打实基础

go圣经一共分为14个章节。其中前八章为基础知识。感觉后边的几章属于有点难度的。这仅仅对于我这种小菜鸟来说。就目前我读的前三章来说，主要为程序结构和基础数据类型这俩部分。

简单说一下这两章的重点：其实在书中就简单的介绍了go的简单语法。比如：命名的规则，声明的方法，变量的使用，赋值的操作，类型的声明，包和文件，和作用域。剩下的就是数据类型。

命名规范：一个名字必须以一个字母（Unicode字母）或下划线开头，后面可以跟任意数量的字母、数字或下划线。说实话哈，我现在用到的都是小驼峰的方式来命名。有时候还是大小写的写法没有抓到重点。书中给出了很多关键字，我在这也复制下来吧，方便自己记忆。

> > 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

除了这些还有预定的名字，
内建常量: true false iota nil

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

内建函数: make len cap new append copy close delete complex real imag panic recover

go中主要有四种声明语句，var,const,type,func,这几个分别对应的是变量，常量，类型和函数实体。变量的几种声明方式。

```
var 变量名字 类型 = 表达式
```

```
var i, j, k int                 // int, int, int
var b, f, s = true, 2.3, "four" // bool, float64, string
```

简短的声明方式：

```
i, j := 0, 1
```

有时候go的一些小点和python有些相似，这大概是语言的通用性？咱也不敢确定啊。

指针的用法让我感觉有点小闹心，接触过一点点的c，但是没学明白，到现在导致指针有点懵逼。不过书里简单的介绍了点。加上自己在撸代码的时候用到的，感觉来感觉了。

简单的说：一个指针的值是另一个变量的地址。一个指针对应变量在内存中的存储位置。

> 如果用“`var x int`”声明语句声明一个x变量，那么`&x`表达式（取x变量的内存地址）将产生一个指向该整数变量的指针，指针对应的数据类型是`*int`，指针被称之为“指向`int`类型的指针”。如果指针名字为`p`，那么可以说“`p`指针指向变量`x`”，或者说“`p`指针保存了x变量的内存地址”。同时`*p`表达式对应p指针指向的变量的值。一般`*p`表达式读取指针指向的变量的值，这里为`int`类型的值，同时因为`*p`对应一个变量，所以该表达式也可以出现在赋值语句的左边，表示更新指针所指向的变量的值。

```
x := 1
p := &x         // p, of type *int, points to x
fmt.Println(*p) // "1"
*p = 2          // equivalent to x = 2
fmt.Println(x)  // "2"
```

还有`new`方法的使用：表达式`new(T)`将创建一个T类型的匿名变量，初始化为T类型的零值，然后返回变量地址，返回的指针类型为`*T`。

```
p := new(int)   // p, *int 类型, 指向匿名的 int 变量
fmt.Println(*p) // "0"
*p = 2          // 设置 int 匿名变量的值为 2
fmt.Println(*p) // "2"
```

赋值操作：最简单的赋值语句是将要被赋值的变量放在=的左边，新值的表达式放在=的右边。

```
x = 1                         // 命名变量的赋值
*p = true                     // 通过指针间接赋值
person.name = "bob"           // 结构体字段赋值
count[x] = count[x] * scale // 数组、slice或map的元素赋值
```

还有一些元组赋值感觉和 `python` 差不多。不过在写斐波那契数列的时候go和python不分上下，就单从代码简单性来讲。

```
func fib(n int) int {
    x, y := 0, 1
    for i := 0; i < n; i++ {
        x, y = y, x+y
    }
    return x
}
```

重要的说说作用域吧，感觉好些地方出到了这个地方。

> 声明语句对应的词法域决定了作用域范围的大小。对于内置的类型、函数和常量，比如int、len和true等是在全局作用域的，因此可以在整个程序中直接使用。任何在在函数外部（也就是包级语法域）声明的名字可以在同一个包的任何源文件中访问的。对于导入的包，例如tempconv导入的fmt包，则是对应源文件级的作用域，因此只能在当前的文件中访问导入的fmt包，当前包的其它源文件无法访问在当前源文件导入的包。还有许多声明语句，比如tempconv.CToF函数中的变量c，则是局部作用域的，它只能在函数内部（甚至只能是局部的某些部分）访问。

最难受的是每个for、if和switch语句的语法决；每个switch或select的分支也有独立的语法决；当然也包括显式书写的语法块（花括弧包含的语句）。 所以在写循环的时候，咦，突然发现自己刚才的变量用不了了。

Go语言将数据类型分为四类：基础类型、复合类型、引用类型和接口类型。但是基础类型就没啥说的，无非是整数，浮点，复数，布尔，字符串，常量等。就简单的介绍一下。

Go语言同时提供了有符号和无符号类型的整数运算。这里有int8、int16、int32和int64四种截然不同大小的有符号整形数类型，分别对应8、16、32、64bit大小的有符号整形数，与此对应的是uint8、uint16、uint32和uint64四种无符号整形数类型。%g来打印参数。

值得去记得的是Unicode字符rune类型是和int32等价的类型，通常用于表示一个Unicode码点。这两个名称可以互换使用。同样byte也是uint8类型的等价类型，byte类型一般用于强调数值是一个原始的数据而不是一个小的整数。一般情况下不知道额外意思就懵。

其中整数中的运算和我们常用的差不多，唯一有的区别是算术上，一个`x<<n`左移运算等价于乘以`2^n2`，一个`x>>n`右移运算等价于除以`2^n2`。 Go中定义了两种浮点型，一个是`float32`,`float64`。

> 一个`float32`类型的浮点数可以提供大约6个十进制数的精度，而`float64`则可以提供约15个十进制数的精度；通常应该优先使用`float64`类型，因为`float32`类型的累计计算误差很容易扩散，并且`float32`能精确表示的正整数并不是很大.

浮点型利用`%d`，`%o`，`%x`参数控制输出格式。