



# 如何实现分工协作及功能复用

---



张华

64174234@qq.com

# 内容

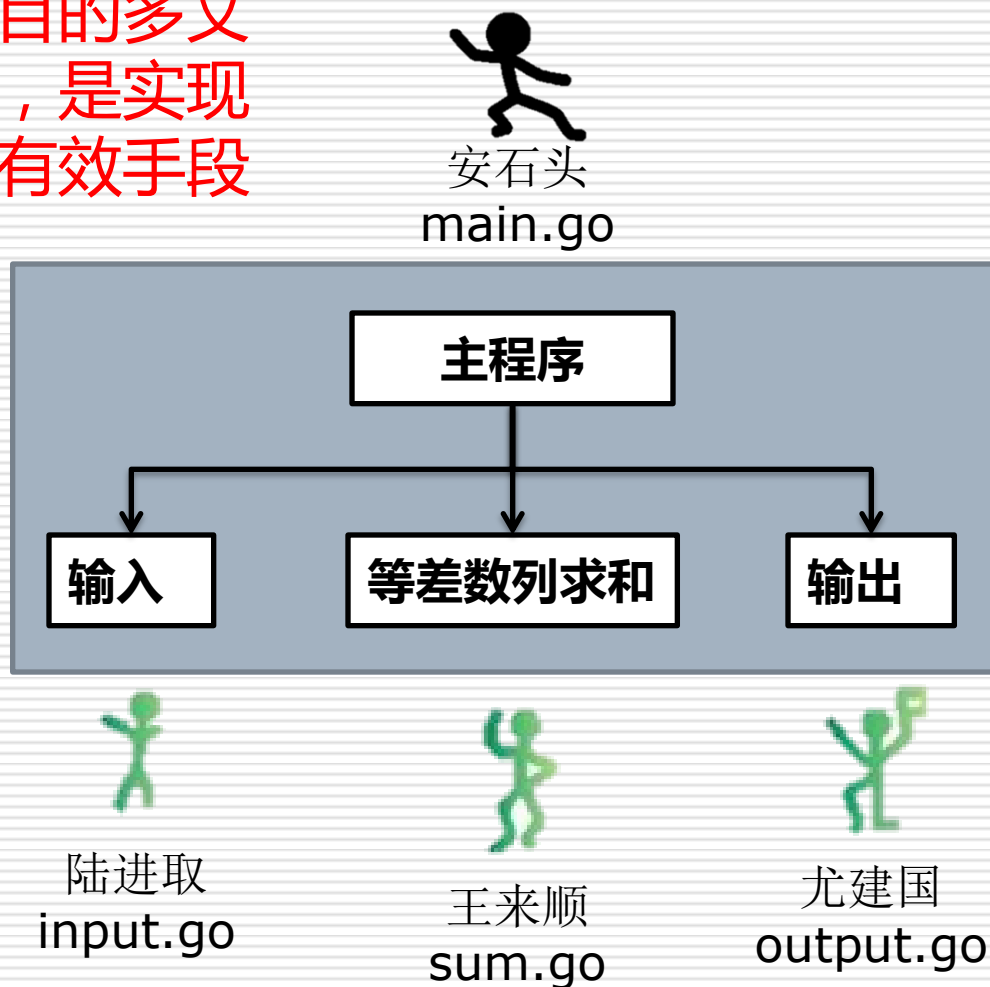
---

- 1.1 如何实现基于项目的多文件编程
- 1.2 如何进行单元测试
- 1.3 如何进行性能测试
- 1.4 如何实现真正的协同开发
- 1.5 如何实现功能复用
- 1.6 思考

# 1.1如何实现基于项目的多文件编程

□ 模块化编程不仅可以提高代码结构的清晰度，还可以促进团队协作水平，提高开发效率。

■ 采用基于项目的多文件开发模式，是实现分工协作的有效手段



# 1.1如何实现基于项目的多文件编程(续)

## □ 建立项目

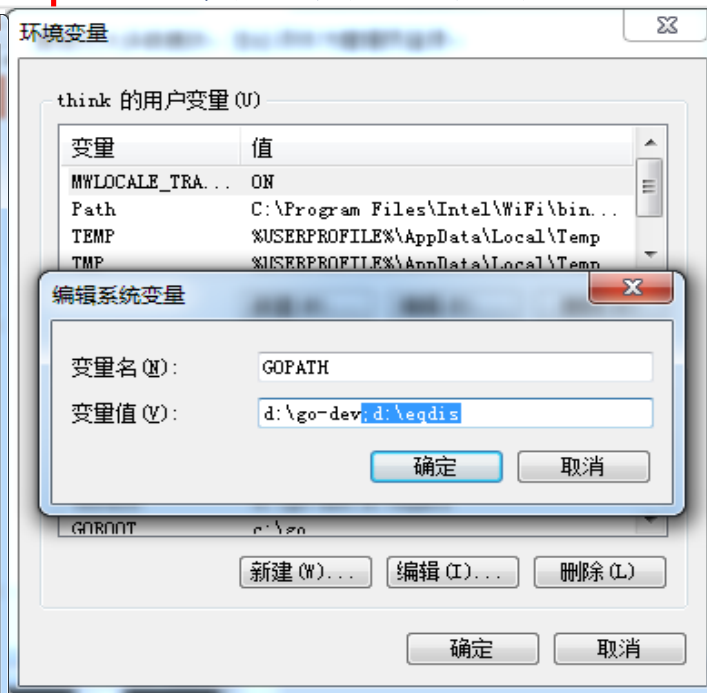
- 建立项目文件夹，明确项目文件存放的具体位置

- 这里在d盘建立了名为eqdis的项目文件夹，同时建立了相应的src、pkg及bin子目录

- 为系统变量GOPATH增加一个值d:\eqdis

- 这样一个名称为eqdis的项目就建立成功了

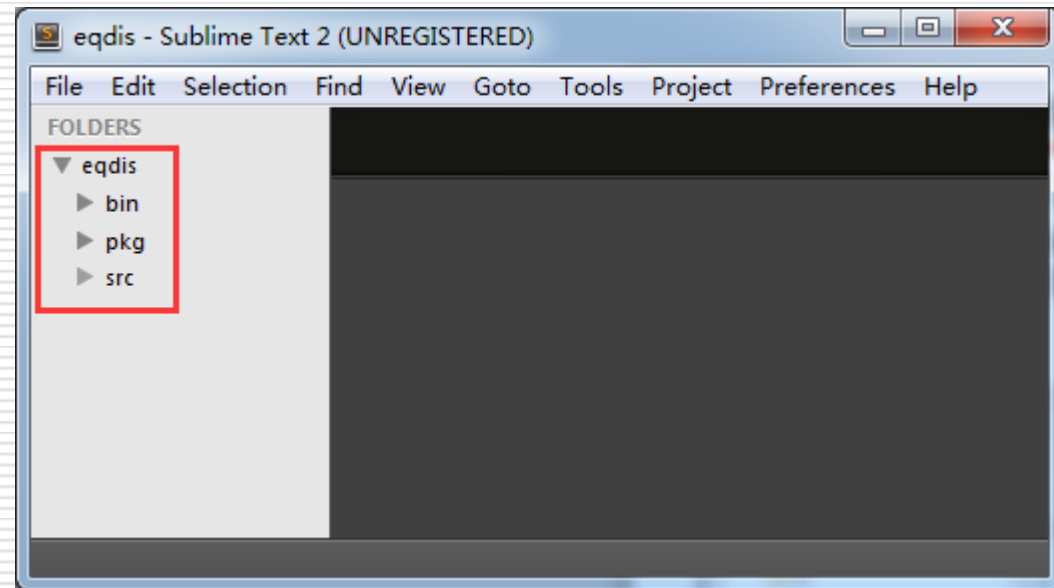
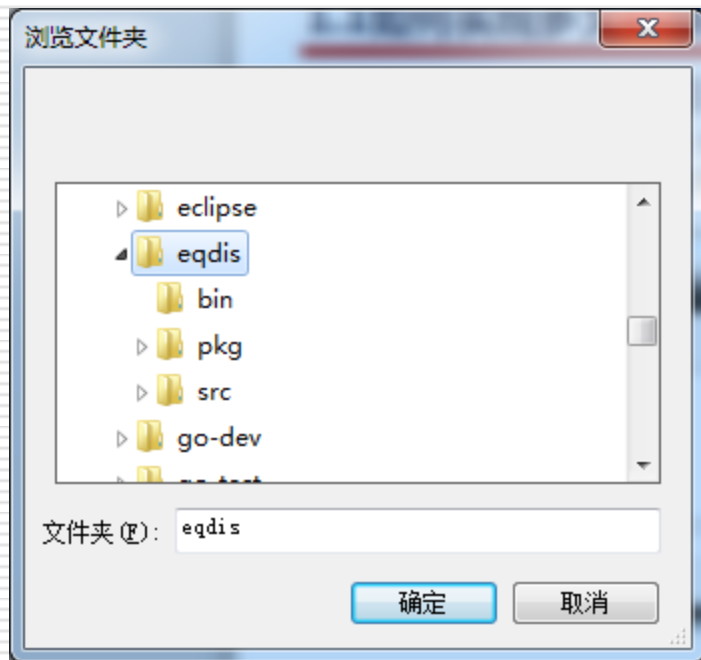
```
C:\Windows\system32\cmd.exe
D:\>md eqdis
D:\>md eqdis\src
D:\>md eqdis\pkg
D:\>md eqdis\bin
D:\>tree eqdis
Folder PATH listing
Volume serial number is 0005-1A3A
D:\EQDIS
├── bin
├── pkg
└── src
D:\>
```



□ GOPATH可以配置多个路径，每个路径对应一个项目，项目间用;分隔

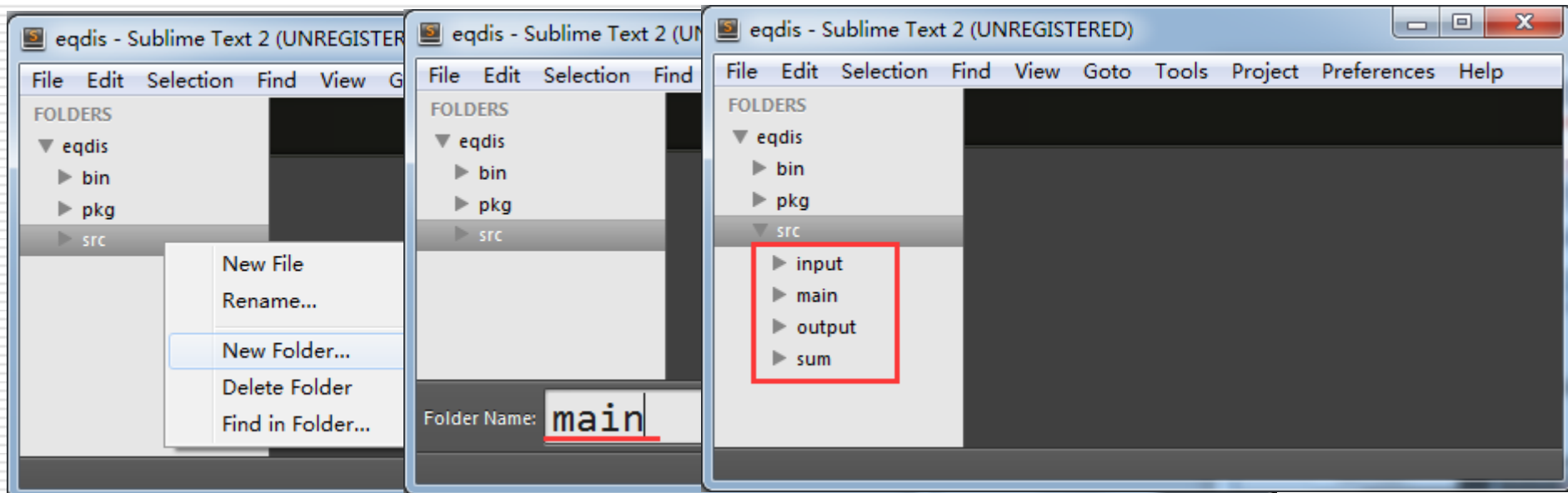
# 1.1如何实现基于项目的多文件编程(续)

- ❑ 在Sublime 中打开项目目录
  - Project/Add Folder to Project... 在弹出的对话框中选择上一步建立的项目目录 d:\eqdis 然后点击确定



