今日总结 2019-03-27.28

每日一学

问题一：

包导入的几种形式：

```
import   declaration    Local name of Sin
import   "math"         math.Sin
import m "math"         m.Sin
import . "math"         Sin
```

除了以上三种形式，还有一种特殊形式，即：`import _ "math"` 大家知道这种特殊形式有什么用途吗，代码中见到过没？

讨论结果：

1. 将该包引入，只初始化里面的init函数和一些变量，不能通过包名来调用包里的函数；
2. init有个特殊的地方，go里面，同一个包同一个文件可以有多个init函数，多个init不会报错，都可以执行的；
3. 导包不调用会报错，但又要加载init函数，就这样做了。

问题二：

关于接口的 nil 问题。 在底层，接口作为两个元素实现：一个类型和一个值。该值被称为接口的动态值， 它是一个任意的具体值，而该接口的类型则为该值的类型。对于 int 值3， 一个接口值示意性地包含(int, 3)。 只有在内部值和类型都未设置时(nil, nil)，一个接口的值才为 nil。特别是，一个 nil 接口将总是拥有一个 nil 类型。若我们在一个接口值中存储一个 int 类型的指针，则内部类型将为 int，无论该指针的值是什么：(*int, nil)。 因此，这样的接口值会是非 nil 的，即使在该指针的内部为 nil。 这种情况会让人迷惑，而且当 nil 值存储在接口值内部时这种情况总是发生， 例如错误返回：

```
func returnsError() error {
    var p MyError = nil
    if bad() {
        p = ErrBad
    }
    return p // 将总是返回一个非nil错误。
}
```

如果一切顺利，该函数会返回一个 nil 的 p， 因此该返回值为拥有(MyError, nil)的 error 接口值。这也就意味着如果调用者将返回的错误与 nil 相比较， 它将总是看上去有错误，即便没有什么坏事发生。要向调用者返回一个适当的 nil error，该函数必须返回一个显式的 nil：

```go
func returnsError() error {
    if bad() {
        return ErrBad
    }
    return nil
}
```

这对于总是在签名中使用 error 类型返回错误（正如我们上面做的）而非像 *MyError 这样具体类型的函数来说是个不错的主意，它可以帮助确保错误被正确地创建。 例如，即使 os.Open 返回一个 error， 若非 nil 的话，它总是具体的类型 *os.PathError。 对于那些描述，无论接口是否被使用，相似的情形都会出现。只要记住，如果任何具体的值已被存储在接口中， 该接口就不为 nil。 接口的内部定义

```go
type iface struct {
    tab  *itab // 代表类型
    data unsafe.Pointer  // 代表数据
}
```

讨论结果：

1. 昨天写了篇文章，看是否能帮助大家理解接口的 nil 值 Go 语言接口详解（一）

问题三：

为什么 T 和 `*T` 有不同的方法集?

- 如果一个接口值包含一个指针 `*T`，一个方法调用可通过解引用该指针来获得一个值，所以，*T 方法集包含 T 的方法集；
- 但反过来，如果一个接口值包含一个值 T，就没有安全的方式让一个方法调用获得一个指针。一方面，有可能 T 不可寻址；另一方面，即使 T 可寻址，但方法可能错误的通过指针修改它的值，而实际上这个修改会丢失，得到不是期望的结果。因此Go语言规范才定义 T 的方法集不包括 *T 的方法集。

知识点学习

问题一：

关于类型断言 Go 语言规范规定，类型断言是指：

- 对于接口类型的表达式 x 与类型 T，主表达式：x.(T)，断言 x 不为 nil 且存储于 x 中的值其类型为 T。 记法 x.(T) 称为 类型断言。
- 这里明确指出，进行类型断言时，x 必须是接口（注意，只要是接口就可以，不在乎是不是空接口）。注意，T 可以是类型或接口。
- 注意，如果类型断言成立，则该表达式的值即为存储于 x 中的值，且其类型为 T；若该类型断言不成立， 就会 panic。

```
var x interface{} = 7  // x 拥有动态类型 int 与值 7
i := x.(int)           // i 拥有类型 int 与值 7
type I interface { m() }
var y I
s := y.(string)        // 非法：string 没有实现 I（缺少方法 m）
r := y.(io.Reader)     // r 拥有 类型 io.Reader 且 y 必须同时实现了 I 和 io.Reader，否则 panic
```

