Go 语言： 互联网时代的 C

Beta 技术沙龙

http://club.blogbeta.com
官方 twitter ： @betasalon
Groups ： http://groups.google.com/group/betasalon

Googol Lee <googollee@gmail.com>

@googollee

2010.04.25

Ken Thompson

Summery

- 系统级开发现状
- 新需求需要新模型
- Go 对并发的支持
- Go 的语法特点

系统测开发现状

- 要求高并发
- 要求开发速度
- 要求性能好
- 要求可分布

系统测开发现状

- C/C++
  - 写的好的话
    - 速度快，内存利用率高
  - 写不好的话
    - 内存泄露
    - Core dump
  - 语言层面完全没有对并发有支持
  - 裸用 os 的并发机制：线程 / 进程

系统测开发现状

- Java
  - 速度快，语言不灵活
  - 语言层面有一定的并发支持，基于 os 并发机制
- PHP/Python/Ruby
  - 开发速度快，灵活
  - 速度慢
  - 语言层面依旧裸用 os 的并发机制，甚至不提供或者有限制（ GIL ）
  - Twisted/asyncore/Multiprocess

能否开发快，性能高？

新模型

新的编程模型（ CSP ）

- 在语言层面加入对并发支持
  - 而不是以库形式提供
- 更高层次的并发抽象
  - 而不是直接暴露 os 的并发机制
- 应用
  - Erlang
  - Ocaml

