Go语言编程

Go语言概述

- 诞生：2009年Google开发
- 设计者：Robert Griesemer/Rob Pike/Ken Thompson
- 语言定位：系统级/云计算时代语言

核心设计理念

- **设计哲学："Less is more"**
- **目标：**
  - 编译速度
  - 开发效率
  - 执行性能
  - 并发支持
-

语言特性

- **Go特性：**
  - 编译　　C, C++
  - 静态类型　　C, C++, Java
  - 垃圾回收　　Java and Python
  - 并发　　Erlang
- **工具链：**
  - go build/test/fmt/doc...

Erlang给分布式系统设计提供了很多有益的启发，参考阅读：

Making reliable distributed systems in the presence of software errors

语言特性

```go
// 经典Hello World
package main
import "fmt"
func main() {
    fmt.Println("Hello, 世界")
}
```

变量

- **静态类型Statically typed:**
  - var x int
  - var s, t string
- **显示/隐式地初始化:**
  - var x int
  - var s, t string = "foo", "bar" // multiple assignment
  - var x = 42 // int
  - var s, b = "foo", true // string, bool
- **短声明，函数内部:**
  - x := 42
  - s, b := "foo", true

类型

- **基础类型**
  - uint8 (byte), uint16, uint32, uint32, uint64,
  - int8, int16, int32, int32 (rune), int64,
  - float32, float64,
  - complex64, complex128,
  - uint, int, uintptr,
  - bool, string,
  - error

类型

- **组合类型**
  - array, struct, pointer, function,
  - slice, map, channel
- **抽象类型**
  - interface

类型

- **从左到右组合：（Pascal 风格，不同于C）**
  - [10]byte // array of 10 bytes
  - struct {
  - name string
  - left, right *Node
  - action func(*Node)
  - }
  - func(a, b, c int)
  - func(http.ResponseWriter, *http.Request) error

类型

- **类型定义**
  - type Weekday int
  - type Point struct {
  - x, y int
  - }

类型

- **Slices**
  - [] T // slice of T
- **Slice操作**
  - len(s)
  - s[i]
  - s[i:j]
  - append(s, x) // append element x to slice s and return new slice

类型

- **Map**
  - map[K]V // map K -> V
- **Map操作**
  - make(map[K]V)
  - len(m)
  - m[k]
  - delete(m, k)
- **Map迭代**
  - for key, value := range m {
  - // order of key sequence different each time
  - }

语句

- 花括号（C 风格）
- 多重赋值以及其他一些新的结构
- 语句不是表达式（不同于C）

语句

- **if**
  - if x < y {
  - return x
  - } else {
  - return y
  - }
- **switch**
  - switch day {
  - case Mon:
  - ...
  - // break is implicit
  - case Tue, Wed:
  - ...
  - }

语句

- **for loop**
  - for {
  - // loop forever
  - }
- **for range**
  - for i, num := range numbers { ... }
  - for city, pop := range population { ... }

函数

- **普通函数**
  - func Sin(x float64) float64
  - func AddScale(x, y int, f float64) int
- **多个返回值**
  - func Write(data []byte) (written int, err error)
- **可变参数列表**
  - func Printf(format string, args ...interface{})
- **函数式编程特性**
  - var delta int
  - return func(x int) int { return x + delta }

方法

- **一个方法就是一个包含了接受者的函数**
  - 接受者可以是命名类型或者结构体类型的值/指针
- **例，方法定义**
  - type Circle struct {
  - radius float64
  - }

  - //该 method 属于 Circle 类型对象中的方法
  - func (c Circle) getArea() float64 {
  - return 3.14 * c.radius * c.radius
  - }

- **方法调用**
  - func main() {
  - var c1 Circle
  - c1.radius = 10.00
  - fmt.Println("圆的面积 = ", c1.getArea())
  - }

接口

- **接口（interface）用于定义行为的集合**
  - 描述类型必须实现的方法
  - 规定了类型的行为契约
- **隐式实现：**
  - Go 中没有关键字显式声明某个类型实现了某个接口
  - 只要类型实现了接口要求的所有方法，该类型就自动被认为实现了该接口

接口

- **例：接口定义：**
  - type Shape interface {
  - Area() float64
  - Perimeter() float64
  - }

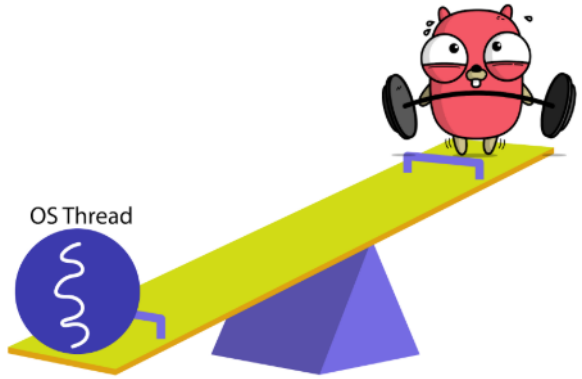
- **接口实现：**
  - // 定义一个结构体
  - type Circle struct {
  - Radius float64
  - }
  - // Circle 实现 Shape 接口
  - func (c Circle) Area() float64 {
  - return math.Pi * c.Radius * c.Radius
  - }
  - func (c Circle) Perimeter() float64 {
  - return 2 * math.Pi * c.Radius
  - }

Go并发

- Don't communicate by sharing memory, share memory by communicating.

