()

<-c //main函数在这里阻塞，直到可以从chanel中读出值

1. Select：
2. 可处理一个或多个 channel 的发送与接收
3. 同时有多个可用的 channel时按随机顺序处理
4. 可用空的 select 来阻塞 main 函数
5. 可设置超时

## 注意事项：

1. 在Golang中，一个中文的Unicode编码占3字节，字符占1字节。
2. 不用像c++，Object-C一样，Golang只要直接编译main文件，就可以生成可执行文件了。