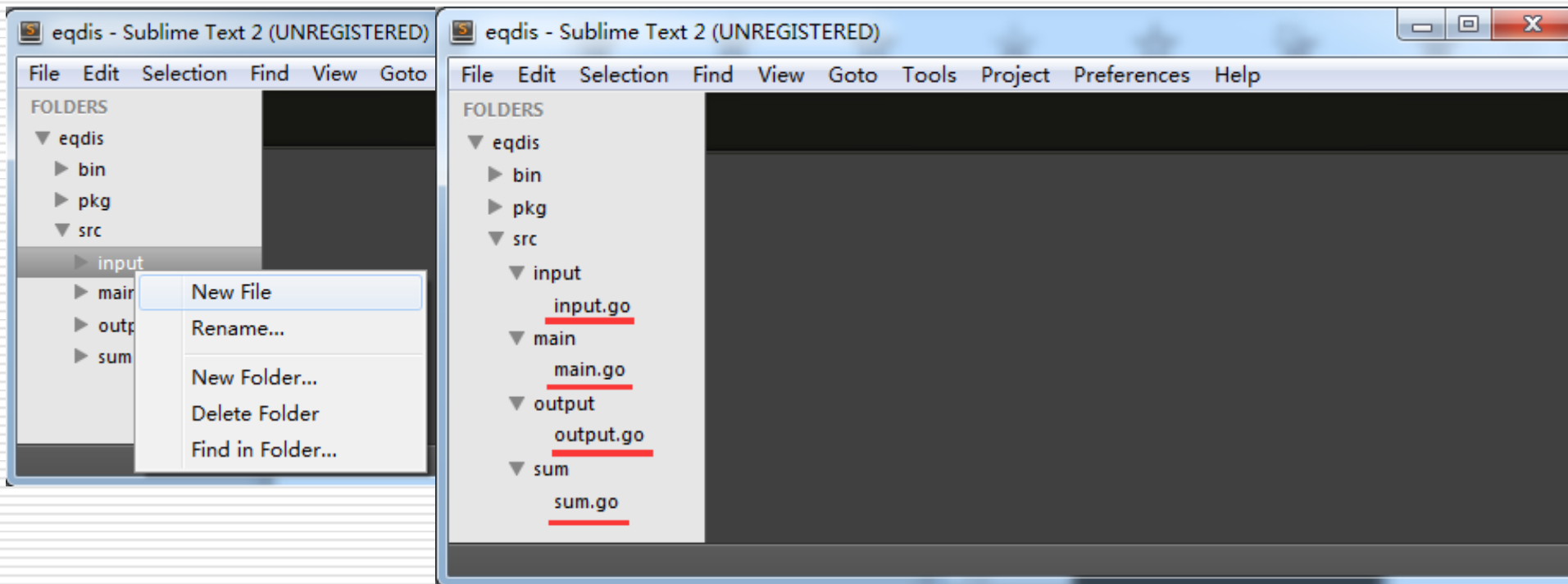
# 1.1如何实现基于项目的多文件编程(续)

- 根据分工，通过右键src\New Folder为每个开发者在src下建立各自的工作目录，分别为main、input、sum、output



# 1.1如何实现基于项目的多文件编程(续)

- 每个开发者右键各自的工作目录\选择New File 建立自己的go文件,分别为 main.go、input.go、sum.go和output.go



# 1.1如何实现基于项目的多文件编程(续)

## □ input.go

```
1 package input
2
3 import "fmt"
4
5 /*****
6 录入首项a1,项数num,公差dis
7 *****/
8 func Input(a1, num, dis int) (int, int, int) {
9     fmt.Println("Please input first term in a1")
10    fmt.Scanln(&a1) //录入首项a
11    fmt.Println("Please input term num in num")
12    fmt.Scanln(&num) //录入项数n
13    fmt.Println("Please input term equal difference in dis")
14    fmt.Scanln(&dis) //录入公差d
15    return a1, num, dis
16 }
```

每个程序都是由包组成的，包名为input。  
按照惯例包名与源码存放路径的最后一个目录一致

函数名首字母大写，具有public属性，  
可被外部调用

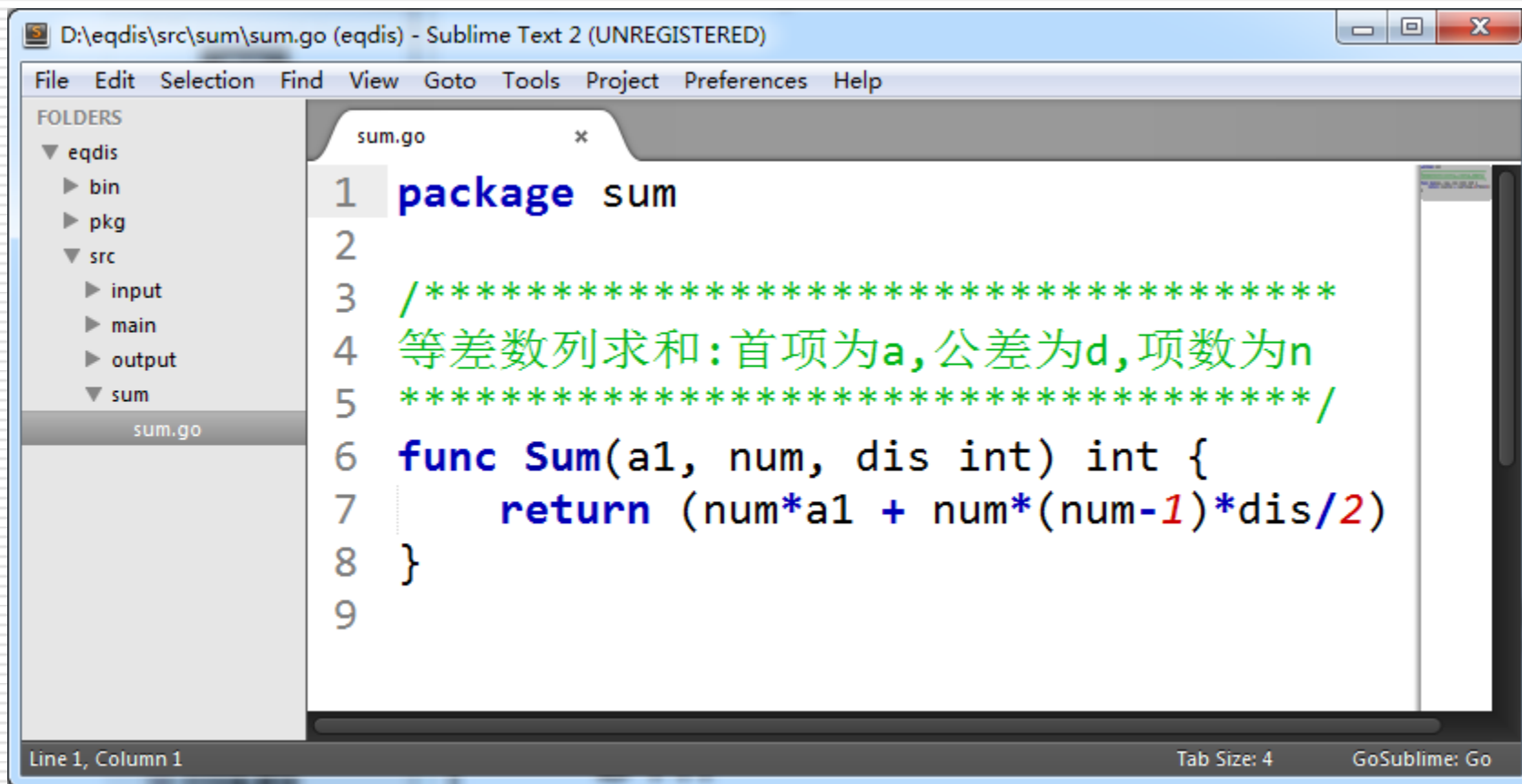
Line 1, Column 1

Tab Size: 4 GoSublime: Go



# 1.1如何实现基于项目的多文件编程(续)

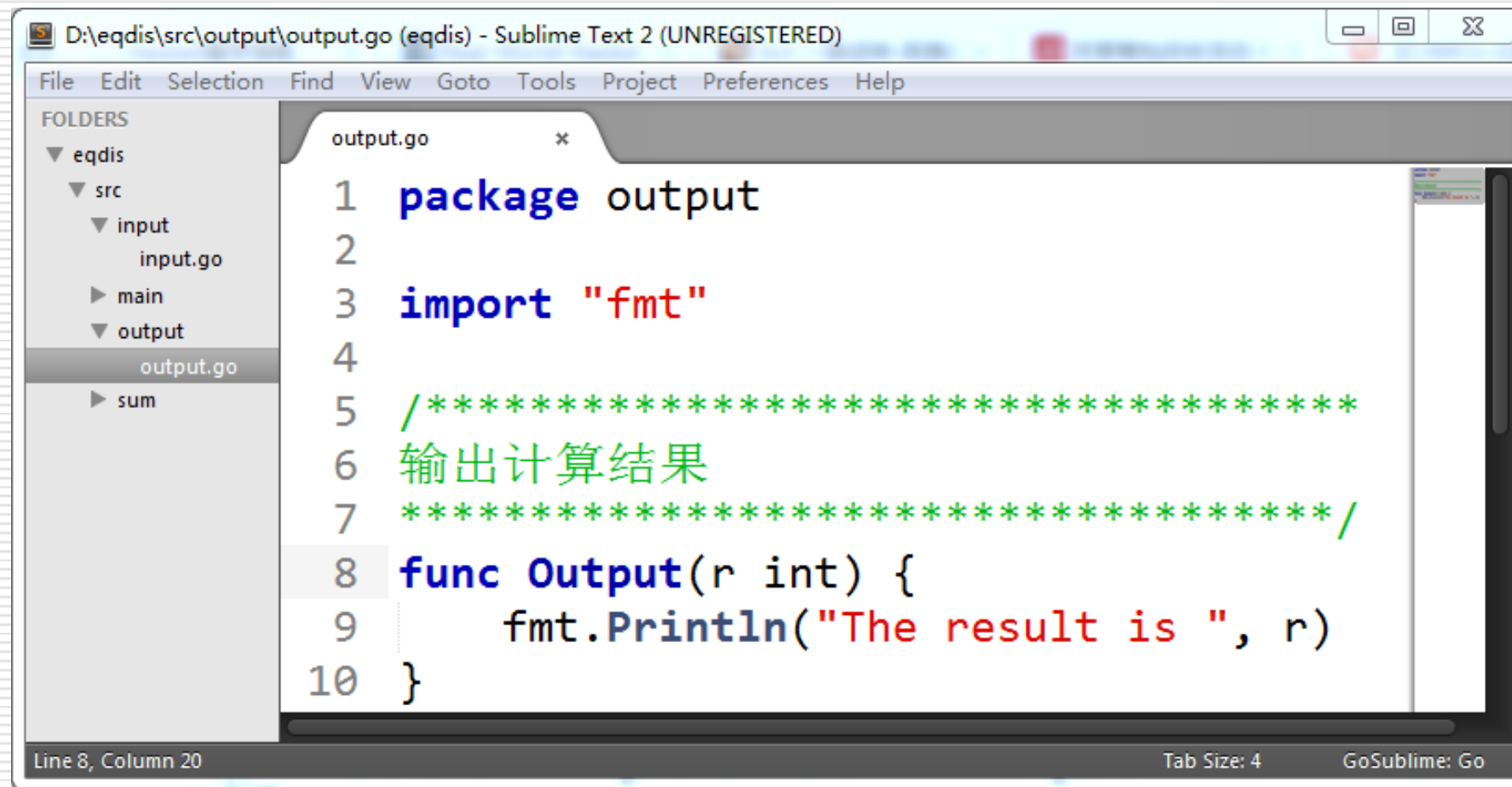
## □ sum.go



```
D:\eqdis\src\sum\sum.go (eqdis) - Sublime Text 2 (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
▼ eqdis
  ► bin
  ► pkg
  ▼ src
    ► input
    ► main
    ► output
    ▼ sum
      sum.go
sum.go
1 package sum
2
3 /*****
4 等差数列求和:首项为a,公差为d,项数为n
5 *****/
6 func Sum(a1, num, dis int) int {
7     return (num*a1 + num*(num-1)*dis/2)
8 }
9
Line 1, Column 1
Tab Size: 4
GoSublime: Go
```

