变量变量变?

N大基本类型

类型
int, int8, int16, int32, int64
uint, uint8, uint16, uint32, uint64
float32, float64
complex64, complex128
bool
byte, rune
string
error

具体解释见代码vars.go

声明

四大天王

类型	对应
`var`	变量
`const`	常量
`type`	类型
`func`	函数实体对象

变量声明

```
var a = 1
const b = 2
var c int = a*a-b
d := a+b
var e int // 0
```

```
var 变量名字 类型 = 表达式
```

"类型" 或 "= 表达式"可以省略其中的一个
省略"类型"，根据值自动推断
省略"= 表达式"，缺省为"零值"（例如，int = 0, string = ""）

:= （简短变量声明）go特色美食之一，yummy!😋

"全局变量" "局部变量"

```
package main

import "fmt"

const a = 1 // global

func main() {
    var b = 2 // local
    fmt.Println(a, b)
    pri()
}

func pri() {
    fmt.Println(a)
    // fmt.Println(b) × undefined: b
}
```

函数示例

```go
package main

import "fmt"

func main() {
    var a int = 1
    fmt.Println(square(a))
}

func square(a int) int {
    return a * a
}
```

#$高级&*声明

```java
int a = 1, b = 2, c = 3;
```

🤔

```go
var a, b, c = 1, 2, 3
```

😋

```
var a, b, c int = 1, 2, 3 // 都是int型
// var a, b, c int, float32, string × 不准乱搞😇
var b, f, s = true, 2.3, "four" // bool, float64, string

i, j := 0, 1
i, j = j, i // swap(i, j)!
// 尊嘟假嘟？来逝逝吧！

var (
    i int
    j float32
    k string
)

var (
    i = 1
    j = 2.3
    k = "4"
)

// := 暗度陈仓
a, b := 1, 2 // 创建两个新变量

var c int = 1
c, d := 2, 3 // 也是可以的!
// c-重新赋值  d-声明新变量
// 有新的就行!
c, d := 4, 5 // × no new variables on left side of :=
// 但凡上点贡都算了，结果你小子想空手套白狼？

var e int = 1
e, f = "1", "2" // ×
// 赋值还赋出花来了？
```

multi returns?

这搁C++想都不敢想😁

e.g.

```go
func main() {
    var a = 1
    _a, __a := OneAndTwo(a)
    println(_a, __a)
}

func OneAndTwo(a int) (int, int) {
    return a+1, a+2
}
```

more exs in multiReturn.go

本地包导入

具体示例详见

`vars.go / func pkgImport()`

操作示例

在现在的 `ch2` 文件夹下，再新建一个文件夹 `pkg` ，里面新建一个文件 `utils.go`

```go
package pkg
```

起手表示包名为 `pkg`

在 `ch2` 里的 `vars.go` 则可以

```go
import pkg "example.com/go-demo/ch2/pkg"
```

导入 `example.com/go-demo/ch2/pkg` 包并取个小名 `pkg`

为什么是 `example.com/go-demo/ch2` ？ 看看go.mod文件😋

Public or private?

代码里有对包中公有/私有内容的介绍，其中

1. 大写字母开头 => 公有（其他包可见）
2. 小写字母开头 => 私有（其他包不可见）

啧，乱起名字要死人力😇

作用域

复习：C++作用域？（死去的程设开始攻击我~）
可见性/可访问范围

块作用域、函数作用域、全局作用域、类作用域、命名空间作用域...

e.g.

```
int x = 10; // 全局变量

int main() {
    int x = 20; // 局部变量，遮蔽了全局x

    {
        int x = 30; // 新的局部变量
        cout << x; // 输出30
    }

    cout << x; // 输出20
    cout << ::x; // 输出10(访问全局x)
}
```

听不懂思密达😋
C++起开，让Go学长来😘

包作用域

整个包内可见（某个包的"全局变量"）

```go
package main

var globalVar int // 包级作用域

func main() {
    println(globalVar) // 可访问
}

func test() {
    println(globalVar) // 可访问
}
```

如果两个.go文件都是 `package main` (a.go, b.go)那么b.go可以访问a.go里面的 `globalVar` 吗?
试试就知道啦!😘

函数级作用域

函数内声明，仅函数内有效
包括参数和局部变量

```go
func foo() {
    x := 10 // 仅在 foo 函数内有效
}

func bar() {
    // println(x) // 错误：x 未定义
}
```

块级作用域

`{}` 包围的代码块（e.g. `if`， `for`， `switch` 等）

```
func main() {
    x := 1
    {
        x := 2 // 新变量，遮蔽外层的 x
        fmt.Println(x) // 输出 2
    }
    fmt.Println(x) // 输出 1
}
```

文件作用域

仅限当前文件，通过 `import . "package"` 实现

花活导入之——点导入！

```
import . "fmt" // 可直接使用 Println 而不需要 fmt. 前缀

func main() {
    Println("Hello") // 无需写 fmt.Println
}
```

Others...

if块

```
if x := 10; x > 5 { // x 仅在 if 块内有效
    fmt.Println(x)
}
// fmt.Println(x) // 错误：x 未定义
```

for块

```
for i := 0; i < 3; i++ { // i 仅在 for 块内有效
    fmt.Println(i)
}
// fmt.Println(i) // 错误：i 未定义
```

作用域链与变量遮蔽

从内向外

```
var x = 1 // 包级变量

func main() {
    x := 2 // 局部变量遮蔽包级变量
    {
        x := 3 // 遮蔽上一层的 x
        fmt.Println(x) // 输出 3
    }
    fmt.Println(x) // 输出 2
    // Go没有C++的::x!（悲）
}
```