六星教育.01.Go快速入门.md

0. 课程内容

1. Go语言介绍

1.1. 什么是Go语言

Go（又称 Golang）是 Google 的 Robert Griesemer，Rob Pike 及 Ken Thompson 开发的一种、静态强类型、编译型、并发型。

Go 语言语法与 C 相近，但功能上有：内存安全，GC（垃圾回收），结构形态及 CSP-style 并发计算。

于2007年9月开始设计Go，稍后Ian Lance Taylor、Russ Cox加入项目。Go是基于Inferno操作系统所开发的。Go于2009年11月正式宣布推出，成为开放源代码项目

1.2. go与golang

事实上Go语言的称呼就是Go，golang只是Go语言官网的域名。

官网地址：http://golang.org

1.3. go与golang

> 自带gc。
>
> 静态编译，编译好后，扔服务器直接运行。
>
> 简单的思想，没有继承，多态，类等。

丰富的库和详细的开发文档。

语法层支持并发，和拥有同步并发的channel类型，使并发开发变得非常方便。

简洁的语法，提高开发效率，同时提高代码的阅读性和可维护性。

超级简单的交叉编译，仅需更改环境变量

2. 安装go语言环境

在官方下载目录下载golang软件包：https://golang.google.cn/dl/

> linux安装

1. 下载二进制包：go1.4.linux-amd64.targz。s
2. 将下载的二进制包解压至 /usr/local目录。

```
tar -C /usr/local -zxvf go1.15.2.linux-amd64.tar.gz
```

3. 将 /usr/local/go/bin 目录添加至PATH环境变量：

```
export PATH=$PATH:/usr/local/go/bin
```

> win

windows下只需要双击.msi文件期即可运行

> 测试

编辑一份hello.go文件

```
package main

import "fmt"

func main() {
        var name string = "go"
        fmt.Println("hello world ", name)
}
```

只需要在命令行中执行 `go run hello.go` 即可

3. go工作目录及关键字

所有的go源码都是以 ".go" 结尾。

> 工作目录

Go平时开发的工作目录结构


```
go
  - bin                     # 可执行文件
  - pkg                     # 存放编译后生成的包文件
  - src                     # 存放项目源码（在这个目录往往会基于项目名或域名区分）
    - github.com            # 代表从github上下载的
```

> 关键字

```
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
```

4. go语言组成

- 包声明

```
package main
```

第一行代码 package main 定义了包名。你必须在源文件中非注释的第一行指明这个文件属于哪个包，如：package main。package main表示一个可独立执行的程序，每个 Go 应用程序都包含一个名为 main 的包。

- 引入包

```
import "fmt"
```

告诉 Go 编译器这个程序需要使用 fmt 包（的函数，或其他元素），fmt 包实现了格式化 IO（输入/输出）的函数。

- 函数

```
func main(){
    fmt.Println("hello world! go and golang");
}
```

是程序开始执行的函数。main 函数是每一个可执行程序所必须包含的，一般来说都是在启动后第一个执行的函数（如果有 init() 函数则会先执行该函数）。

- 变量

```
var name string = "go"
```

go的变量声明会采用var关键词进行声明同时会携带上声明的数据类型

5. go基础数据类型

https://www.cnblogs.com/itbsl/p/9854681.html

类型	描述
uint	32位或64位
uint8	无符号 8 位整型 (0 到 255)
uint16	无符号 16 位整型 (0 到 65535)
uint32	无符号 32 位整型 (0 到 4294967295)
uint64	无符号 64 位整型 (0 到 18446744073709551615)
int	32位或64位
int8	有符号 8 位整型 (-128 到 127)
int16	有符号 16 位整型 (-32768 到 32767)
int32	有符号 32 位整型 (-2147483648 到 2147483647)
int64	有符号 64 位整型 (-9223372036854775808 到 9223372036854775807)
byte	uint8的别名(type byte = uint8)
rune	int32的别名(type rune = int32)，表示一个unicode码

uintptr	无符号整型，用于存放一个指针是一种无符号的整数类型，没有指定具体的bit大小但是足以容纳指针。uintptr类型只有在底层编程是才需要，特别是Go语言和C语言函数库或操作系统接口相交互的地方。
float32	IEEE-754 32位浮点型数
float64	IEEE-754 64位浮点型数
complex64	32 位实数和虚数
complex128	64 位实数和虚数