# 1.1如何实现基于项目的多文件编程(续)

## □ output.go



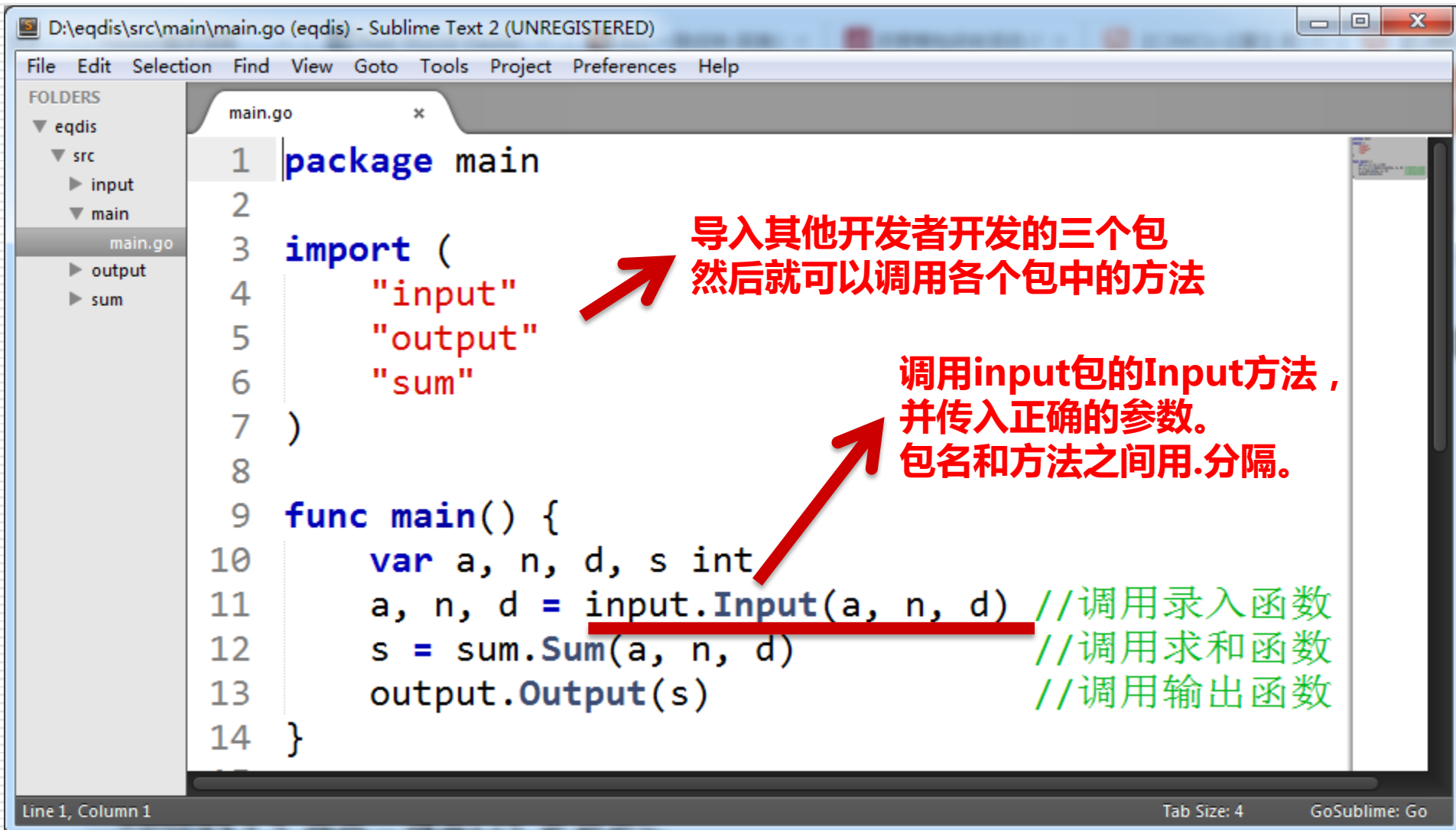
The screenshot shows the Sublime Text 2 editor interface. The title bar indicates the file path is D:\eqdis\src\output\output.go (eqdis) - Sublime Text 2 (UNREGISTERED). The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. On the left, a 'FOLDERS' sidebar shows a tree structure: eqdis (expanded) -> src (expanded) -> input (expanded) -> input.go, main, output (expanded) -> output.go (selected), and sum. The main editor area displays the content of output.go with line numbers 1 through 10. The code is as follows:

```
1 package output
2
3 import "fmt"
4
5 /*****
6 输出计算结果
7 *****/
8 func Output(r int) {
9     fmt.Println("The result is ", r)
10 }
```

The status bar at the bottom shows 'Line 8, Column 20', 'Tab Size: 4', and 'GoSublime: Go'.

# 1.1如何实现基于项目的多文件编程(续)

## □ main.go



```
D:\eqdis\src\main\main.go (eqdis) - Sublime Text 2 (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS
▼ eqdis
  ▼ src
    ► input
    ▼ main
      main.go
    ► output
    ► sum

main.go
1 package main
2
3 import (
4     "input"
5     "output"
6     "sum"
7 )
8
9 func main() {
10     var a, n, d, s int
11     a, n, d = input.Input(a, n, d) //调用录入函数
12     s = sum.Sum(a, n, d)           //调用求和函数
13     output.Output(s)              //调用输出函数
14 }
```

导入其他开发者开发的三个包  
然后就可以调用各个包中的方法

调用input包的Input方法，  
并传入正确的参数。  
包名和方法之间用.分隔。

Line 1, Column 1 Tab Size: 4 GoSublime: Go

# 1.1如何实现基于项目的多文件编程(续)

□ 如何生成可执行文件到bin及依赖的包文件到pkg

■ go install 包源码存放路径的最后一个目录

□ 生成可执行文件到bin，依赖的包文件到pkg

```
C:\Windows\system32\cmd.exe

D:\eqdis\src>tree /f d:\eqdis
Folder PATH listing
Volume serial number is 0005-1A3A
D:\EQDIS
├── bin
├── pkg
└── src
    ├── input
    │   └── input.go
    ├── main
    │   └── main.go
    ├── output
    │   └── output.go
    └── sum
        └── sum.go
```

生成之前的项目目录  
及文件结构

```
C:\Windows\system32\cmd.exe

D:\eqdis\src>go install main
D:\eqdis\src>tree /f d:\eqdis
Folder PATH listing
Volume serial number is 0005-1A3A
D:\EQDIS
├── bin
│   └── main.exe
├── pkg
│   └── windows_amd64
│       ├── input.a
│       ├── output.a
│       └── sum.a
└── src
    ├── input
    │   └── input.go
    ├── main
    │   └── main.go
    ├── output
    │   └── output.go
    └── sum
        └── sum.go
```

生成之后的项目目录  
及文件结构

依赖包编译后的扩展  
名为 .a

# 1.1如何实现基于项目的多文件编程(续)

## □ 如何生成包文件到pkg及可执行文件到bin？

- 在当前路径为src下，*go install 其他依赖包的最后一个路径*，可单独生成依赖包文件到pkg
  - go install input 可单独生成input.a到pkg
  - go install output 可单独生成output.a到pkg
  - go install sum 可单独生成sum.a到pkg
- 在建立项目目录的时候，可不必手工建立bin和pkg文件夹，go install成功后可以自动建立bin及pkg文件夹

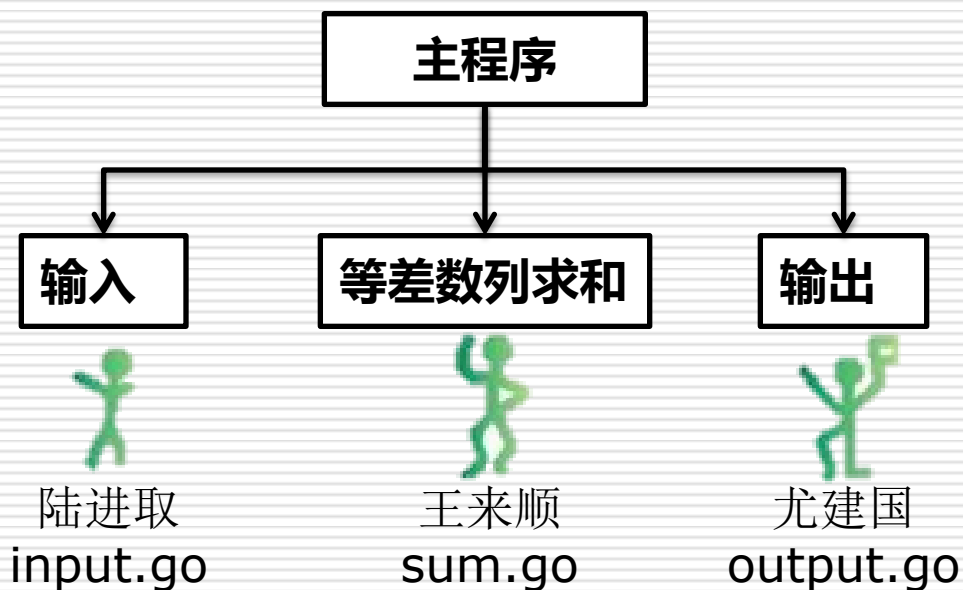
```
C:\Windows\system32\cmd.exe
D:\eqdis\src>tree /f d:\eqdis
Folder PATH listing
Volume serial number is 0005-1A3A
D:\EQDIS
├── src
│   ├── input
│   │   └── input.go
│   ├── main
│   │   └── main.go
│   ├── output
│   │   └── output.go
│   └── sum
│       └── sum.go
D:\eqdis\src>go install main
D:\eqdis\src>
```

**go install成功后自动  
生成bin及pkg目录**

**及相应的可执行  
文件和依赖包文件  
(这里没有展示相应  
的文件)**

```
C:\Windows\system32\cmd.exe
D:\eqdis\src>tree /f d:\eqdis
Folder PATH listing
Volume serial number is 0005-1A3A
D:\EQDIS
├── bin
│   └── main.exe
├── pkg
│   └── windows_amd64
│       ├── input.a
│       ├── output.a
│       └── sum.a
├── src
│   ├── input
│   │   └── input.go
│   ├── main
│   │   └── main.go
│   ├── output
│   │   └── output.go
│   └── sum
│       └── sum.go
```

# 1.2如何进行单元测试



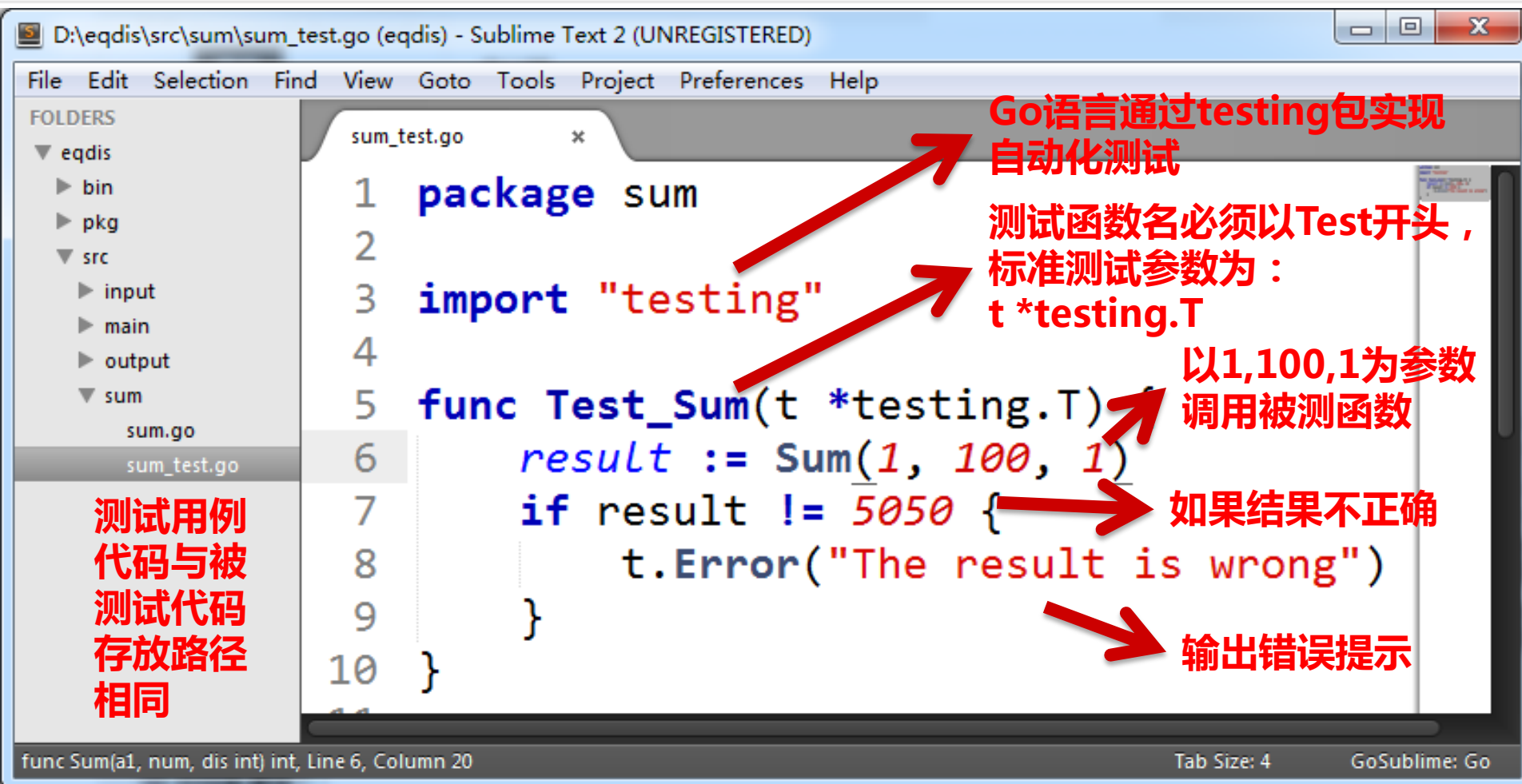
如何测试自己负责的代码功能对不对？

```
1 package sum
2
3 /**
4 等差数列求和:首项为a,公差为d,项数为n
5  */
6 func Sum(a1, num, dis int) int {
7     return (num*a1 + num*(num-1)*dis/2)
8 }
```