如果不确定接口的动态类型是什么，为了避免 panic，可以接收表达式的第2个参数，即：

```
v, ok = x.(T)
v, ok := x.(T)
var v, ok = x.(T)
```

通过判断 ok 是 true 还是 false，如果为 true，表示 x 的动态类型是 T；否则 x 的动态类型不是 T。

讨论结果：

1. x.(T) 语法中，如果 T 是接口，编译器会自动检测 x 的动态类型是否实现了接口 T。

```
type Shape interface {
    Area() float32
}

type Perimeter interface {
    P() float32
}

type Circle struct {
    radius float32
}

func (c Circle) Area() float32 {
    return math.Pi * (c.radius * c.radius)
}

func main() {
    var s Shape = Circle{3}
    v1,ok1 := s.(Shape)
    v2,ok2 := s.(Perimeter)
    fmt.Println(v1,ok1)
    fmt.Println(v2,ok2)
}
```

问题二：

关于类型选择 (type switch) 类型选择的语法和类型断言的语法类似，但有如下要求：

1. 只能用于 switch 语句；
2. x.(type) 中的 type 是固定的，只能是 type 这个关键词；

3. 不允许使用 fallthrough 语句； 例如：

```
switch x.(type) {
// case
}
```

和类型断言一样，x 必须是接口。 每一个 case 中的类型必须实现了 x 接口。 另外一种语法：

```
switch i := x.(type) {
// case
}
```

当匹配到具体某个 case 时，i 即为 x 中该类型的值。

问题三：

关于 T 和 *T 方法集的，qq 群有人有另外的疑惑。这个可能是比较普遍的疑惑，我在此总结一下。

- 之前有一个主题讲解了为什么 T 的方法集不包含 *T 的方法集，但要注意，方法集的概念是用来判断类型是否实现了某个接口，不能因为 T 的方法集没有包含 *T 的方法集，就以为 T 就不能调用 *T 的方法，Go 语言规范明确说了，直接调用指针的方法，编译器会自动取 T 的指针，然后调用，这跟方法集没有关系。看如下例子，注意注释部分。希望大家明白这两者的区别和使用场景。

```
package main
import (
    "fmt"
)
type Speaker interface {
    Speak(language string)
}
type Chinese struct {
    Name string
}
func (c Chinese) Speak(language string) {
    fmt.Println("My name is", c.Name, ", I am Chinese, I speak", language)
}
type American struct {
    Name string
}
func (a *American) Speak(language string) {
    fmt.Println("My name is", a.Name, ", I am American, I speak", language)
}
func main() {
    var speaker Speaker = Chinese{Name: "zhangsan"}
    speaker.Speak("Chinese")
    // &Chinese{} 返回的指针，虽然 Chinese 的 Speek() 方法接收者是值类型，它会包含在指针类型 *Chinese 中，因此 *Chinese 实现了 Speaker 接口
    speaker = &Chinese{Name: "lisi"}
    speaker.Speak("English")
```

```
    // 编译不通过， American{}返回的是值，方法集中没有 Speak() 方法，因此没有实现 Speaker 接口
    // speaker = American{Name: "John"}
    // speaker.Speak("Chinese")
    // 但是 American 的值类型却可以直接调用指针接收者的方法，如：
    american := American{Name: "Tom"}
    american.Speak("English")
    speaker = &American{Name: "Mary"}
    speaker.Speak("English")
}
```