5.1. 常见类型操作

5.1.1. 整型

整型数据分为两类，有符号和无符号两种类型

有符号: int, int8, int16, int32, int64

无符号: uint, uint8, uint16, uint32, uint64, byte

不同位数的整型区别在于能保存整型数字范围的大小；有符号类型可以存储任何整数，无符号类型只能存储自然数int和uint的大小和系统有关，32位系统表示int32和uint32，如果是64位系统则表示int64和uint64

byte与uint8类似，一般用来存储单个字符

在保证程序正确运行下，尽量使用占用空间小的数据类型fmt.Printf("%T", var_name)输出变量类型unsafe.Sizeof(var_name)查看变量占用字节

5.1.2. 浮点型

浮点型也就是小数类型，可以存放小数。比如6.6，-12.34

1. 关于浮点数在机器中存放形式的简单说明，浮点数=符号位+指数位+尾数位
2. 尾数部分可能丢失，造成精度损失。-123.0000901

```
package main

import "fmt"

func main() {
    var num1 float32 = -123.0000901
    var num2 float64 = -123.0000901
    fmt.Println("num1 = ", num1, "num2 = ", num2);
}
```

效果

```
C:\Users\shineyork>go run shineyork/02
num1 =  -123.00009 num2 =  -123.0000901
```

说明：float64的精度要比float32的要准确

说明：如果我们要保存一个精度高的数，则应该选择float64

3. 浮点型的存储分为三部分：符号位+指数位+尾数位，在存储过程中，精度会有丢失
4. golang的浮点型默认为float64类型
5. 通常情况下，应该使用float64,因为它比float32更精确
6. 0.123可以简写成.123,也支持科学计数法表示:5.1234e2 等价于512.34

5.1.3. 字符与字符串

> 字符

Golang中没有专门的字符类型，如果要存储单个字符(字母)，一般使用byte来保存。

字符串就是一串固定长度的字符连接起来的字符序列。Go的字符串是由单个字节连接起来的，也就是说对于传统的字符串是由字符组成的，而**Go的字符串不同，它是由字节组成的**。

- 字符只能被单引号包裹，不能用双引号，双引号包裹的是字符串
- 当我们直接输出type值时，就是输出了对应字符的ASCII码值
- 当我们希望输出对应字符，需要使用格式化输出
- Go语言的字符使用UTF-8编码，英文字母占一个字符，汉字占三个字符
- 在Go中，字符的本质是一个整数，直接输出时，是该字符对应的UTF-8编码的码值。
- 可以直接给某个变量赋一个数字，然后按格式化输出时%c，会输出该数字对应的unicode字符
- 字符类型是可以运算的，相当于一个整数，因为它们都有对应的unicode码

```
//字符只能被单引号包裹，不能用双引号，双引号包裹的是字符串
var c1 byte = 'a'
var c2 byte = '0'

//当我们直接输出type值时，就是输出了对应字符的ASCII码值
//'a' ==> 97
fmt.Println(c1, "--", c2)
//如果我们希望输出对应字符，需要使用格式化输出
fmt.Printf("c2 = %c c2 = %c", c1, c2)
```

但是如果我们保存的字符大于255，比如存储汉字，这时byte类型就无法保存，此时可以使用uint或int类型保存

字符类型本质探讨

1. 字符型存储到计算机中，需要将字符对应的码值(整数)找出来

   ```
   存储：  字符-》码值-》二进制-》存储
   读取：  二进制-》码值-》字符-》读取
   ```

2. 字符和码值的对应关系是通过字符编码表决定的(是规定好的)

3. Go语言的编码都统一成了UTF-8。非常的方便，很统一，再也没有编码乱码的困扰了

> 字符串

字符串就是一串固定长度的字符连接起来的字符序列。Go的字符串是由单个字节连接起来的。Go语言的字符串的字节使用UTF-8编码标识Unicode文本

1. 字符串一旦赋值了，就不能修改了:**在Go中字符串是不可变的。**

2. 字符串的两种标识形式

   ```
   * 双引号，会识别转义字符
   ```



```go
var str = "abc\nabc" //输出时会换行
```

* 反引号，以字符串的原生形式输出，包括换行和特殊字符，可以实现防止攻击、输出源代码等效果

```go
var str string = `abc\nabc` //输出时原样输出，不会转义
```

