程序结构

名称

- 变量名、函数名以及文件名的命名方式与C语言相同: 以字母或下划线开头，后面可跟任意数量的字符、数字以及下划线
  - 如: _name, _Name, studentName_
  - Go官方采用驼峰命名，而非下划线
  - 注: json数据传输格式通常采用下划线命名
- Go没有诸如Java中的public以及private关键字，而是采用首字母大小写决定其是否可以跨包(Go包与Python中的package含义相同)
  - 函数名、变量名首字母大写: 对包外可见 — 如 GetConfig
  - 函数名、变量名首字母小写: 对包外可见 — 如 getConfig

变量

- 声明: var name type = expression
  - 例如: var s string = "smart"
  - Go有内置的类型推断，故type有时可以省略，如:
    var s = "smart"
- 初始化: 若表达式省略，则Go会主动的将变量初始化为对应类型的零值
  - int/float/double: 初始值为0
  - bool: 初始值为false
  - string: 初始值为""
  - 接口以及引用类型(slice,指针,通道，函数): 初始值为nil
  - 数组与结构体: 初始值为其所有元素或成员的零值

```
type User struct {
    Name string
    Age uint8
}
```

```
User{"", 0}
```

- 短变量声明(仅能声明并初始化局部变量)

```
// 定义单个变量
s := "smart"

// 定义多个变量，并接收
f, err := os.Open(name)
```

  - 短变量声明最少声明一个新的变量

```
f, err := os.Open(name)
// ...
f, err := os.Open(otherNmae)  // 编译无法通过
```

- 指针: 所有声明的变量均有指针，其值指向变量的内存地址

```
s := "smart"
ptr := &s
fmt.Println(*ptr)
```

  - *ptr其实就是变量s的别名
  - Go语言中的指针不能进行运算，所以其复杂度要比C低许多
  - **空指针声明: var ptr *int，此时ptr就是没人要的"野指针"，即ptr不指向任何位置**
  - **C语言喜欢将变量置于函数起始位置，但Go的设计哲学是用的时候再声明和初始化**
- new函数
  - 表达式new(T)创建一个未命名的T类型变量，初始化为T类型的零值，并返回其地址(*T)

```
p := new(string)
fmt.Println(*p) // 输出""
```

  - new函数并没有什么特殊的地方，只是语法上的便利而已

```
func newUser() *User {
    var s User
    return &s  // 此时变量s将在堆上分配内存
}
```

- 类型声明
  - type声明定义一个新的命名类型

```
type timestamp int
```

  - 可以用这种方法来拓展Go内置数据类型的行为(可以认为是继承的一种实现)

作用域

- Go变量作用域为每一个语句块中
- 短变量声明依赖一个明确的作用域

```
var s string

func init() {
    // 并不会更新包级别的变量s
    // 而是在init函数作用域中重新声明了一个变量
    s := "smart"
}
```