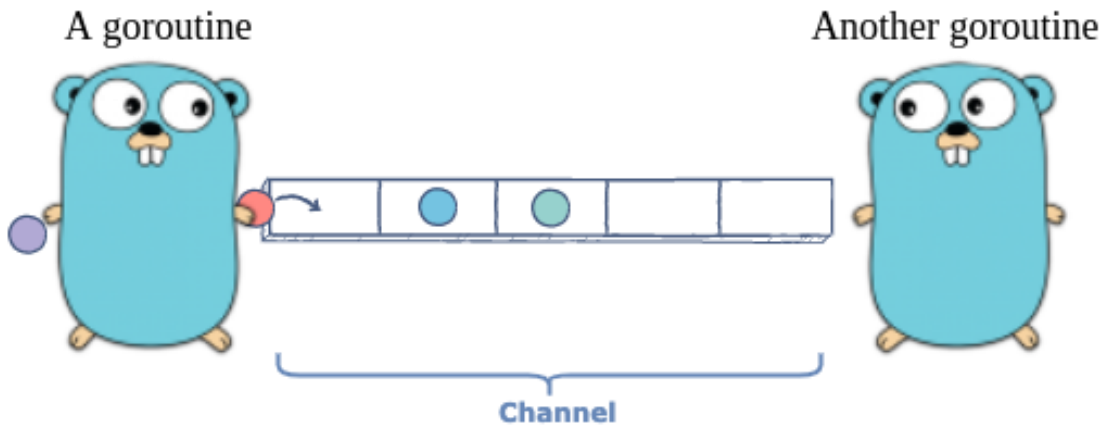
Goroutine

- **Goroutine：**
  - Go 中的并发执行单位，类似于轻量级的线程
  - Goroutine 的调度由 Go 运行时管理，用户无需手动管理
  - 使用 go 关键字启动 Goroutine
    - go f(x, y, z)
  - Goroutine 是非阻塞的，可以高效地运行成千上万个 Goroutine



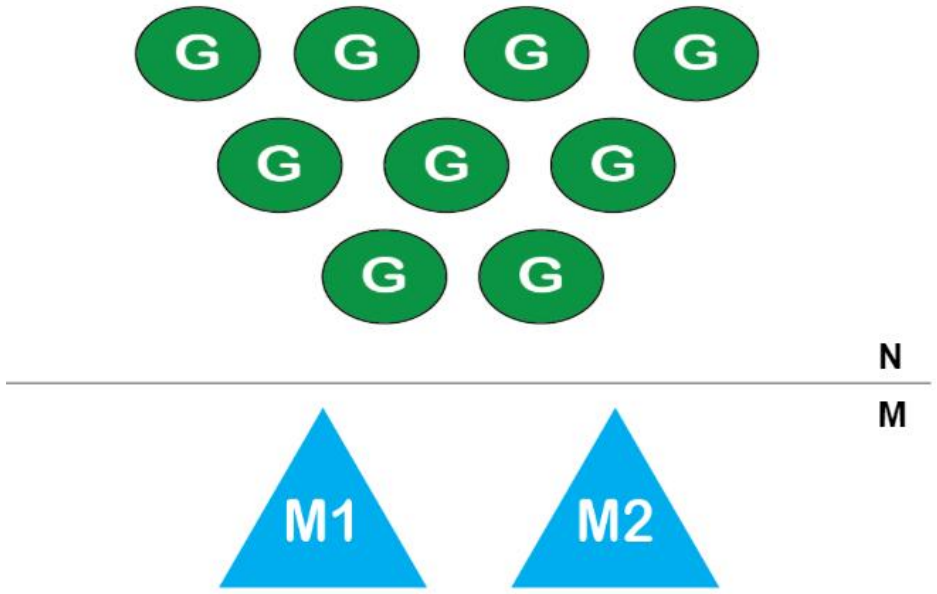
Channel

- **Channel：**
  - Go 中用于在 Goroutine 之间通信的机制
  - 支持同步和数据共享，避免了显式的锁机制
  - 使用 chan 关键字创建，通过 <- 操作符发送和接收数据
  - 缓冲区，有限/无限



Goroutine调度

- **M: N**
  - M个内核线程执行N个goroutine



典型应用

- 云计算基础设施（Docker/Kubernetes）
- 微服务架构（Go-kit）
- 网络服务（ Gin）
- 运维工具（Terraform/Prometheus）
- 企业案例：Google/腾讯/字节跳动

构建分布式系统

- **一些建议：**
  - 构建调试工具
    - 如，日志，全局状态，观察服务器
  - 做好模拟
    - 模拟故障
    - 模拟“分布式”，让系统可以跑在一个进程里，方便调试
  - 确定性
    - 方便重现故障