

Golang学习分享

——语法特点

刘翔宇

要点

- 变量的声明：对比C和Go，为什么要这么写
- 数组和指针：对比C和Go、Array和Slice
- 多值返回：使用场景、返回值命名
- If语句：语法、使用场景、陷阱题
- Defer：使用场景
- 运算符优先级：语法、陷阱题
- “_”符号：使用场景1、使用场景2
- TODOs：还有哪些要学习的

变量的声明

	Variable	Array	Function
C	int x	int arr[10]	int add(int x, int y) { ... }
Go	x int	arr [10]int	func add(x int, y int) int { ... }

`“void (*signal(int, void (*)(int)))(int)”`

– 《C陷阱与缺陷》

The "Clockwise/Spiral Rule"

```
void (*signal(int, void (*fp)(int)))(int);
```

signal is a function
passing a int and a pointer to function passing
a int returning nothing
returning a pointer to a function passing int
returning nothing

C: `void (*signal(int, void (*)(int)))(int)`

改写成Go就变成了:

Go: `signal func(int, *func(int)) *func(int)`

数组和指针

- Arrays are values. Assigning one array to another copies all the elements.
- In particular, if you pass an array to a function, it will receive a copy of the array, not a pointer to it.
- The size of an array is part of its type. The types `[10]int` and `[20]int` are distinct.

归结为一句话就是：

在 C 语言里，数组类似指针；在 Go 语言里，数组是值。

```
main() {  
    arr := [...]int{2, 5, 3, 4, 1}  
    sort(arr)  
    fmt.Println(arr) // arr = ?  
}  
  
func sort(arr [5]int) {  
    .....  
}
```



```
func sort(arr [5]int)
```

改写成

```
func sort(arr *[5]int) //数组指针  
func sort(arr []int)   //Slice
```

切片 Slice 对数组进行封装，保存了对底层数组的引用

多值返回

```
func swap(x, y string) (string, string) {  
    return y, x  
}
```

场景1

```
tmp=a[j];  
a[j]=a[j+1];  
a[j+1]=tmp;
```

$$a[j], a[j+1] = a[j+1], a[j]$$

场景2

```
func Write(b []byte) (n int, err error)
```

返回值命名

```
func nextInt(b []byte, pos int)  
(value, nextPos int) {
```

If 语句

```
func pow(x, n, lim float64) float64 {  
    if v := math.Pow(x, n); v < lim {  
        return v  
    }  
    //v的作用域只有if和else  
    return lim  
}
```

使用场景

- 这个特性简直就是纯粹的实用主义体现，它使得我们可以很方面地只使用一个 `err` 值，例如，在一个相当长的 `if-else` 语句链中， 你会发现它用得很频繁。

```
var a int = 1
if a := false; a {
    fmt.Println(a)
} else {
    fmt.Println(a)
}
fmt.Println(a)
```


Defer

- 推迟执行函数，该函数会在执行 defer 的函数返回之前立即执行。
- 例如无论以何种路径返回，都必须释放资源的函数。典型的例子就是解锁互斥和关闭文件。

Java:

```
InputStream is = null;
try {
    is = new InputStream(...)
    .....
} catch (Exception e) {
    throw e;
} finally {
    if (is != null) {
        is.close();
    }
}
```

Go:

```
InputStream is = NewInputStream(...)
if is != nil {
    defer is.Close()
}
.....
```

Defer

- 两个好处:
- 不会忘了关闭资源（有时候会在中间加return语句）
- 代码可读性好

运算符优先级

Precedence	Operator
5	* / % << >> & &^
4	+ - ^
3	== != < <= > >=
2	&&
1	

```
r = low << 4 + hi  
r = low << 4 | hi
```

	$1 + 1 \mid 1 + 1$	$1 \ll 1 + 1 \ll 2$
C	$(1 + 1) \mid (1 + 1) = 2$	$1 \ll (1 + 1) \ll 2 = 16$
Go	$1 + 1 \mid 1 + 1 = 4$	$(1 \ll 1) + (1 \ll 2) = 6$

Format会帮我们做一些事

`1 + 1 | 1 + 1 to 1 + 1 | 1 + 1`

`1 << 1 + 1 << 2 to 1 << 1 + 1 << 2`

“_”符号

- 声明过的变量必须要使用
- 引用的包必须要使用
- 开发过程中就会有麻烦

```
import (  
    "fmt"  
    _ "log"  
)  
func main() {  
    var _ = foo()  
  
    var tmp = goo()  
    _ = tmp  
}
```


TODOs

- 不像 Java 有个 Maven
- IDE 上暂时还只会用 Eclipse
- 官网上的其他文章
- 常用的package的用法