GO 并发模型

- Goroutine
- Channel
- Rpc
- 内存模型

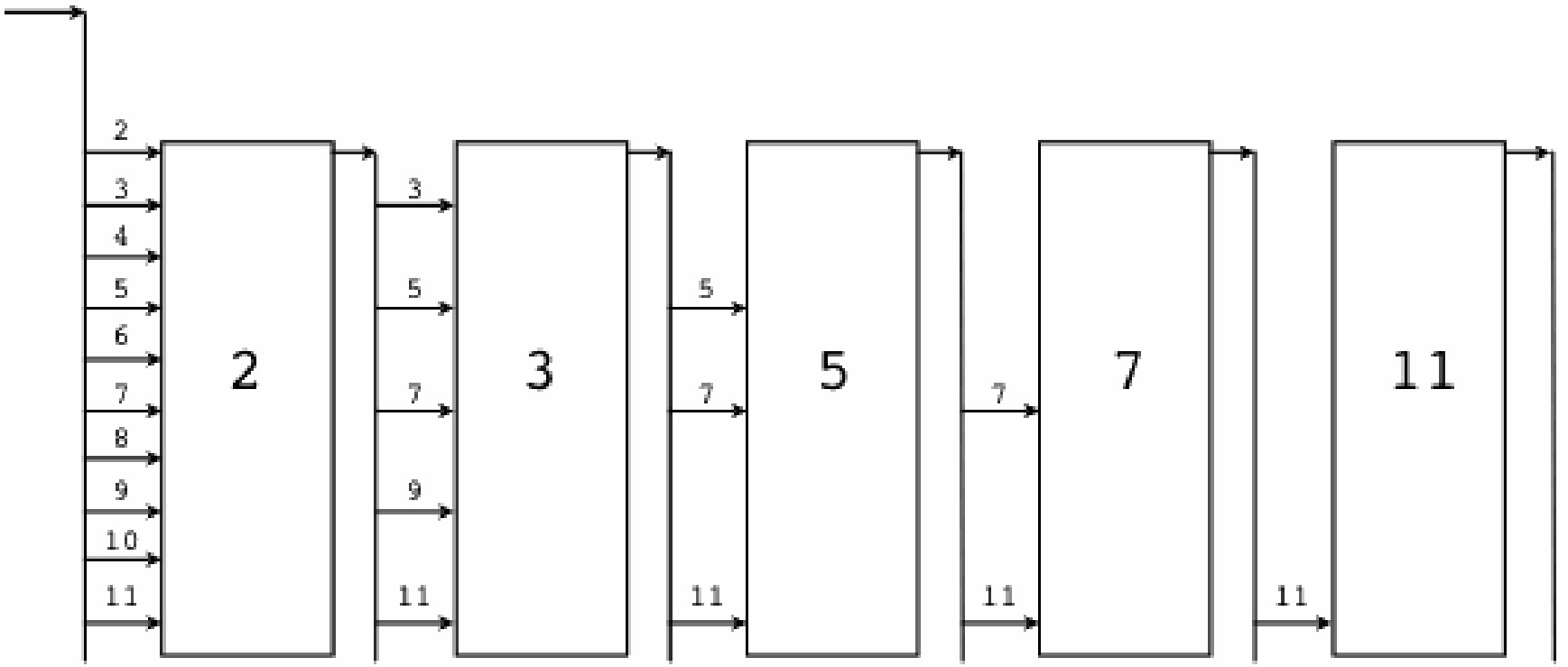
并发模型 - goroutine

- 轻量
- Goroutine 间是并行的
- 底层混合使用非阻塞 IO 和线程
- 关键字： go

并发模型 - channel

- 通过通信来共享
  - 而不是通过共享来通信
- 对 channel 的读写是阻塞的
  - 读阻塞到读出内容
  - 写阻塞到内容写入 buffer
- channel 的两个功能
  - 传值
  - 同步

例子 - 筛法求素数



例子

```
func generate(ch chan int) {
  for i := 2; ; i++ {
    ch <- i  // Send 'i' to channel 'ch'.
} }

func filter(in, out chan int, prime int) {
  for {
    I := <- in  // Receive value of new variable 'i' from 'in'.
    if i % prime != 0 {
      out <- i  // Send 'i' to channel 'out'.
} } }

func main() {
  runtime.GOMAXPROCS(1);
  ch := make(chan int)  // Create a new channel.
  go generate(ch)  // Start generate() as a goroutine.
  for {
    prime := <-ch
    fmt.Println(prime)
    ch1 := make(chan int)
    go filter(ch, ch1, prime)
    ch = ch1 } }
```

例子 - 续

Strace 结果：

- 并发手段：
  - 有 Println ：有一次 clone ，占用一个 core
    - 可以在 write 时继续进行计算
    - 需要 futex 做同步
  - 没有 Println ：没有 clone ，占用一个 core
    - 在 chan 读写时切换 goroutine
    - 没有同步信号量

select

- 同时监听多个 channel

```
for { // loop forever
  select {
    case req := <- service:
      … // process the request
    case <- quit:
      return; // quit loop
} }
```

RPC

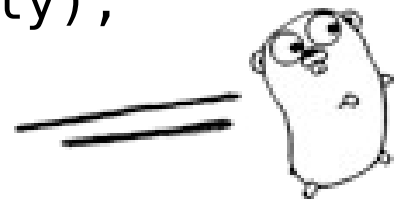
- 远程调用
- 使用 golang 库 gob 作序列化
- 目前只能用 http 做 server
  - http 使用长链接
- 限制
  - gob 不能序列化函数和 channel

例子

```
type Class struct {}
func (p *Class) Function (a *Args, r *Reply) os.Error

// server
func main() {
  a := new(module.Class);
  rpc.Register(a);
  rpc.HandleHTTP();
  l, e := net.Listen("tcp", ":1234");
  http.Serve(l, nil);
}

// client
func main() {
  client, err := rpc.DialHTTP("tcp", "127.0.0.1:1234");
  args := &module.Args{7, 8};
  reply := new(module.Reply);
  err = client.Call("Class.Function", args, reply);
}
```



内存模型

- 简化并发编程必须有 GC 参与
  - 单件如何释放
- 初始化
  - Import package 的 init
    - 包间顺序不定
  - 包里 goroutine 在所有 init 之后才执行
  - Main package 的 main

内存模型

- 原子 IO
  - 对变量的读写都是原子的
    - c/c++ 的变量读写都不是原子的
- 乱序
  - 仅保证在 goroutine 内，乱序后执行结果不变
- once
  - 安全的初始化手段
- Lock
  - 用 channel 更好