# 1.2如何进行单元测试(续)

## □ 建立sum.go的测试用例代码sum\_test.go

- 测试用例代码的文件名的格式必须是：**\*\_test.go**，\*最好与测试文件的主文件名相同，以方便阅读



The screenshot shows the Sublime Text 2 editor with the file `sum_test.go` open. The code is as follows:

```
1 package sum
2
3 import "testing"
4
5 func Test_Sum(t *testing.T) {
6     result := Sum(1, 100, 1)
7     if result != 5050 {
8         t.Error("The result is wrong")
9     }
10 }
```

Annotations and arrows pointing to the code:

- Go语言通过testing包实现自动化测试** (Go language implements automated testing through the testing package) - points to the `import "testing"` line.
- 测试函数名必须以Test开头，标准测试参数为：t \*testing.T** (Test function name must start with Test, standard test parameters are: t \*testing.T) - points to the `func Test_Sum(t *testing.T)` line.
- 以1,100,1为参数调用被测函数** (Call the function being tested with parameters 1, 100, 1) - points to the `Sum(1, 100, 1)` line.
- 如果结果不正确** (If the result is incorrect) - points to the `if result != 5050` line.
- 输出错误提示** (Output error message) - points to the `t.Error("The result is wrong")` line.

On the left side of the editor, the file explorer shows the directory structure:

- eqdis
  - bin
  - pkg
  - src
    - input
    - main
    - output
    - sum
      - sum.go
      - sum\_test.go

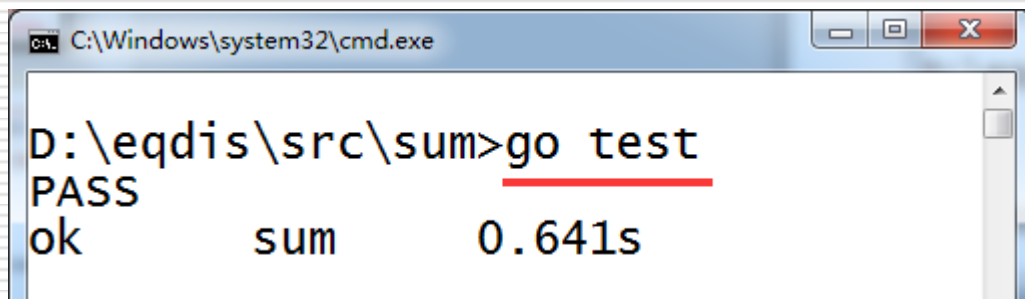
At the bottom left, a red box contains the text: **测试用例代码与被测试代码存放路径相同** (Test case code and code being tested are stored in the same path).

At the bottom of the window, the status bar shows: `func Sum(a1, num, dis int) int, Line 6, Column 20`, `Tab Size: 4`, and `GoSublime: Go`.

# 1.2如何进行单元测试(续)

□ 在测试用例文件目录下可执行如下命令进行测试

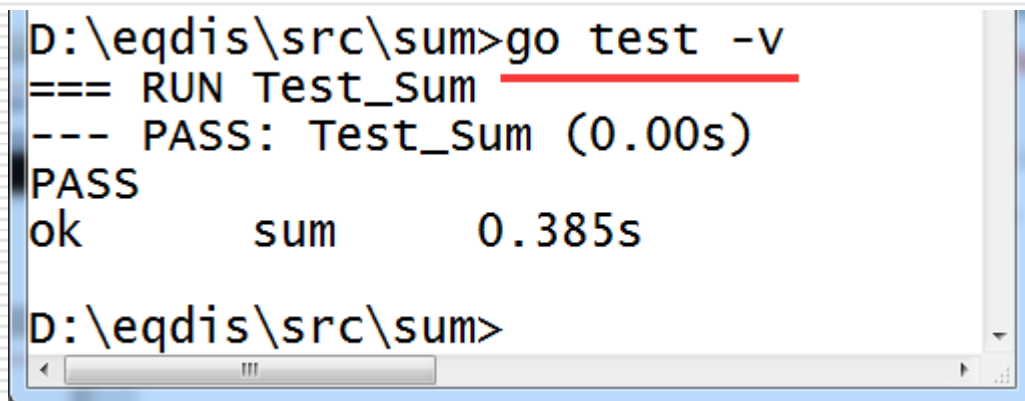
■ `go test`



A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The command prompt shows the directory 'D:\eqdis\src\sum' and the command 'go test' being executed. The output is 'PASS', 'ok', 'sum', and '0.641s'.

```
C:\Windows\system32\cmd.exe
D:\eqdis\src\sum>go test
PASS
ok      sum      0.641s
```

■ `go test -v`

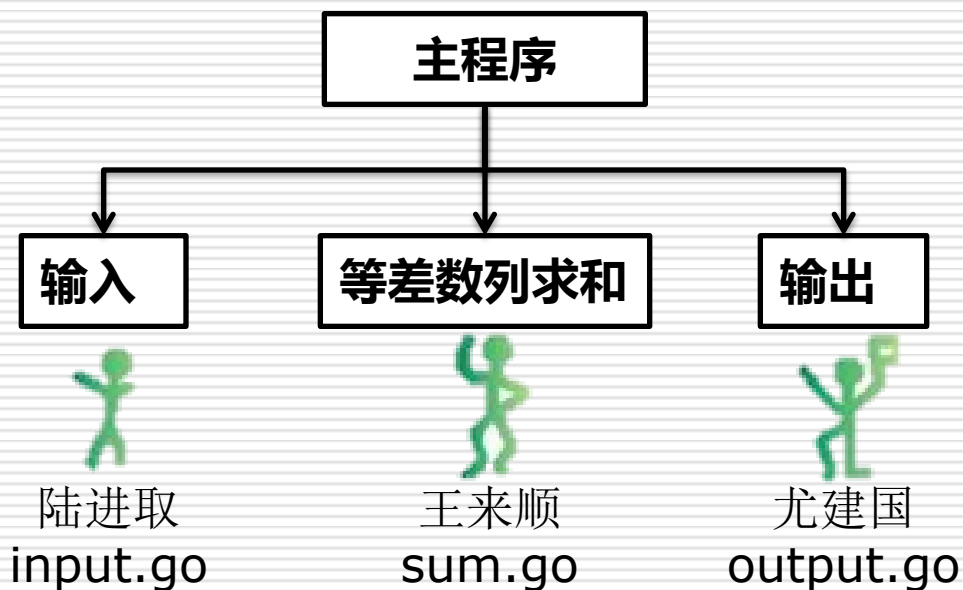


A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The command prompt shows the directory 'D:\eqdis\src\sum' and the command 'go test -v' being executed. The output is '=== RUN Test\_Sum', '--- PASS: Test\_Sum (0.00s)', 'PASS', 'ok', 'sum', and '0.385s'.

```
C:\Windows\system32\cmd.exe
D:\eqdis\src\sum>go test -v
=== RUN Test_Sum
--- PASS: Test_Sum (0.00s)
PASS
ok      sum      0.385s
D:\eqdis\src\sum>
```



# 1.3如何进行性能测试

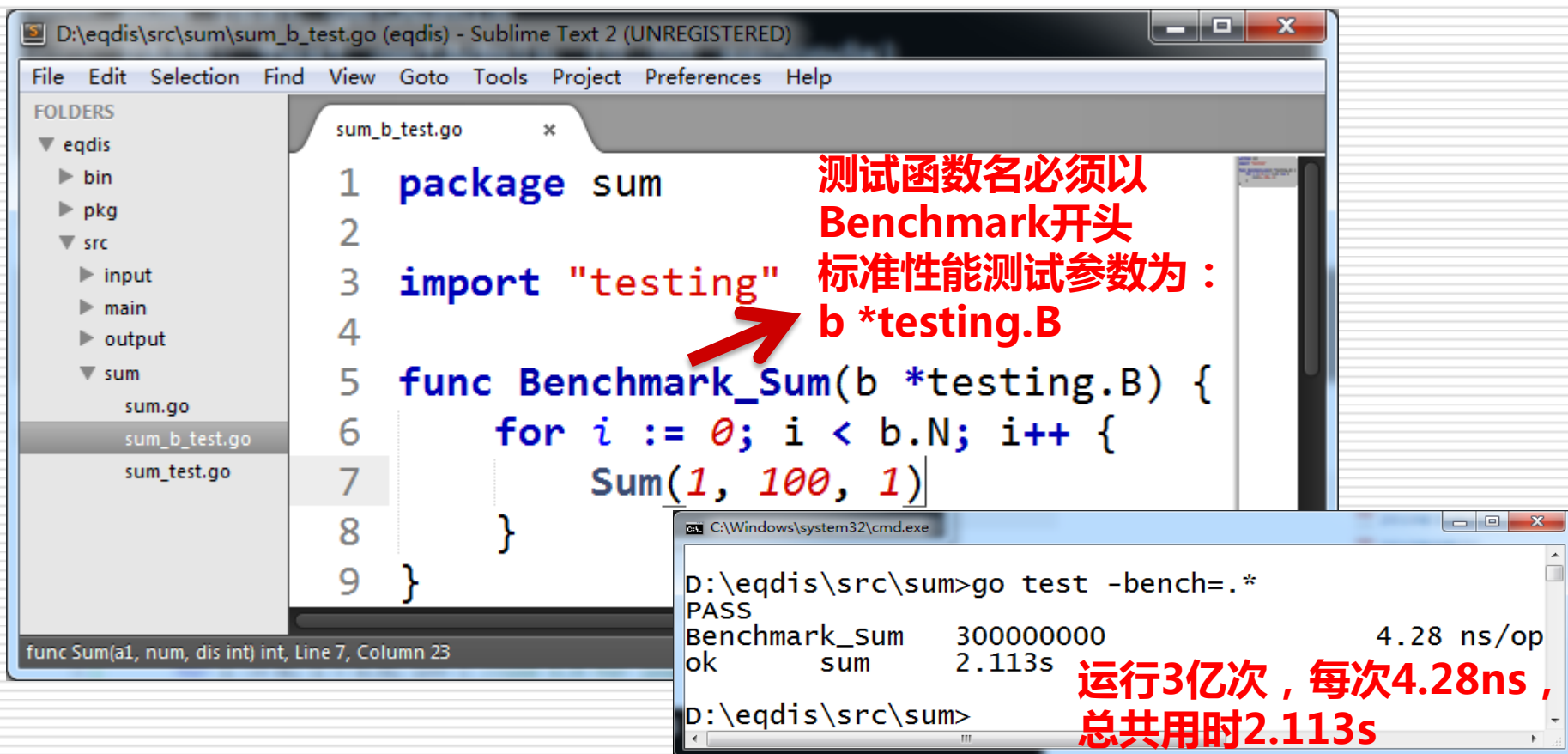


如何测试自己负责的代码的性能？

```
D:\eqdis\src\sum\sum.go (eqdis) - Sublime Text 2 (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
  ▼ eqdis
    ► bin
    ► pkg
    ▼ src
      ► input
      ► main
      ► output
      ▼ sum
        sum.go
sum.go
1 package sum
2
3 /**
4 等差数列求和:首项为a,公差为d,项数为n
5 */
6 func Sum(a1, num, dis int) int {
7     return (num*a1 + num*(num-1)*dis/2)
8 }
```

# 1.3如何进行性能测试(续)

- ❑ 单元测试保证了功能正确，但还需进行性能测试，主要评价在压力环境下的的响应时间、负载等指标
  - 建立sum.go的性能测试用例代码sum\_b\_test.go
  - 执行 `go test -bench= ".*"` 进行性能测试



