Go Language Notes

部分非常有用的知识请参考Useful\_Code文件夹中的代码

1. Programming language category
   1. 偏性能敏感的编译型 C C++ Java C# Delphi Objective-C etc.
   2. 偏快速业务开发的动态解析型 PHP Python Perl Ruby JavaScript Lua etc.
   3. 面向特定领域型 Erlang R MATLAB
2. Go language characteristic
   1. 自动垃圾回收
   2. 更丰富的内置类型
   3. 函数多返回值
   4. 错误处理
   5. 匿名函数类型和闭包
   6. 类型和接口
   7. 并发编程
   8. 反射
   9. 语言交互性
3. Go环境变量与工作目录，根据约定，GOPATH下需要建立3个目录
   1. bin(存放编译后生成的可执行文件)
   2. pkg（存放编译后的包文件）
   3. src（存放项目源码）
4. Go 常用命令
   1. go run 直接运行程序
   2. go build 测试编译，检查编译错误，并生成可执行文件
   3. go fmt 格式化源码
   4. go install 编译包文件并编译整个程序
   5. go test 运行测试文件以xxx\_text.go结尾的文件，会返回测试结果
   6. go doc 查看文档
5. Go的内置关键字（全部小写 25个）
6. Go注释 ：// and /\*\*/
7. Go程序的一般结构：
   1. Go程序是通过package进行组织的
   2. 只有package名称为main的包可以包含main包
   3. 一个可执行程序有且只有一个main包
8. Go中可以通过()批量导入包或者定义变量 此种方法称为“组”
9. Type newType int 重定义数据类型
10. Go中的可见性规则：使用首字母大小写来决定常量、变量、类型、接口、结构或者函数是否可以被外部调用，小写private大写public
11. Go语言程序基本结构 参考go\_basic\_structure.go文件
12. Go的基本类型
    1. 布尔型
       1. 长度：1字节
       2. 取值范围：true or false
       3. 不可以使用数字代替true and false
    2. 整型
       1. Int（有符号的整型）/uint（无符号整型）,此时要根据平台判断具体是多少位32或者64
       2. Int8/uint8指定位数
    3. 字节型byte(uint8的别名)
    4. 浮点型：float32/float64,很明显此时不需要double了。。
    5. 复数类型（比较生疏）：complex64/complex128
    6. 足够保存指针的32位或者64位整型：uintptr
    7. 其他值类型
       1. array
       2. struct
       3. string
    8. 引用类型
       1. Slice(薄片，原义)
       2. map
       3. chan(高并发)
    9. 接口类型
    10. 函数类型
13. 类型的零值和空值，这个和其他语言中类似，零值就是变量声明时候的初始值（此时系统会根据具体的数据类型赋予默认的数值），而空值就是值为空，没有默认值
14. 变量的声明（有多种方法，视情况使用，比较傲常用最后一种变量的声明方法）
    1. var a int
    2. var a int = 1
    3. var a = 1 此时系统根据数值自动给类型
    4. a := 1
15. 多个变量的声明与赋值
    1. 全局变量的声明可以使用var()的方式进行简写
    2. 全局变量的声明不可以省略var，但是可以使用并列方式
    3. 所有变量都可以使用类型推断（就是系统自动推断变量的具体类型）
    4. 局部变量不可以使用var()的方式简写，只能使用并行方式
       1. var a, b , c, d = 1, 2, 3, 4
       2. var(

a = 1

b = 2

c = 3

d = 4

)此种形式局部变量不可以使用

1. 类型转换
   1. Go中不存在隐式的类型转换，所有变量转换都必须要显式声明
   2. 转换只能发生在两种相互兼容的数据类型之间
   3. 类型转换的格式
      1. <ValueA> [:] = <TypeOfValueA>(<ValueB>)
2. 常量的定义
   1. 常量 const a int = 12,支持组定义
3. 流程控制