语法细节 - 原则

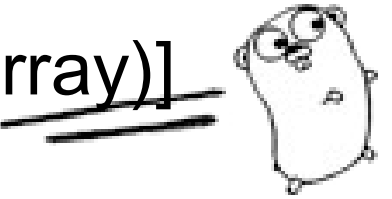
- 静态语言为基础
  - 强类型
- 加入动态语言的优点
- 简化语法
  - 混合了 C 和 Python
- 便于解析
  - 加快编译速度

语法细节 – 值 vs 引用

- 值类型创建
  - 基本类型
    - 字面量： 0 ， 1.1 ， 'c' ，” string” ， [3]int{1,2,3}/[...]
  - 构造
    - Point{1, 1}
- 具有引用语义的基本类型： maps ， slice ， channel
  - maps ， channel 的值类型必须用 make 创建
- 指针
  - 空指针： nil
  - 没有指针运算 - 需要指针运算时应该用 slice
  - 创建
    - var pInt *int = &someIntVar;
    - var point *Point = &Point{1, 1}

语法细节 - slice

- 定义方法
  - 数组： var array [100]int
  - slice ： var slice []int
- slice 可以对数组内任意一段做引用
  - slice = array[X:Y]
  - len(slice) = Y – X
  - cap(slice) ： slice 实际占用的内存
- 取代指针，安全的数组引用
  - slice = &array 等同于 slice = array[0:len(array)]



语法细节 – 类

- 定义方式类似 C

```
type SomeClass struct { … }
func (self *SomeClass) method(...) { … }
```

- 调用方式类似 C++

```
var class *SomeClass = &SomeClass{ … }
class.method( … )
```

- 独特的继承

```
type Base struct { … }
type Child struct { Base; … }
func (p *Child) method() {
  p.BaseMethod( … ); }
```

语法细节 – 非继承

- 动态绑定 interface

```
type PrintInterface interface { print(); }
type Printable struct { … } // no inherit here
func (p *Printable) print() { … }
var i PrintInterface = &Printable{ … }
i.print()
```

- Any
  - 可以传入任意类型

```
type Any interface {}
```

  - 调用方式

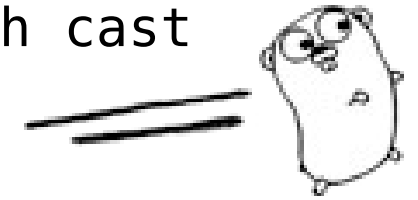
```
any.(PrintInterface).print()
```

语法细节 - type

- True typedef
  - 让错误的代码~~显而易见~~直接报错

    ——《软件随想录》 P189
  - 例子：

```
type UnsafeString string;
type SafeString string;
var input UnsafeString = form.data;
var valid SafeString;
valid = input; // compile error.
valid = SafeString(input); // can convert with cast
```



语法细节 – const

- 常量不用定义类型

```
const MAX_NUMBER = 100;
const DEFAULT_STRING = “string”;
```

- 枚举是特殊的常量，特殊递增符 iota

```
const (
  _ = iota      // ignore
  A = iota * 2 // 1 * 2 = 2
  B             // 2 * 2 = 4, inherit from above
  C             // 3 * 2 = 6
  _             // ignore, iota = 4
  D             // 5 * 2 = 10
)
```

语法细节 - 实现 C++0x

- Auto

```
x := 6; y := make(chan int);
```

- Concept
  - 使用 Interface 在运行期实现
- GC
- Lambda

```
someFunc = func ( … ) { … }
someFunc()
```

- 闭包

包管理

- 首字母大写是 public ，小写是 private
- 需要预先编译才能 import
- 已有库
  - *nix/c 标准库 - os ， rand
  - C 互操作 - C
  - Container - heap ， list ， ring ， vector ， hash
  - golang 的词法 / 语法分析库 – ast
  - 网络库 - websocket ， http ， json

Q&A

http://golang.org