The screenshot shows a Sublime Text 2 editor window with the file `sum_b_test.go` open. The code defines a package `sum`, imports `testing`, and implements a `Benchmark_Sum` function. A red arrow points to the `Benchmark_Sum` function name, with a red text overlay stating: "测试函数名必须以 Benchmark 开头 标准性能测试参数为: b \*testing.B".

```
1 package sum
2
3 import "testing"
4
5 func Benchmark_Sum(b *testing.B) {
6     for i := 0; i < b.N; i++ {
7         Sum(1, 100, 1)
8     }
9 }
```

Below the editor window, a Windows command prompt window shows the execution of the performance test:

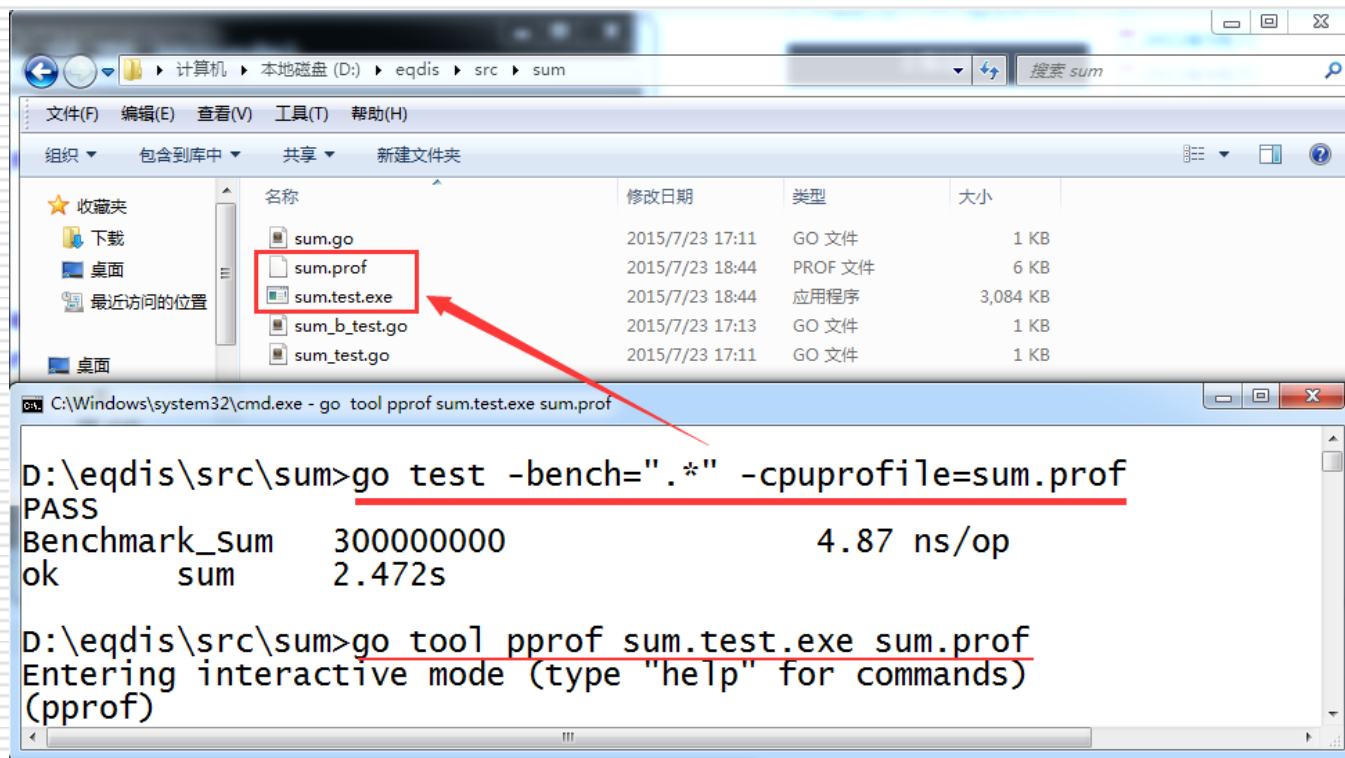
```
D:\eqdis\src\sum>go test -bench=.*
PASS
Benchmark_Sum    3000000000    4.28 ns/op
ok              sum        2.113s
```

Red text overlay on the command prompt output: "运行3亿次，每次4.28ns，总共用时2.113s".

# 1.3如何进行性能测试(续)

## □ 生成性能测试的CPU状态图

- 首先 `go test -bench=".*" -cpuprofile=sum.prof`
  - 生成可执行测试文件 `sum.test.exe` 和CPU性能文件 `sum.prof`
- 然后使用 `go tool pprof` 工具进入 `pprof` 命令模式分析数据：  
`go tool pprof sum.test.exe sum.prof`



The screenshot shows a Windows file explorer window displaying the contents of the directory `D:\eqdis\src\sum`. The files listed are:

| 名称            | 修改日期            | 类型      | 大小       |
|---------------|-----------------|---------|----------|
| sum.go        | 2015/7/23 17:11 | GO 文件   | 1 KB     |
| sum.prof      | 2015/7/23 18:44 | PROF 文件 | 6 KB     |
| sum.test.exe  | 2015/7/23 18:44 | 应用程序    | 3,084 KB |
| sum_b_test.go | 2015/7/23 17:13 | GO 文件   | 1 KB     |
| sum_test.go   | 2015/7/23 17:11 | GO 文件   | 1 KB     |

A red box highlights the `sum.prof` and `sum.test.exe` files, with a red arrow pointing to the command prompt below. The command prompt shows the execution of the following commands:

```
D:\eqdis\src\sum>go test -bench=".*" -cpuprofile=sum.prof
PASS
Benchmark_sum    300000000    4.87 ns/op
ok               sum        2.472s

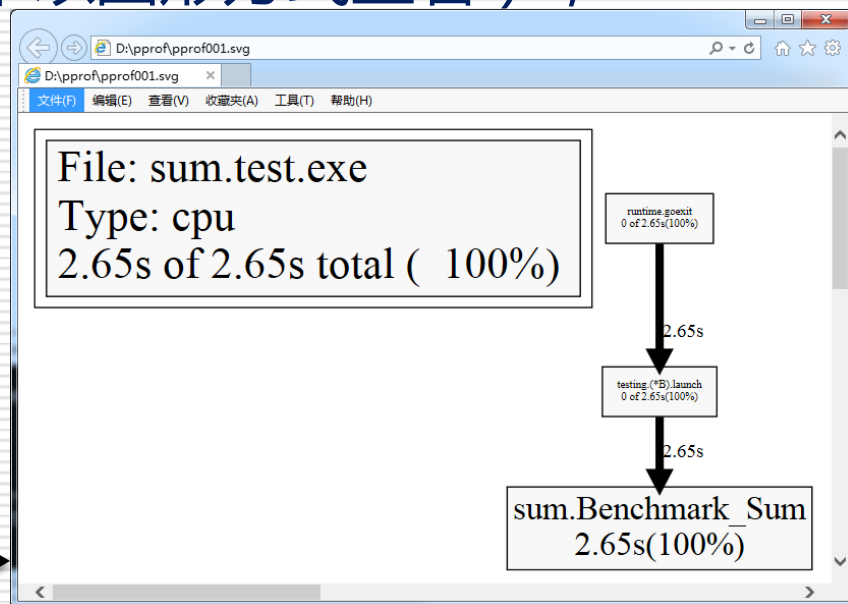
D:\eqdis\src\sum>go tool pprof sum.test.exe sum.prof
Entering interactive mode (type "help" for commands)
(pprof)
```

# 1.3如何进行性能测试(续)

## □ 生成性能测试的CPU状态图

- 输入web命令，在IE中查看cpu性能分析图（需要提前安装好graphviz才能在IE中以图形方式查看），

```
C:\Windows\system32\cmd.exe - go tool pprof sum.test.exe sum.prof
D:\eqdis\src\sum>go tool pprof sum.test.exe sum.prof
Entering interactive mode (type "help" for commands)
(pprof) web
(pprof)
```



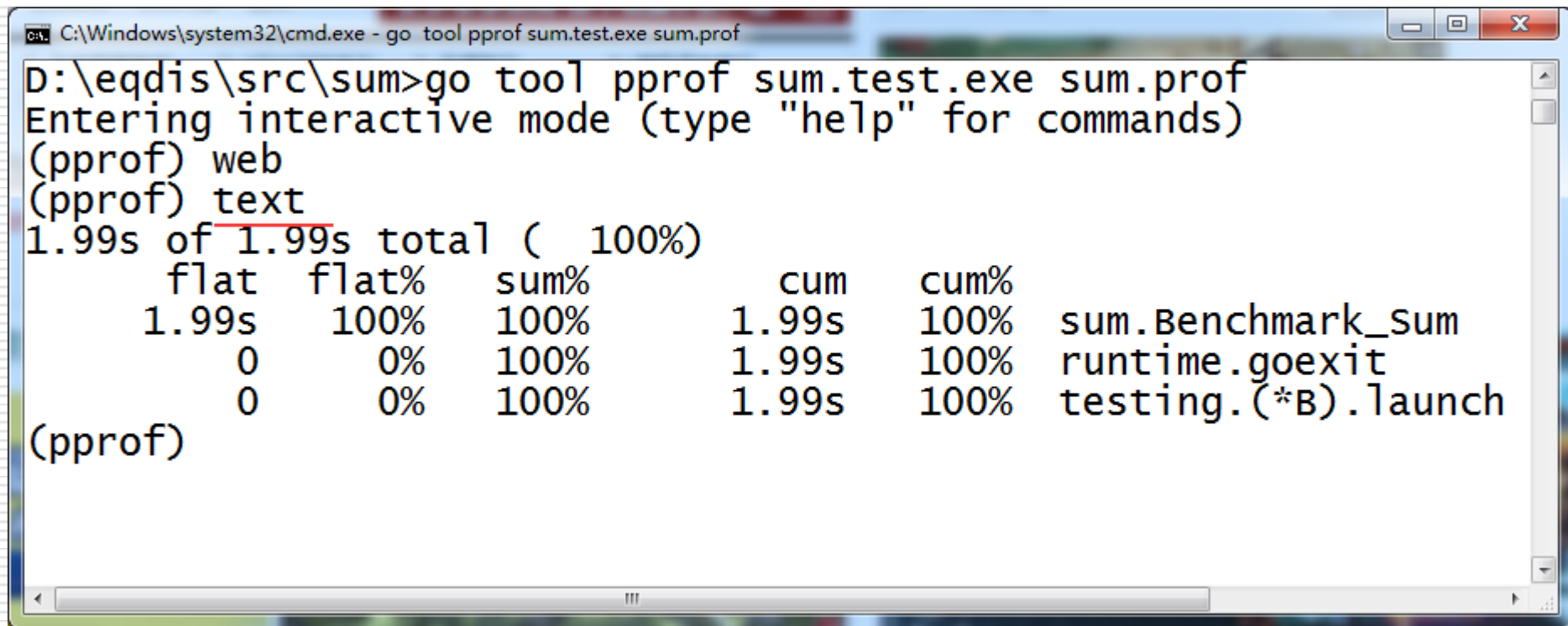
## ■ Graphviz

- 下载：<http://pan.baidu.com/s/1mgBZz1e>
- 安装好后，将安装路径\bin加入到系统变量path中

# 1.3如何进行性能测试(续)

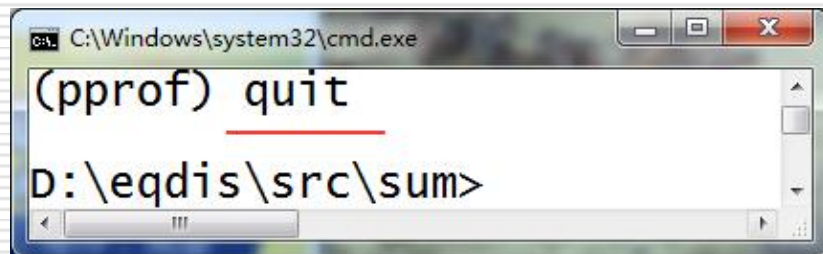
## □ 生成性能测试的CPU状态图

- 也可以输入text命令，以文本方式查看cpu性能文件



```
C:\Windows\system32\cmd.exe - go tool pprof sum.test.exe sum.prof
D:\eqdis\src\sum>go tool pprof sum.test.exe sum.prof
Entering interactive mode (type "help" for commands)
(pprof) web
(pprof) text
1.99s of 1.99s total ( 100%)
      flat  flat%   sum%        cum   cum%   sum.Benchmark_Sum
      1.99s   100%   100%       1.99s   100%   runtime.goexit
           0     0%   100%       1.99s   100%   testing.(*B).launch
           0     0%   100%       1.99s   100%
(pprof)
```