程序设计语言的流程控制语句，用于设定计算执行的次序，建立程序的逻辑结构。可以说，

流程控制语句是整个程序的骨架。

从根本上讲，流程控制只是为了控制程序语句的执行顺序，一般需要与各种条件配合，因此，

在各种流程中，会加入条件判断语句。流程控制语句一般起以下3个作用：

选择，即根据条件跳转到不同的执行序列；

循环，即根据条件反复执行某个序列，当然每一次循环执行的输入输出可能会发生变化；

跳转，即根据条件返回到某执行序列。

Go语言支持如下的几种流程控制语句：

条件语句，对应的关键字为if、else和else if；

选择语句，对应的关键字为switch、case和select（将在介绍channel的时候细说）；

循环语句，对应的关键字为for和range；

跳转语句，对应的关键字为goto。

在具体的应用场景中，为了满足更丰富的控制需求，Go语言还添加了如下关键字：break、continue和fallthrough。在实际的使用中，需要根据具体的逻辑目标、程序执行的时间和空

间限制、代码的可读性、编译器的代码优化设定等多种因素，灵活组合。

1. 在 if 的便捷语句定义的变量同样可以在任何对应的 else 块中使用。
2. defer 语句会延迟函数的执行直到上层函数返回(执行完), 延迟调用的参数会立刻生成，但是在上层函数返回前函数都不会被调用，延迟的函数调用被压入一个栈中，当函数返回时， 会按照后进先出的顺序调用被延迟的函数调用。

func main() {

defer fmt.Println("World")

fmt.Println("Hello")

}

Result: Hello World

1. Go指针
   1. 指针保存了变量的内存地址
   2. 类型 \*T 是指向类型 T 的值的指针，其零值是 nil
2. Map中通过双赋值检测某个键是否存在：
   1. elem, ok := m[key],如果key在m中则ok==true，elem指向对应的值，否则ok==false，elem为零值

Go 没有类。然而，仍然可以在结构体类型上定义方法，*方法接收者* 出现在 func 关键字和方法名之间的参数中。你可以对包中的 *任意* 类型定义任意方法，而不仅仅是针对结构体，但是，不能对来自其他包的类型或基础类型定义方法。

type myType struct {

x, y float64

}

func main() {

v := &myType{3, 4}

fmt.Println(v.abs())

}

// 定义了结构体myType的方法abs()

func (v \*myType) abs() float64 {

return math.Sqrt(v.x\*v.x + v.y\*v.y)

}

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

type MyFloat float64

func (f MyFloat) Abs() float64 {

if f < 0 {

return float64(-f)

}

return float64(f)

}

func main() {

f := MyFloat(-math.Sqrt2)

fmt.Println(f.Abs())

}

1. 接口：接口类型是由一组方法定义的集合，接口类型的值可以存放实现这些方法的任何类型的值。也就是说Go语言中接口的实现不需要implements关键字，而是只要结构体中有（而且必须要包含接口中所有的方法，此点和其他语言相似）接口所包含的函数就行，此时就是说结构体实现了接口（实际上是实现了接口中的方法）

package main

import (

"fmt"

"math"

)

type MyFloat float64

type myType struct {

x, y float64

}

type abser interface {

abs() float64

abc() int

}

func main() {

var a abser

f := MyFloat(-math.Sqrt2)

v := myType{3, 4}

a = f

fmt.Println(a.abs())

a = &v

fmt.Println(a.abs())

}

func (f MyFloat) abs() (result float64) {

if f < 0 {

result = float64(-f)

} else {

result = float64(f)

}

return

}

func (v \*myType) abs() (result float64) {

result = float64(math.Sqrt(v.x\*v.x + v.y\*v.y))

return

}

1. Web服务器
   1. 包http通过任何实现了http.handler的值来响应HTTP请求
   2. http.handler具体代码

package http

type Handler interface {

ServeHTTP(w ResponseWriter, r \*Requeset)  
}

// 包http通过任何实现了http.handler的值来响应HTTP请求

package main

// 导入一些需要的包

import (

"fmt"

"log"

"net/http"

)

//

type user\_Info struct {

name string

age int

}

func (user user\_Info) ServeHTTP(w http.ResponseWriter, r \*http.Request) {

// 自定义一些值，实际要处理http.Request中传来的值

fmt.Fprint(w, user)