3. 字符串拼接方式"+"

```
var str string = "hello " + "world"
str += "!"
```

4. 当一行字符串太长时，需要使用到多行字符串，可以使用如下处理

```
//正确写法
str := "hello" +
       " world!"
fmt.Println(str)

//错误写法
str := "hello "
       + "world!"
fmt.Println(str)
```

5.2. 类型强转

1. Go语言不存在隐式类型转换，所有的类型转换都必须显式的声明（隐式转化就是自动转化）
2. 格式 ：valueOfTypeB = typeB(valueOfTypeA)（类型 B 的值 = 类型 B(类型 A 的值)）

```
var a int = 100
var b int8 = 100
// 正确写法
// 1. c := a + int(b)  // c = 200
// 2. c := int8(a) + b // c = -56

// 错误写法
c := a + b
fmt.Println(c)
```

显示

```
C:\Users\shineyork>go run shineyork/02
# shineyork/02
D:\phpStudy\PHPTutorial\WWW\go\src\shineyork\02\main.go:8:9: invalid operation: a + b (mismatched types int and int8)
```

3. 只有相同底层类型的变量之间可以进行相互转换（如将 int16 类型转换成 int32 类型），不同底层类型的变量相互转换时会引发编译错误（如将 bool 类型转换为 int 类型）；

4. String与int之间转换：

```
* int => string

        ```go
        var a int = 100
        // 0. 非正常操作
        fmt.Println(string(a)) // d

        // 1. 采用fmt.Sprintf("%d", n)
```



```


```
        b := fmt.Sprintf("%d", a) // 效率最低
        fmt.Println(b) // 100

        // 2. 采用strconv.Itoa(n)
        c := strconv.Itoa(a)
        fmt.Println(c)

        // 3. 采用strconv.FormatInt(int64(a), 10)
        d := strconv.FormatInt(int64(a), 10) // 效率最高
        fmt.Println(d)
        ```
* string => int

        ```go
        var a string = "100"
        // 1. string转成int：
        b, _ := strconv.Atoi(a)
        fmt.Println(b)
        // 2. string转成int64：
        c, _ := strconv.ParseInt(a, 10, 64)
        fmt.Println(c)
        ```
```

5. parse 解析： Parse 系列函数用于将字符串转换为指定类型的值，其中包括 ParseBool()、ParseFloat()、ParseInt()、ParseUint()。
   - ParseBool() 函数用于将字符串转换为 bool 类型的值，它只能接受 1、0、t、f、T、F、true、false、True、False、TRUE、FALSE，其它的值均返回错误：func ParseBool(str string) (value bool, err error)

```
changBool, err := strconv.ParseBool("1")
```

if err == nil { fmt.Printf("str1: %v\n", err) } //字符串转化为bool型 fmt.Printf("type:%T value:%#v\n", changBool, changBool) ```

```
* ParseInt() 函数用于返回字符串表示的整数值（可以包含正负号） ParseUint() 函数的功能类似于 ParseInt() 函数，但 ParseUint() 函数不接受正负号，用于无符号整型

```go
//base 指定进制，取值范围是 2 到 36。如果 base 为 0，则会从字符串前置判断，“0x”是 16 进制，“0”是 8 进制，否则是 10 进制。
//bitSize 指定结果必须能无溢出赋值的整数类型，0、8、16、32、64 分别代表 int、int8、int16、int32、int64。
//返回的 err 是 *NumErr 类型的，如果语法有误，err.Error = ErrSyntax，如果结果超出类型范围 err.Error = ErrRange
//转化为整型ParseInt()
changeInt, err := strconv.ParseInt("-11", 10, 0)
fmt.Printf("%#v\n", changeInt)
changeUint, err := strconv.ParseUint("10", 10, 0)
if err == nil {
        printf("err:%v\n", err)
}
fmt.Printf("value:%v\n", changeUint)
```

* ParseFloat() 函数用于将一个表示浮点数的字符串转换为 float 类型 func ParseFloat(s string, bitSize int) (f float64, err error)
```

7. format 格式： Format 系列函数实现了将给定类型数据格式化为字符串类型的功能，其中包括 FormatBool()、FormatInt()、FormatUint()、FormatFloat()。
8. append 附加 Append 系列函数用于将指定类型转换成字符串后追加到一个切片中，其中包含 AppendBool()、AppendFloat()、AppendInt()、AppendUint()。