- 输入quit可退出pprof工具



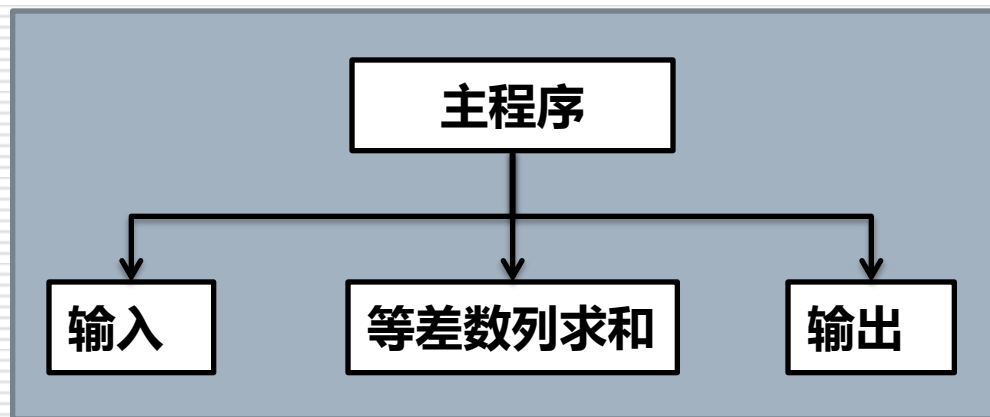
```
C:\Windows\system32\cmd.exe
(pprof) quit
D:\eqdis\src\sum>
```

# 1.4如何实现真正的协同开发

- 在项目研发过程中，单元测试和性能测试通过后，如何将各部分代码**汇总**形成完整的系统？各个项目成员又如何**获取**到完整的代码呢？



安石头  
main.go



陆进取  
input.go



王来顺  
sum.go

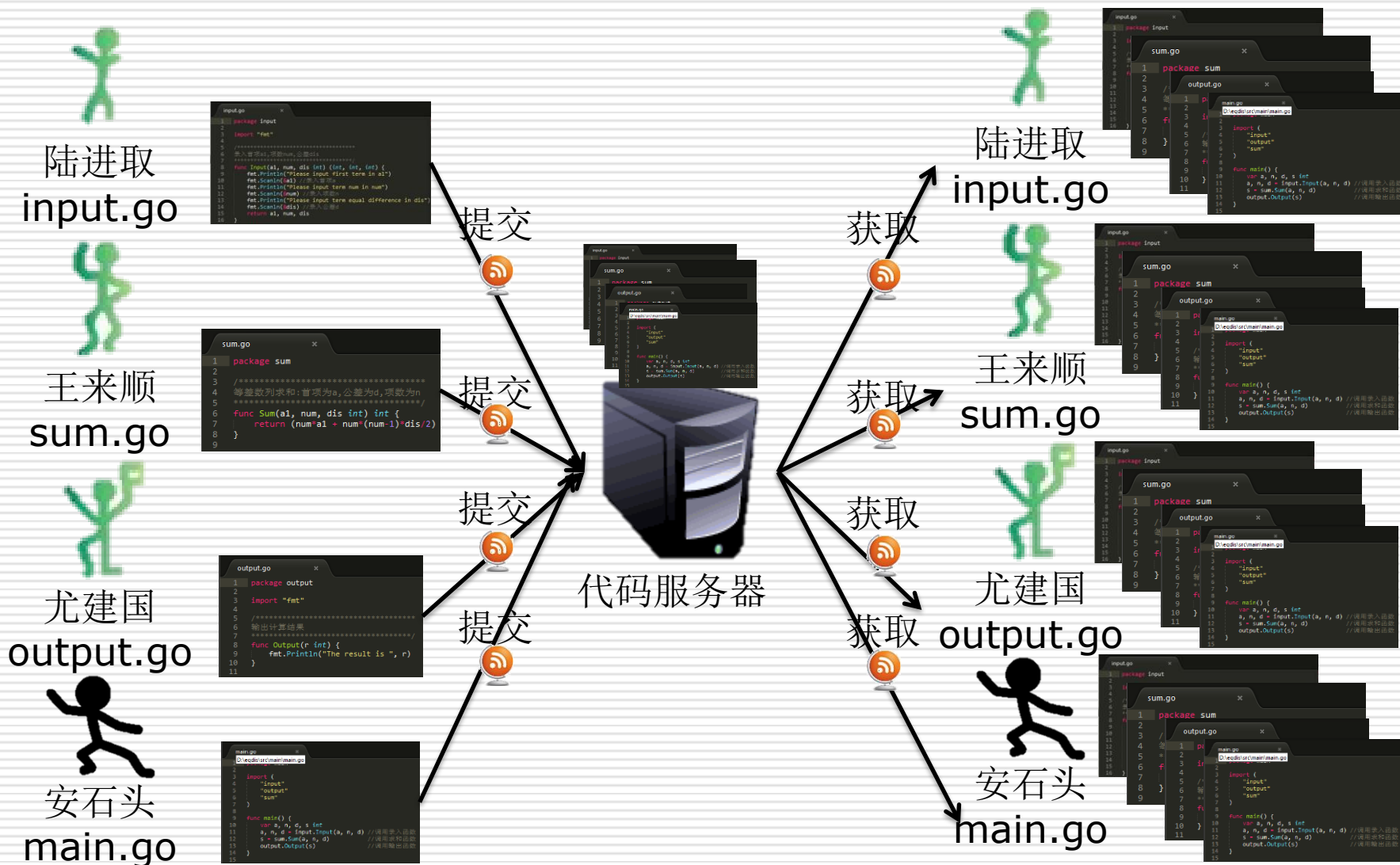


尤建国  
output.go



# 1.4如何实现真正的协同开发(续)

## □ 简单的协同开发的思想



# 1.4如何实现真正的协同开发(续)

---

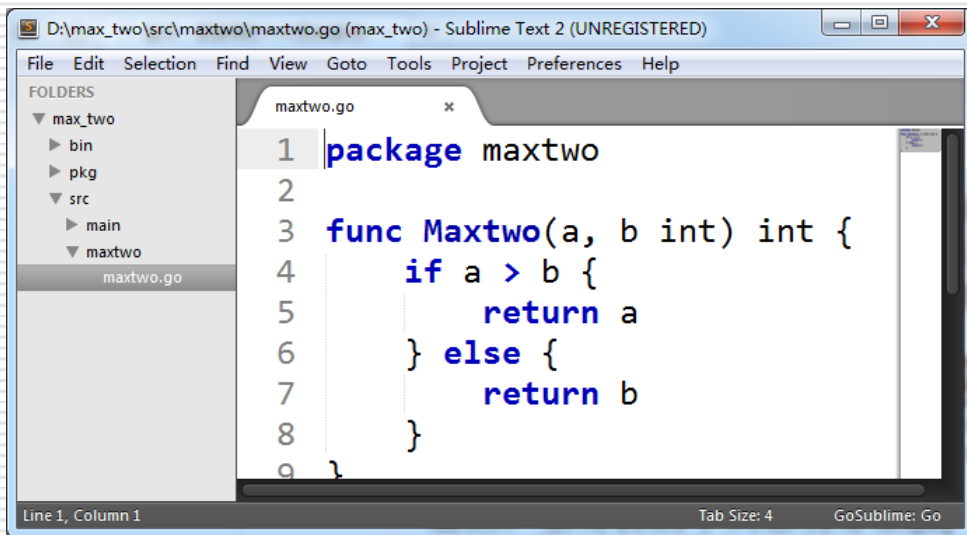
## □ 协同开发工具的推荐

- Source Control: CVS/SVN、git 和github
- Bug Tracking: Bugzilla, Trac, Roundup
- 交流工具: maillist, IM, Forum, IRC, Wiki
- 协同开发平台: sourceforge, BaseCamp



# 1.5如何实现功能复用

- ❑ 在做项目的过程中，经常出现用到之前开发的功能或代码的情况。如何实现功能的复用，避免重复开发，节省时间，提高效率呢？



```
1 package maxtwo
2
3 func Maxtwo(a, b int) int {
4     if a > b {
5         return a
6     } else {
7         return b
8     }
9 }
```

通过两次调用Maxtwo函数的方式实现

- ❑ 首先计算前两个数的最大
- ❑ 然后计算第三个数和前两个数最大的最大



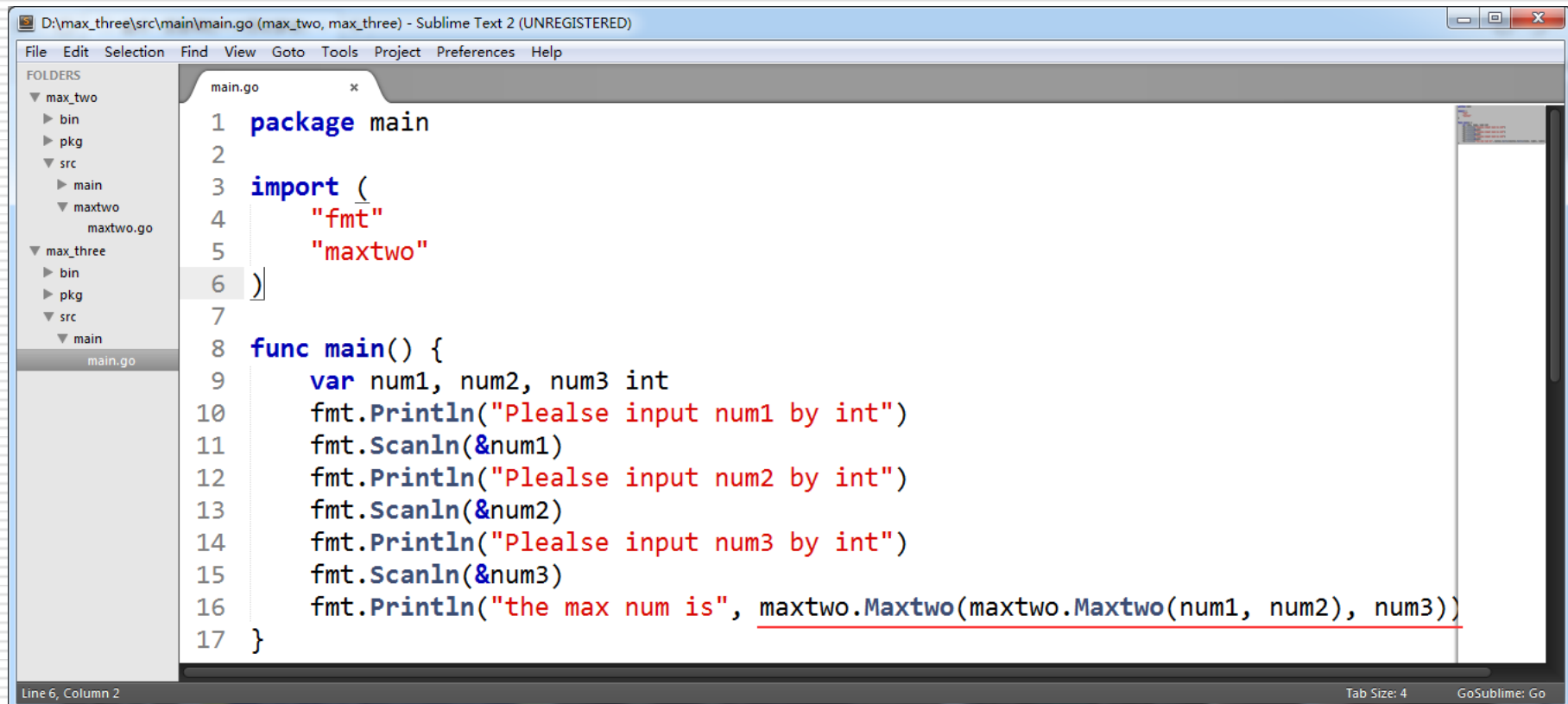
已完工项目：计算两个整数的最大值

待做项目：计算三个整数的最大值

# 1.5如何实现功能复用(续)

## ❑ 功能复用的实现

- 建立一个项目max\_three
- 建立main工作目录，在其中建立main.go
  - ❑ 在main.go中导入包：maxtwo
  - ❑ 对maxtwo.Maxtwo()方法两次调用计算出三个数的最大



The screenshot shows a Sublime Text 2 window titled "D:\max\_three\src\main\main.go (max\_two, max\_three) - Sublime Text 2 (UNREGISTERED)". The left sidebar displays a folder tree with the following structure:

- max\_two
  - bin
  - pkg
  - src
    - main
    - maxtwo
      - maxtwo.go
- max\_three
  - bin
  - pkg
  - src
    - main
      - main.go

The main editor window shows the following Go code in `main.go`:

```
1 package main
2
3 import (
4     "fmt"
5     "maxtwo"
6 )
7
8 func main() {
9     var num1, num2, num3 int
10    fmt.Println("Please input num1 by int")
11    fmt.Scanln(&num1)
12    fmt.Println("Please input num2 by int")
13    fmt.Scanln(&num2)
14    fmt.Println("Please input num3 by int")
15    fmt.Scanln(&num3)
16    fmt.Println("the max num is", maxtwo.Maxtwo(maxtwo.Maxtwo(num1, num2), num3))
17 }
```

The status bar at the bottom indicates "Line 6, Column 2", "Tab Size: 4", and "GoSublime: Go".