直接运行： The Go Playground

面试题

问题一：

以下代码是否有问题？如果没有输出什么？

```
func main() {
    i := GetValue()
    switch i.(type) {
        case int:
            println("int")
        case string:
            println("string")
        case interface{}:
            println("interface")
        default:
            println("unknown")
    }
}
func GetValue() int {
    return 1
}
```

讨论结果：

1. i.(type) type是保留字。不是实际类型，所以这是类型选择，i.(T) T为实际类型，这才是类型断言，当然类型选择和类型断言一样， i必须是接口类型；
2. 执行了一下，提示报错：cannot type switch on non-interface value i (type int)；
3. 期待球主后期总结类型断言……

问题二：

下面函数有问题吗?

```
func funcMui(x, y int)(sum int, error){
    return x+y, nil
}
```

讨论结果:

1. 多值返回值声明要么都直接声明类型，要么都命名返回值参数，不能混合使用;

同学问的问题

问题一:

脚本中的编译命令:

```
CGO_ENABLED=0 GOOS=linux go build -ldflags "-s -w" -o xxx -a main.go
```

编译期间，compile进程cpu占用率大于200%； 比普通的go build 生成的二进制程序小（41M->31M）; 中间的 `CGO_ENABLED=0` ； `-ldflags "-s -w"` ； 两部分该怎么理解呢?

解决思路:

1. `CGO_ENABLED=0` 表示禁用 CGO;
2. -s 去掉符号表（然后panic时候的stack trace就没有任何文件名/行号信息了）;
3. -w去掉DWARF调试信息，得到的程序就不能用gdb调试了;
4. 所以文件变小了。

问题二:

请教一下，go里面有没有批量检查空指针的方法或者工具呢， 就是在很多文件里去检查有没有引用空指针的情况。

球主回答:

1. 我明白你的意思了。就是想检测出项目中所有的空指针异常情况。
2. 这个得语言层面支持了。比如，Java 没法静态分析时候检查出空指针，但 Kotlin 却可以做到不会有空指针一样。目前，Go 有很多工具，比如这里列出的：GitHub - 360EntSecGroup-Skylar/goreporter: A Golan...
3. 但要工具检查空指针异常，没发现有这样的工具。原理上，需要跟踪指针的赋值情况，理论上不是没有可能，做词法分析，跟踪它们赋值情况。

问题三:

老师，我问个比较实际的问题，我的计算机基础比较差，现在用PHP开发一年多了，然后我看昨天看go语言的招聘要求大多对网络编程，多线程要比较熟悉，所以在学好go语言基础的情况下怎么学习这些东西或者说学这些东西到达什么程度就可以去投go的岗位做go开发呢

球主回答：

- PHP 希望转 Go，或者说学 Go 的人挺多的。Go 定位为系统编程语言（用于 Web 也很常见），所以网络编程确实挺重要的。这里面涉及比较多相关的知识。我专门较系统的总结下，到时发到咱们星球里。

问题四：

我想问大家一个问题，我现在的golang版本是1.9.2，可我想升级到最新的1.12.1去学习一下新特性，然后我遇到一些问题。

1. 我看到官网有提示说，可以在现有版本做升级，「go get golang.org/dl/go1.12.1」可我代理了之后还是没有发现有下载这个最新的包。
2. 然后我又看到一个是卸载了现在的版本，下载最新的安装。（这个因为工作主语言golang，没有试过，我怕这个万一整不好会耽误一些时间）。

   有经验的大大能分享一下吗？谢谢你们~

球主回答：

- go 本身的升级，不能 go get 来搞。go get 下载 go 包用的。
- 自从 go1.5 起，实现了自举。如果你通过源码安装，别卸载原来的。其实安装过程很简单。建议你源码安装，不会的可以星球里咨询。

资源分享

- 官方关于 module 的博文，推荐大家认真读读，并实际试验。 Go Modules 的使用方法 —— Go 官方博文 - Go语言中文网 - Golang中文社...