```
// 将转换为10进制的string，追加到slice中
b10 = strconv.AppendInt(b10, -42, 10)
fmt.Println(string(b10))
b16 := []byte("int (base 16):")
b16 = strconv.AppendInt(b16, -42, 16)
fmt.Println(string(b16))
```

6. go语言声明与作用域

6.1. 作用域


```

1. 声明在函数内部，是函数的本地值，类似private
2. 声明在函数外部，是对当前包可见(包内所有.go文件都可见)的全局值，类似protect
3. 声明在函数外部且首字母大写是所有包可见的全局值,类似public

6.2. go语言声明

有四种主要声明方式：

```
var（声明变量），const（声明常量），type（声明类型），func（声明函数）。
```

> 实例：变量 var

语法

```
// 变量
var 变量名 变量类型
var (
    a int
    b int
)
```

例子

```
// 正常定义方式
var a int = 0
fmt.Println("常定义方式 ： ", a)

// 根据值得知类型
var c = 0
fmt.Println("根据值得知类型 ： ", c)

// := 定义方式
b := 0
fmt.Println(" := 定义方式 ： ", b)

// 批量定义
var (
  v_a int = 201
  v_b int
)
v_b = 202
fmt.Printf("批量声明变量 v_a:%a | v_b:%b \n", v_a, v_b)
```

> 实例：常量 const

语法

```
const identifier [type] = value
const (
    n1 = iota //0
    n2        //1
    n3        //2
    n4        //3
)
const c_name1, c_name2 = value1, value2
```

例子

```
const GNAME = "sixstar"

const (
```



```
  a = 1
  b = 2
)

const (
        a1 = 10
        b1
        c1
)

fmt.Println("GNAME", GNAME)
fmt.Println("a, b: ", a, b)
fmt.Println("a1, b1, c1: ", a1, b1, c1)
```

iota

iota，特殊常量，可以认为是一个可以被编译器修改的常量。

iota是go语言的常量计数器，只能在常量的表达式中使用。 iota在const关键字出现时将被重置为0。const中每新增一行常量声明将使iota计数一次(iota可理解为const语句块中的行索引)。 使用iota能简化定义，在定义枚举时很有用

```
const (
        a1 = iota
        b1
        c1
)
const (
        a1 = iota
        b1 = iota
        c1 = iota
)
fmt.Println("a1, b1, c1: ", a1, b1, c1)
// 输出都是 ：  a1, b1, c1:  0 1 2

const (
        a = iota //0
        b        //1
        c        //2
        d = "ha" //独立值，iota += 1
        e        //"ha"   iota += 1
        f = 100  //iota +=1
        g        //100  iota +=1
        h = iota //7,恢复计数
        i        //8
)
fmt.Println(a, b, c, d, e, f, g, h, i)
// 输出是 ：0 1 2 ha ha 100 100 7 8

const (
  a, d = iota + 1, iota + 10
  b, e
  c, f
)
fmt.Println(a, b, c, d, e, f)
// 输出是 ：1 2 3 10 11 12
```

> 实例：函数 func

特点

- 无需声明原型。
- 支持不定 变参。
- 支持多返回值。

- 支持命名返回参数。
- 支持匿名函数和闭包。
- 函数也是一种类型，一个函数可以赋值给变量。
- 不支持 嵌套 (nested) 一个包不能有两个名字一样的函数。
- 不支持 重载 (overload)
- 不支持 默认参数 (default parameter)。

语法

```
func function_name( [parameter list] ) [return_types] {
   函数体
}
```

例子

```
func main() {
        a, b := test(1, 2, "sum")
        fmt.Println("a,b : ", a, b)
   dome()
}
func test(x, y int, s string) (int, string) {
        // 类型相同的相邻参数，参数类型可合并。 多返回值必须用括号。
        n := x + y
        return n, s
}
func dome() {
        fmt.Println("dome")
}
```

> 实例：结构体 type

语法

```
type 类型名 struct {
    字段名 字段类型
    字段名 字段类型
    …
}
```

其中：

1. 类型名：标识自定义结构体的名称，在同一个包内不能重复。
2. 字段名：表示结构体字段名。结构体中的字段名必须唯一。
3. 字段类型：表示结构体字段的具体类型。

对于学习过其他语言的同志来说可以理解为是对象属性，只是不存在方法

例子

```
type go_class struct {
        class_name  string
        class_stage int
}

func main() {
        fmt.Println(go_class{"go class", 1})

        var g go_class
        g.class_name = "shineyork go class"
```



```
        g.class_stage = 1
        fmt.Println("课程名：", g.class_name)
        fmt.Println("课程阶段：", g.class_stage)
}
```

