go指针 1、类型指针，允许对这个指针类型的数据进行修改，传递的数据可以直接使用指针，无须拷贝数据，类型指针不能偏移和运算2、切片，由指向起始元素的原始指针、元素数量和容量组成

ptr := &house //对字符串取地址，将指针保存到变量ptr中

fmt.print(“%T”,ptr) //res: \*string

value:=\*ptr //对ptr指针进行取值，变量value的类型变成string

指针交换

func swap(a,b \*int){

t:=\*a

\*a=\*b //取b指针的值，赋给a指针指向的变量

\*b=t

}

go 切片slice

由数组、len、cap，在append扩容时，会查看数组后面有没有连续内存块，没有就重新生成大的

go数组

数组是值类型，赋值和传参都会复制整个数组数据（地址变化）。数组长度不可变。

map不是线程安全的

go map通过hash表实现

**var map\_var map[key\_data\_type]value\_data\_type**

**map\_var := make(map[key\_data\_type]value\_data\_type)**

value, ok := map\_var[key]

**Go部分面试题**

1. **等待一百个协程执行完。var w syn.WaitGroup。w.add(1) go func{w.done}。w.wait()**
2. **可以限制线程数量(初始一万SetMaxThreads)。**
3. **可以从已经关闭的chan读数据，v,ok= <-ch读完返回默认值和false**
4. **goroutine数量受限制于内存大小。**
5. **goroutine协程池。（通过启动多少个goroutine的数量的channel）**
6. **单个协程的栈内存2KB**
7. **go defer是栈结构**
8. **类型的实例不可比较，指针类型。**
9. channel实现协程的同步，

for range ch{

count--

if 0==count{

close(ch)

}

}

1. map无顺序，通过slice让key有序。

sort.Strings(slice) //进入地址进行改变

11. go默认switch的每个case后面带break，匹配成功后跳出整个switch，使用fallthrough强制执行后面的代码。但是不能用在最后一个分支，比如default。

12. go并发机制以及它所使用的CSP并发模型

**CSP以通信的方法来共享内存。用于描述两个独立的并发实体通过共享的通讯channel（管道）进行通信的并发工具。不关注发送信息的实体，而关注与发送信息时使用的channel。**

**go中的channel是被单独创建并且可以在进程之间传递，它的通信模式类似于boss-worker，一个实体将消息发送到channel中，然后又监听这个channel的实体处理，两个实体之间是匿名的，这个就实现实体中间的解耦，其中channel是同步的：一个消息被发送到channel中，最终是一定要被另外的实体消费掉的，在实现原理上类似一个阻塞的消息队列。**

**goroutine是go实际并发执行的实体，它底层是使用协程(coroutine)实现并发的，coroutine是一种处在用户态的用户线程。使用原因：1、用户空间 避免了内核态和用户态的切换导致的成本2、可以由语言和框架层进行调度3、更小的栈空间允许创建大量的实例**

**go中的goroutine的特性：1、golang内部有三个对象GMP，P上下文or处理器M工作线程G对象（goroutine）--一个P对应一个工作线程对象，线程去检查并执行goroutine对象。碰到goroutine对象阻塞的时候，会启动一个新的工作线程，以充分利用cpu资源。**

**G(Goroutine)：协程，用户级的轻量级线程，每个goroutine对象中的sched保存着其上下文信息。**

**M(Machine):对内核级线程的封装，数量对应真实的CPU数**

**P(Processor):G和M的调度对象，用来调度G和M之间的关联关系，其数量可以通过GOMAXPROCS()来设置，默认为核心数。**