# 1.5如何实现功能复用(续)

## ■ 编译运行main.go

```
package main

import (
    "fmt"
    "maxtwo"
)

func main() {
    var num1, num2, num3 int
    fmt.Println("Plealse input num1 by int")
    fmt.Scanln(&num1)
    fmt.Println("Plealse input num2 by int")
    fmt.Scanln(&num2)
    fmt.Println("Plealse input num3 by int")
    fmt.Scanln(&num3)
    fmt.Println("the max num is", maxtwo.Maxtwo(maxtwo.Maxtwo(num1, num2), num3))
}
```

```
D:\max_three\src\main>go run main.go
Plealse input num1 by int
1
Plealse input num2 by int
3
Plealse input num3 by int
2
the max num is 3
```

# 1.5如何实现功能复用(续)

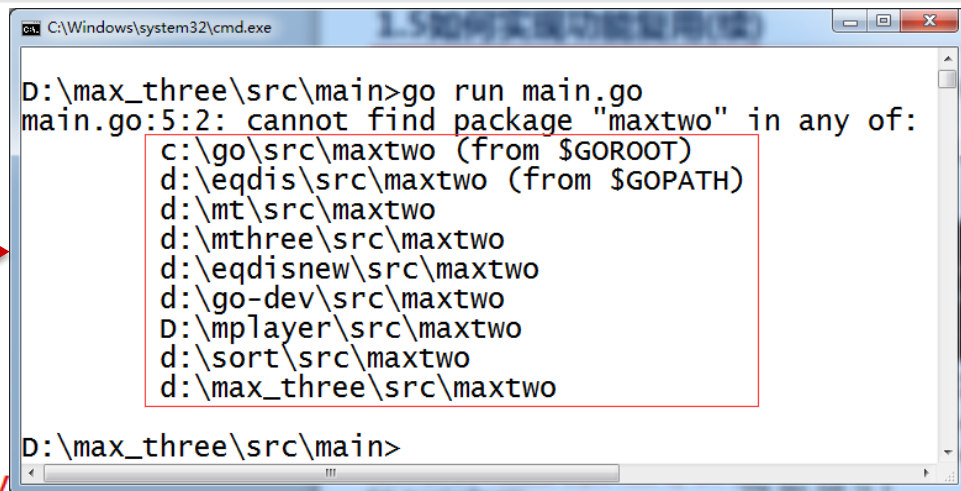
- 编译器会自动在GOROOT和GOPATH中找maxtwo包（先找.a,如果找不到找.go），都没有则报错
- 建议专门建立一个项目，用来存放自定义或第三方包

```
package main
```

```
import (  
    "fmt"  
    "maxtwo"  
)
```

如果在GOPATH、GOROOT中都没有maxtwo包则报错

```
func main() {  
    var num1, num2, num3 int  
    fmt.Println("Plealse input num1 by int")  
    fmt.Scanln(&num1)  
    fmt.Println("Plealse input num2 by int")  
    fmt.Scanln(&num2)  
    fmt.Println("Plealse input num3 by int")  
    fmt.Scanln(&num3)  
    fmt.Println("the max num is", maxtwo.Maxtwo(maxtwo.Maxtwo(num1, num2), num3))  
}
```



```
C:\Windows\system32\cmd.exe  
D:\max_three\src\main>go run main.go  
main.go:5:2: cannot find package "maxtwo" in any of:  
c:\go\src\maxtwo (from $GOROOT)  
d:\eqdis\src\maxtwo (from $GOPATH)  
d:\mt\src\maxtwo  
d:\mthree\src\maxtwo  
d:\eqdisnew\src\maxtwo  
d:\go-dev\src\maxtwo  
D:\mplayer\src\maxtwo  
d:\sort\src\maxtwo  
d:\max_three\src\maxtwo  
D:\max_three\src\main>
```

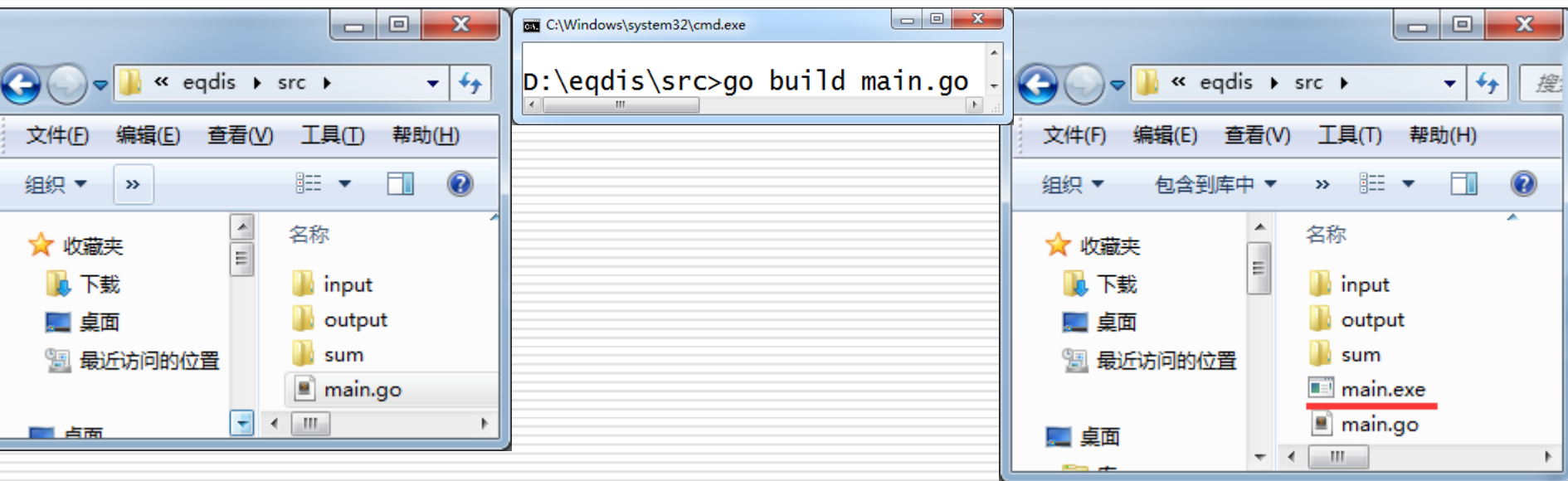
# 1.6总结

---

- ❑ 基于项目的多文件编程(掌握)
- ❑ 单元测试和性能测试(学会)
- ❑ 功能复用(学会)
- ❑ 协同开发的思想(理解)

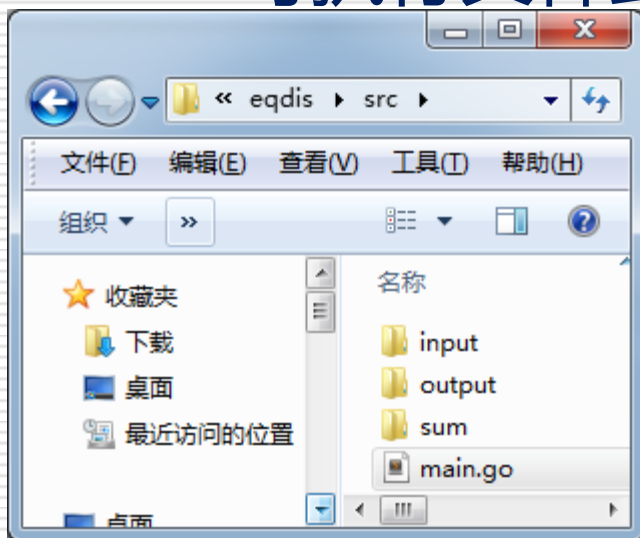
# 1.7思考

- ❑ 1.1中我们把main.go放在了src下的main目录，如果直接放在src下，可以用go build生成可执行文件吗？



# 1.7思考(续)

- ❑ 1.1中我们把main.go放在了src下的main目录，如果直接放在src下，可以用 go install 生成可执行文件到bin，依赖包文件到pkg吗？



```
C:\Windows\system32\cmd.exe

D:\eqdis\src>go install
go install: no install location for directory D:\eqdis\src outside GOPATH
D:\eqdis\src>go install main.go
go install: no install location for .go files listed on command line (GOBIN not set)
D:\eqdis\src>
```

**超出GOPATH管理范围，在src下建立个目录，将main.go放入其中即可**

**没有设置GOBIN,需要设置GOBIN变量，即指定可执行文件的生成位置才行**

## 1.7思考(续)

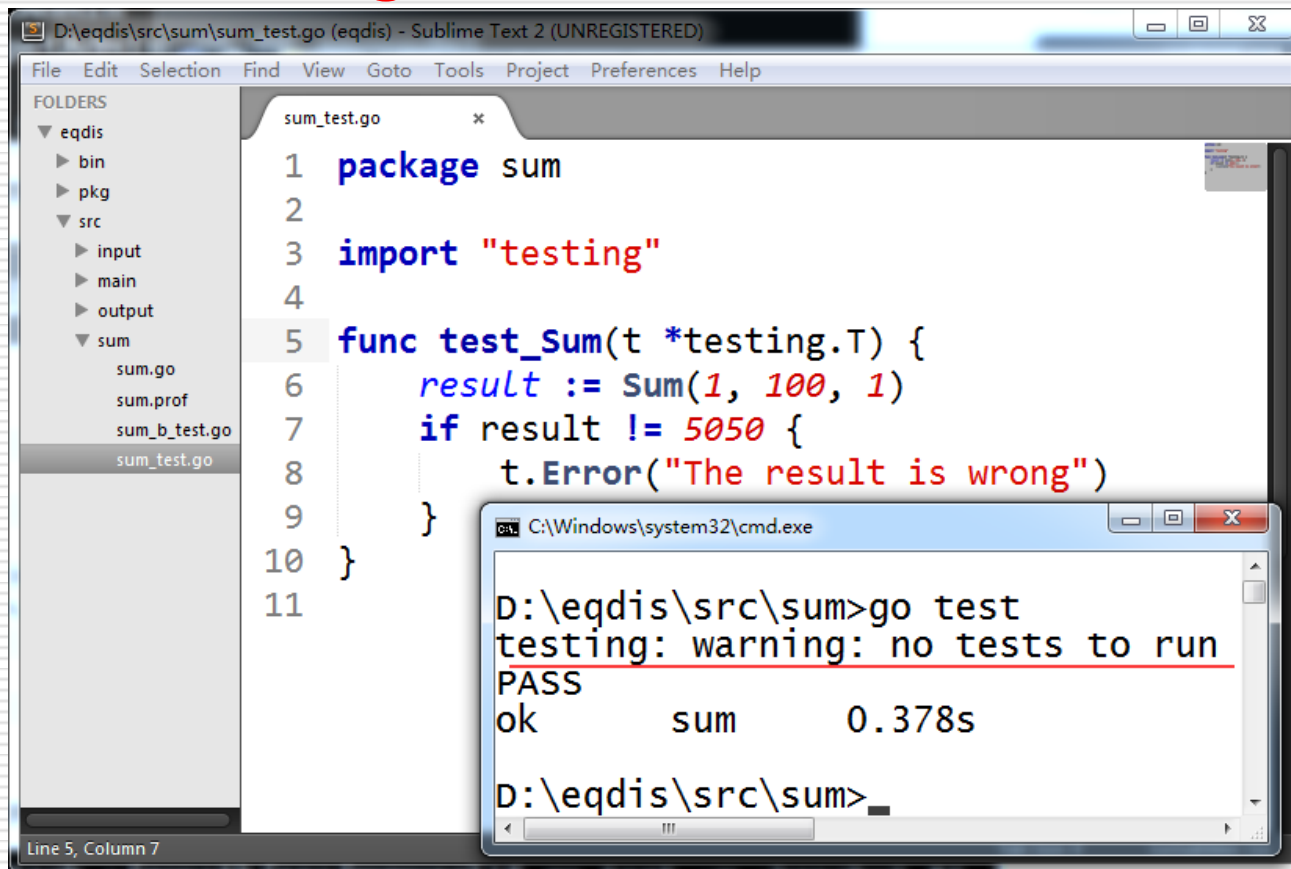
---

- 根据1.1请说出go run、go build及go install的区别
  - go build 编译包，如果是main包则在当前目录生成可执行文件，其他包不会生成.a文件；
  - go install 编译包，分别生成可执行文件和依赖包文件到%GOPATH%/bin，%GOPATH%/pkg
  - go run gofiles... 编译列出的文件，并生成可执行文件然后执行。注意只能用于main包，否则会出现go run: cannot run non-main package的错误。
  - go run是不需要设置GOPATH的，但go build和go install必须设置。
  - go run常用来测试一些功能，这些代码一般不包含在最终的项目中。