}

func main() {

user := user\_Info{"Eadmund Lee", 23}

// 启动服务器端的监听，说明地址和端口以及处理情况（用实现http.Handler接口的结构体来进行处理），此时访问跟目录就可以得到返回跌结果

// err := http.ListenAndServe("localhost:4000", h)

http.Handle("/user\_Info", user)

err := http.ListenAndServe("localhost:4000", nil)

if err != nil {

log.Fatal(err)

}

}

1. 图片相关知识没有掌握（空闲时间或者用到的时候解决他）
2. Goroutine：Go运行时环境管理的轻量级线程
   1. 用法：go f(x, y, z) , 虽然方便，但是有很大的缺陷（以后再说）
   2. 主线程（main）结束（退出）之后不能再运行goroutine的线程了（此种情况和Java的线程不同），此时如果要跑goroutine线程就要采用阻塞主线程的方法，例如让主线程sleep或者当goroutine跑完的时候告诉main线程：“我要结束了，cpu你拿去吧”等
3. Channel（“管道”或者“信道”）：有类型的管道（是一种队列），可以用channel操作符<-对其进行发送或者接受操作
   1. Channel的创建 ch := make(chan int)//声明是int类型的管道
   2. Channel接收值 ch <- v 普通变量接收channel值 v := <-ch(含赋值过程)
   3. <-ch取ch的值fmt.Println(<-ch)就输出了ch的值
   4. 包含channel类型的函数必须要通过goroutine线程启动（具体原因还未清楚）
   5. 默认情况下，在channel类型（特别注意：此种channel类型没有带缓冲，即不是make(chan int, 2)）(2是长度，整型，参考API)另一端准备好之前，发送和接收都会阻塞（会出现fatal error: all goroutines are asleep - deadlock!）。这使得 goroutine 可以在没有明确的锁或竞态变量的情况下进行同步。（也就是说此时在包含channel类型的变量或者函数之间加上go就可以运行该“管道”或者“信道”）（为什么这样设计，暂时还不清楚，有待查找相关文档）

package main

import (

"fmt"

)

func sum(a []int, c chan int) {

sum := 0

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

sum += a[i]

}

c <- sum

}

func main() {

a := []int{1, 2, 3, 4, 5, 6, 7, 8}

// chan类型是一种队列（未验证）

c := make(chan int)

go sum(a[:len(a)/2], c)

go sum(a[len(a)/2:], c)

// fmt.Println(c)

x, y := <-c, <-c

fmt.Println(x, y, x+y)

}

1. 函数（func）
   1. 不支持
      1. 嵌套
      2. 重载
      3. 默认参数
   2. 支持
      1. 无需声明原型
      2. 不定长度变参
      3. 多返回值
      4. 命名返回值参数
      5. 匿名函数
      6. 闭包
   3. 注意：
      1. 定义函数使用关键字func，且左大括号不能另起一行
      2. 函数也可以作为一种类型使用
2. 函数中有时候输出的并不是自己需要的此时考虑是值传递还是引用传递往往能够解决问题
3. 结构体struct（本质就是字段的集合）
   1. Golang中通过struct提供面向对象
   2. 结构体字段可以通过结构体指针来访问，也可以通过结构体普通变量来访问（指针访问传递的是引用，而普通变量是值传递，需要copy，不仅效率低，而且修改数据不透明）
4. Select关键字还没有搞定
5. 下面是复习一遍Go指南之后的Notes
6. 函数的返回值一般来说如果程序较短可以不命名，但是一般情况下（项目开发）程序较长，逻辑较为复杂，所以建议返回值要有参数，而且return 语句后面要加上参数的名称（虽然不加上也行）
7. 数据类型转换，Golang中的数据类型转换非常简单（形式单一），只要在要转换类型前面加上转换后数据的类型，如var a int = 10 var c float64 = (float64)a 即，是不是非常方便简单呢?!哈哈,再也不用为数据类型转换发愁了。
8. Stringer：fmt包中定义的，type Stringer interface{String() string}
   1. 用来包装输出数据的格式的
   2. func (p Person) String() string {

return fmt.Sprintf("%v (%v years)", p.Name, p.Age)

}