Part I: 语法特点

不一样的声明语法

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

为什么要这么声明呢？先看一个例子，《C 陷阱与缺陷》中有这样一个函数声明：

*void (*signal(int,void(*)(int)))(int)*
*What the f*ck…?*

翻译成英文是：

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

翻译成中文是：

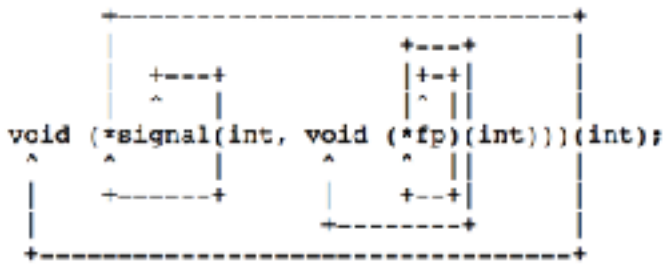
signal 是一个函数，这个函数有两个参数 *int* 和 *fp1*，返回 *fp2*
fp1 是一个函数指针，指向一个参数为 *int* 的函数
fp2 是一个函数指针，指向一个参数是 *int* 的函数

如果用 Go 来表示呢?

*signal func(int, *func(int)) *func(int)*

题外话1：如何看懂 C 语言中复杂的声明语句

The "Clockwise/Spiral Rule":
1 Starting with the unknown element, move in a spiral/clockwise direction; when ecountering the following elements replace them with the corresponding english statements:
[X] or []
=> Array X size of... or Array undefined size of...
(type1, type2)
=> function passing type1 and type2 returning...
***
=> pointer(s) to...



2 Keep doing this in a spiral/clockwise direction until all tokens have been covered.
3 Always resolve anything in parenthesis first!

题外话2：Swift 中的函数声明

fun add(a: Int, b: Int) -> Int { ... }
是不是和 *Golang* 很像?

数组和指针

3个不同于C的地方，在C里面，数组类似于指针

```
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.
```

// 一般不会把数组作为函数参数，因为复制所有元素会消耗性能。

下面这题能猜出答案么?

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

func sort(arr [5]int) {
    ……
}
```

怎么改呢?

```
func sort(arr *[5]int) //数组指针
func sort(arr []int)   //Slice，更多是用这种写法
// 切片 Slice 实际上是对数组进行封装，保存了对底层数组的引用
```

顺带一提结构体

写惯Java的写Go，可能会犯如下的错误

```
Go:
type Vertex struct {
    X, Y int
}

func rotate(v Vertex) {
    v.X, v.Y = v.Y, v.X
}
v := Vertex{1, 2}
rotate(v)
fmt.Println(v) //{1, 2}
// 直接把结构体作为参数传递，实际上是把结构体里的元素复制了一遍
```

改成下面这样就可以实现函数的目的

```
func rotate(v *Vertex) {
    v.X, v.Y = v.Y, v.X
}
v := Vertex{1, 2}
rotate(&v)
fmt.Println(v) //{2, 1}
```

多值返回

先看看神奇的语法

Go:

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

记得最早写排序的程序时，经常出现类似下面这样的代码

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

如果用 Go 来写，就只需要

```
a[j], a[j+1] = a[j+1], a[j] // 既简洁又直观，对吧
```

另外一个常用的地方是异常的处理

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

再也不用纠结下面这种情况，到底是 return null，还是抛异常了

```
func getStudent(id int) (s Student, err error) {
    …
}
```

从上面的例子可以看出，返回值还可以命名。对于只有一个返回值的函数影响不大，因为大多数函数光看函数名就能猜出来返回值是什么意思，对于多个返回值的函数就有区别了

```
fun Read(is InputStream) (string, string) { ... } // 非要用 string 来表示 error
fun Read(is InputStream) (string content, string err) { ... }
```

流程处理

if 中可以有表达式，好处在于减小作用域，在处理 err 时很实用，结合":="，只需要一个 err 就行

```
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
}
```

于是想出一道挖坑题

```
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。

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

为了 finally 而加 try catch

```
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()
}
...
```

switch case不用break，default可以放第一行

type switch，.(type)只能用在switch中，且只能用在interface

面向对象

方法

关于方法的指针还是值，有这么个规则
The rule about pointers vs. values for receivers is that value methods can be invoked on pointers and values, but pointer methods can only be invoked on pointers.
When the value is addressable, the language takes care of the common case of invoking a pointer method on a value by inserting the address operator automatically

接口

不像 *Java* 有 *implements* 关键字，*Go* 中只要某个类型实现了某个 *interface* 所有的方法，就表示实现了这个接口

Others

自动添加 “;”，除了if、for语句中的”;”，其他地方都不需要写 ";"

习惯了 C 语言这么写的同学就需要时间来适应

```
if i < f() //Error
{          //Error
g()
}
```

字面量构造struct、slice、array、map

简化构造假数据的代码

首字母大写表示被导出

运算优先级

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

所以

```
1 + 1 | 1 + 1 = ?
1<<1 + 1<<2 = ?
// both in Go and C
```

The blank identifier

```
import (
    “fmt”
    _ “log”
)
func main() {
    var _ = 1
    //or
    var tmp = 2
    _ = tmp
}
```

Formatting

```
type T struct  {
    name string // blablabla
    value int // blablabla
}
```

=>

```
type T struct {
    name  string // blablabla
    value int    // blablabla
}
```

Part II: 怎么写代码

完整的项目结构

```
bin/
    hello                          # command executable
    outyet                         # command executable
pkg/
    linux_amd64/
        github.com/golang/example/
            stringutil.a           # package object
src/
    github.com/golang/example/
        .git/                      # Git repository metadata
          hello/
              hello.go               # command source
          outyet/
              main.go                # command source
              main_test.go           # test source
          stringutil/
              reverse.go             # package source
              reverse_test.go        # test source
    golang.org/x/image/
        .git/                      # Git repository metadata
          bmp/
              reader.go              # package source
              writer.go              # package source
    ... (many more repositories and packages omitted) ...
```

IDE 搭建之 Windows

习惯 *Eclipse* 的，直接用 *Eclipse Neon* 就行

IDE 搭建之 Linux(CentOS 6.5)

```
wget https://storage.googleapis.com/golang/go1.7.5.linux-amd64.tar.gz //下载
tar zxvf go1.6.3.linux-amd64.tar.gz -C /usr/local/ //解压
export PATH=$PATH:/usr/local/go/bin //设置环境变量

//这时候可以使用go命令了，bin目录下有三个工具：go、godoc、gofmt
//上一步也可以这么设置：
//export GOROOT=$HOME/go
//export PATH=$PATH:$GOROOT/bin

mkdir ~/gopath
export GOPATH=~/gopath

go install github.com/user/hello
$GOPATH/bin/hello

cd $GOPATH/src
git clone https://github.com/golang/example.git github.com/golang/example

vim-go：

mkdir -p .vim/bundle
```

```
git clone https://github.com/fatih/vim-go.git ~/.vim/bundle/Vundle.vim

cd ~/.vim/bundle
git clone https://github.com/Valloric/YouCompleteMe.git
```

Part III: 优缺点

不像 Java 有个 Maven

Part IV: 参考资料

https://blog.golang.org/gos-declaration-syntax（Go's Declaration Syntax）

http://c-faq.com/decl/spiral.anderson.html（The "Clockwise/Spiral Rule"）

https://golang.org/ref/spec（Language Specification）

https://golang.org/doc/effective_go.html（Effective Go）

https://golang.org/doc/code.html（How to Write Go Code）