## 1.7思考(续)

❑ 执行图中所示测试代码，请分析出现 **testing: warning: no tests to run** 的原因？



The screenshot shows a Sublime Text editor window titled "D:\eqdis\src\sum\sum\_test.go (eqdis) - Sublime Text 2 (UNREGISTERED)". The editor displays the following Go code in `sum_test.go`:

```
1 package sum
2
3 import "testing"
4
5 func test_Sum(t *testing.T) {
6     result := Sum(1, 100, 1)
7     if result != 5050 {
8         t.Error("The result is wrong")
9     }
10 }
11
```

On the left, a "FOLDERS" pane shows the project structure, with `sum_test.go` selected under the `sum` folder. In the foreground, a Windows command prompt window titled "C:\Windows\system32\cmd.exe" shows the output of running `go test` in the `D:\eqdis\src\sum` directory:

```
D:\eqdis\src\sum>go test
testing: warning: no tests to run
PASS
ok      sum      0.378s

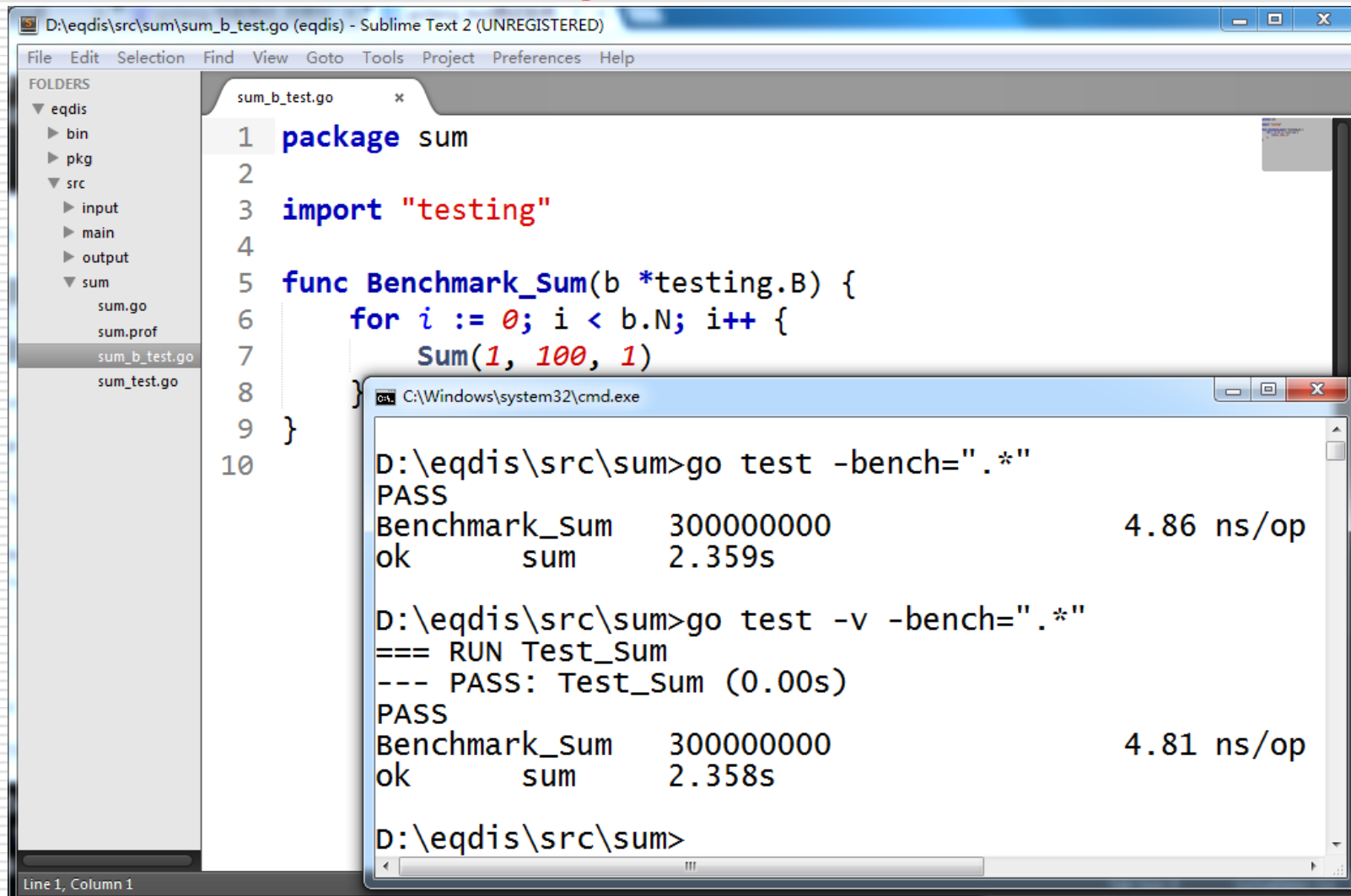
D:\eqdis\src\sum>
```

The status bar at the bottom left of the editor indicates "Line 5, Column 7".

❑ 是 **test\_Sum** 的首字母小写的原因导致，应为大写 **T**

# 1.7思考(续)

□ 执行图中所示的测试代码，请分析 `go test -v -bench= ".*"` 与 `go test -bench= ".*"` 区别



The screenshot shows a Sublime Text editor window titled "D:\eqdis\src\sum\sum\_b\_test.go (eqdis) - Sublime Text 2 (UNREGISTERED)". The editor displays the following Go code in `sum_b_test.go`:

```
1 package sum
2
3 import "testing"
4
5 func Benchmark_Sum(b *testing.B) {
6     for i := 0; i < b.N; i++ {
7         Sum(1, 100, 1)
8     }
9 }
10
```

Overlaid on the bottom right is a Windows command prompt window titled "C:\Windows\system32\cmd.exe". It shows the execution of two `go test` commands:

```
D:\eqdis\src\sum>go test -bench=".*"
PASS
Benchmark_Sum      3000000000          4.86 ns/op
ok      sum      2.359s

D:\eqdis\src\sum>go test -v -bench=".*"
=== RUN Test_Sum
--- PASS: Test_Sum (0.00s)
PASS
Benchmark_Sum      3000000000          4.81 ns/op
ok      sum      2.358s

D:\eqdis\src\sum>
```

The command prompt output demonstrates that `go test -bench=".*"` only runs benchmarks, while `go test -v -bench=".*"` runs all tests, including the `Test_Sum` unit test.

## 1.7思考(续)

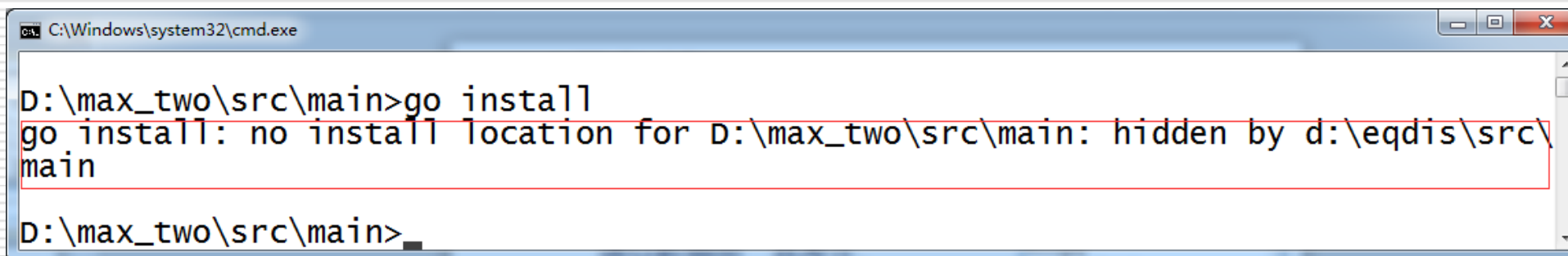
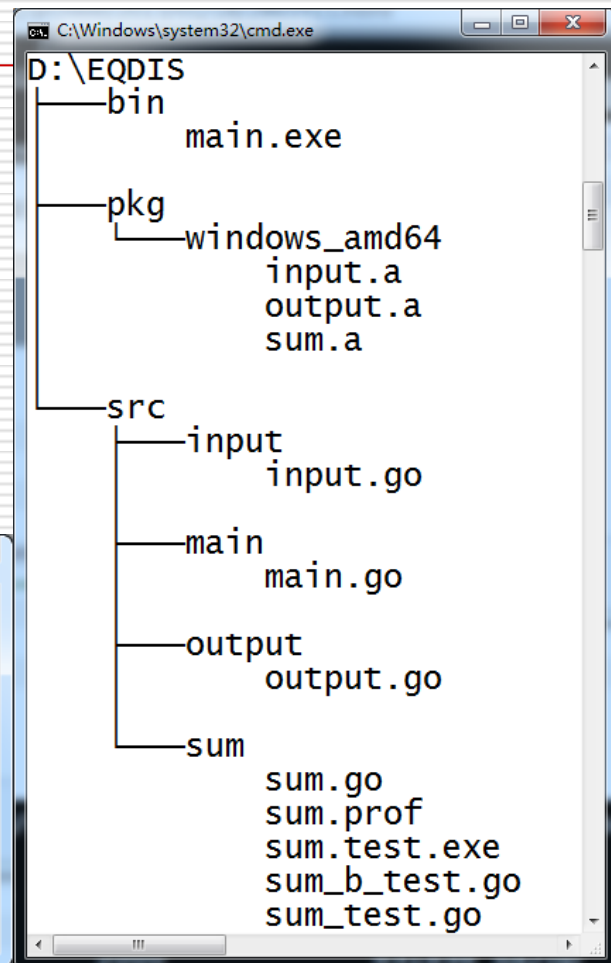
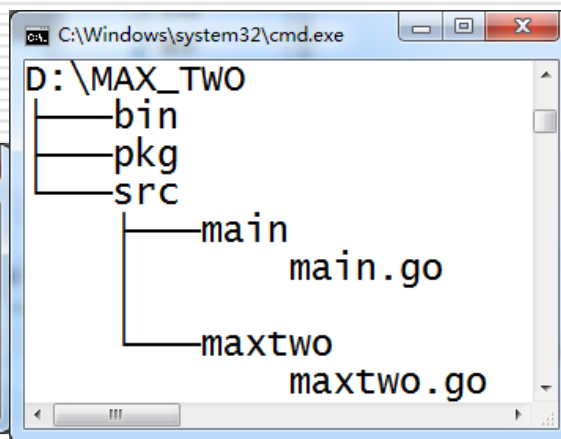
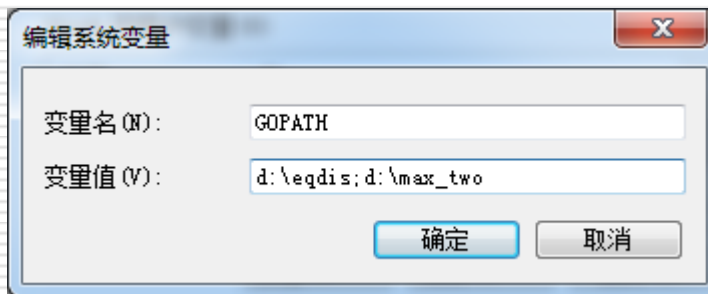
- 请完成图中所示源代码的单元测试和性能测试，并能以文本及图形方式查看CPU性能。

```
1  package split
2
3  func Split(sum int) (x, y int) {
4      x = sum * 4 / 9
5      y = sum - x
6      return x, y
7  }
```

# 1.7思考(续)

## □ 当前已有项目及结构如下

- 在对项目max\_two执行go install命令时，出现了如图所示错误，请分析原因，并改正。



# 1.7思考(续)

---

## □ 自定义包练习

- 建立一个项目go\_mypkg,然后在其src下建立sum和maxtwo工作目录，并将1.1中的sum.go 和1.5中的maxtwo.go分别放进去，并用go install 命令编译成.a文件到pkg中。
- 建立两个项目，利用上述自定义包，分别实现等差数列求和及计算三个数的最大。

---

**Thank you very much**

*Any comments and suggestions  
are beyond welcome*