7. go语中的init与main函数

7.1. main函数

```
Go语言程序的默认入口函数(主函数)：func main()
函数体用 { } 一对括号包裹。

func main(){
    //函数体
}
```

main函数只能用于main包中，且只能定义一个。如同PHP框架中的index.php

7.2. init函数

go语言中init函数用于包(package)的初始化，该函数是go语言的一个重要特性。可以理解为如同其他语言中的面向对象

```
func init()  {

}
```

有下面的特征：

1. init函数是用于程序执行前做包的初始化的函数，比如初始化包里的变量等
2. 每个包可以拥有多个init函数
3. 包的每个源文件也可以拥有多个init函数
4. 同一个包中多个init函数的执行顺序go语言没有明确的定义(说明)
5. 不同包的init函数按照包导入的依赖关系决定该初始化函数的执行顺序
6. init函数不能被其他函数调用，而是在main函数执行之前，自动被调用

例子

```
var name string

func init() {
        name = "shineyork"
}

func main() {
        fmt.Println(name)
}
```

对同一个go文件的init()调用顺序是从上到下的。

对同一个package中不同文件是按文件名字符串比较"从小到大"顺序调用各文件中的init()函数。

对于不同的package，如果不相互依赖的话，按照main包中"先import的后调用"的顺序调用其包中的init()，如果package存在依赖，则先调用最早被依赖的package中的init()，最后调用main函数。

如果init函数中使用了println()或者print()你会发现在执行过程中这两个不会按照你想象中的顺序执行。这两个函数官方只推荐在测试环境中使用，对于正式环境不要使用。

8. import 和 package 的使用

golang使胞package来管理定义模块,可以使用import关键字来导入使用。

- 如果导入的是go自带的包，则会去安装目录$GOROOT/src按包路径加载，如fmt包
- 如果是我们go get安装或自定义的包，则会去$GOPATH/src下加载

8.1. package的定义

注意 相信对很多phper来说遵循PSR4的Namespace会将与路径紧密相关命名空间也作为类名的一部分,而golang则只将模块目录文件夹名作为包名,前面的路径只是用来导入而和包名无关,还是有一-点点需要 注意的。.

package的存放位置是以$GOPATH/src作为根目录，然后灵活的按照目录去组织，且胞名需与最后一级目录名一致。

例如我们自定义baz包,包模块的存放位置则为$GOPATH/src/foo/bar/baz, baz 包的源码都存放在此目录下，foo/bar/baz则作为包路径被import载入。

我们需要规范的将baz包中源码的package定义为baz,就定义好一个可import载入的的包了。

`shineyork\01\util\log.go`

```
package util

import "fmt"

func Print(args string) {
        fmt.Println(args)
}
```

`shineyork\01\util\sms\email.go`

```
package sms

import "fmt"

func Email() {
        fmt.Println("sms email")
}
```

注意作用域

1. 声明在函数内部，是函数的本地值，类似private
2. 声明在函数外部，是对当前包可见(包内所有.go文件都可见)的全局值，类似protect
3. 声明在函数外部且首字母大写是所有包可见的全局值,类似public

8.2. import的定义

> 常用方式

```
import (
        "shineyork/01/util"
        "shineyork/01/util/sms"
)

func main() {
        util.Print("shineyork")
        sms.Email()
}
```

普通导入就是按照加载机制，将要使用的包导入进来，然后使用 packageName.MethodName 的方式调用包内的方法即可。注意如果要包方法在其他包中可以调用，包方法需要首字母大写，例如：fmt.Println() fmt.Printf()。

> 别名方式

如果两个包的包名存在冲突时，我们可以使用别名导入来解决。

`shineyork/sms`

```
package sms
```



```
import "fmt"

func Email() {
        fmt.Println("this is sms email")
}
```

`main`

```
import (
        "shineyork/01/util"
        utilSms "shineyork/01/util/sms"
        "shineyork/sms"
)

func main() {
        util.Print("shineyork")
        utilSms.Email()
        sms.Email()
}
```

9. 下划线

9.1. 在包中运用

先看一下没有用下划线的情况

`shineyork/sms/email.go`

```
package sms

import "fmt"

func init() {
        fmt.Println("this is sms/email.go init")
}

func Email() {
        fmt.Println("this is sms email")
}
```

`shineyork/01/util/sms/email.go`

```
package sms

import "fmt"

func init() {
        fmt.Println("this is util/sms/email.go init")
}

func Email() {
        fmt.Println("sms email")
}
```

```
package main

import (
        "shineyork/01/util"
        utilSms "shineyork/01/util/sms"
        "shineyork/sms"
)

func main() {
        util.Print("shineyork")
```


```
        utilSms.Email()
        sms.Email()
}
```

测试

```
C:\Users\shineyork>go run shineyork/01/
this is util log.go init
this is util/sms/email.go init
this is sms/email.go init
shineyork
sms email
this is sms email
```

import 下划线（如：import hello/imp）的作用：当导入一个包时，该包下的文件里所有init()函数都会被执行，然而，有些时候我们并不需要把整个包都导入进来，仅仅是是希望它执行init()函数而已。这个时候就可以使用 import 引用该包。即使用【import _ 包路径】只是引用该包，仅仅是为了调用init()函数，所以无法通过包名来调用包中的其他函数。 示例：

```
import "database/sql"
import _ "github.com/go-sql-driver/mysql"
```

9.2. 下划线在代码中运用

```
package main

import "fmt"

func main() {
        a, _ := compare(1)
        fmt.Println(a)
}

func compare(a int) (int, error) {
        return a, nil
}
```

解释1：

1. 下划线意思是忽略这个变量.
2. 比如compare,返回值为int, error
3. 普通写法是a,err := compare(1)
4. 如果此时不需要知道返回的错误值
5. 就可以用a，_ := compare(1)
6. 如此则忽略了error变量

解释2：

1. 占位符，意思是那个位置本应赋给某个值，但是咱们不需要这个值。
2. 所以就把该值赋给下划线，意思是丢掉不要。
3. 这样编译器可以更好的优化，任何类型的单个值都可以丢给下划线。
4. 这种情况是占位用的，方法返回两个结果，而你只想要一个结果。
5. 那另一个就用"_"占位，而如果用变量的话，不使用，编译器是会报错的。

10. 命令

可以在命令行执行go命令查看相关的Go语言命令

```
C:\Users\shineyork>go
Go is a tool for managing Go source code.

Usage:
```



```
        go <command> [arguments]

The commands are:

        bug         start a bug report
        build       compile packages and dependencies
        clean       remove object files and cached files
        doc         show documentation for package or symbol
        env         print Go environment information
        fix         update packages to use new APIs
        fmt         gofmt (reformat) package sources
        generate    generate Go files by processing source
        get         add dependencies to current module and install them
        install     compile and install packages and dependencies
        list        list packages or modules
        mod         module maintenance
        run         compile and run Go program
        test        test packages
        tool        run specified go tool
        version     print Go version
        vet         report likely mistakes in packages

Use "go help <command>" for more information about a command.

Additional help topics:

        buildconstraint build constraints
        buildmode       build modes
        c               calling between Go and C
        cache           build and test caching
        environment     environment variables
        filetype        file types
        go.mod          the go.mod file
        gopath          GOPATH environment variable
        gopath-get      legacy GOPATH go get
        goproxy         module proxy protocol
        importpath      import path syntax
        modules         modules, module versions, and more
        module-get      module-aware go get
        module-auth     module authentication using go.sum
        module-private  module configuration for non-public modules
        packages        package lists and patterns
        testflag        testing flags
        testfunc        testing functions

Use "go help <topic>" for more information about that topic.
```

go env用于打印Go语言的环境信息。

go run命令可以编译并运行命令源码文件。

go get可以根据要求和实际情况从互联网上下载或更新指定的代码包及其依赖包，并对它们进行编译和安装。

go build命令用于编译我们指定的源码文件或代码包以及它们的依赖包。

go install用于编译并安装指定的代码包及它们的依赖包。

go clean命令会删除掉执行其它命令时产生的一些文件和目录。

go doc命令可以打印附于Go语言程序实体上的文档。我们可以通过把程序实体的标识符作为该命令的参数来达到查看其文档的目的。

go test命令用于对Go语言编写的程序进行测试。

go list命令的作用是列出指定的代码包的信息。

go fix会把指定代码包的所有Go语言源码文件中的旧版本代码修正为新版本的代码。

go vet是一个用于检查Go语言源码中静态错误的简单工具。

go tool pprof命令来交互式的访问概要文件的内容。

.