让我们一起来学习

GO语言编程

晨笛　2013.02.20

1. 为什么我们需要Go语言
2. Go语言的简介
3. Go语言语法
4. 面向对象编程
5. 并发编程

系统开发现状

- 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

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

➢是什么促使Go的出现

随着机器性能的提升、软件规模与复杂度的提高，Java逐步取代了单机时代的编程之王C语言的位置。然后Java编程的体验并未尽如人意。历年来的编程语言排行榜显示，Java语言的市场份额在逐步下跌。Go语言此时应运而生，Go语言官方自称Go语言的出现是因为“近10年来开发程序之难让我们有点沮丧”。Go希望成为互联网时代的C语言。

➢互联网时代的C语言需要考虑哪些关键问题呢

✓**并行与分布式支持**

多核化和集群化是互联网时代的典型特征。

✓**软件工程支持**

互联网时代C语言需要考虑软件品质保障和团队协作相关的话题

✓**编程哲学的重塑**

互联网时代C语言需要回答什么才是最佳的编程实践这个问题。

➢Go语言的在这些问题上的处理

✓并行与分布式支持

•并发执行的“执行体”。Go语言在语言级别支持协程（微线程）。多数语言在语法层面并不直接支持协程，而通过库的方式支持协程的功能也不完整。

•执行体间的互斥和同步。Go语言提供协程之间的互斥和同步。

•执行体间的消息传递。多数语言在并发编程模型上选择了共享内存模型，而Go语言选择了消息队列模型。

✓软件工程支持

Go语言可能是第一个将代码风格强制统一的语言。

➢Go语言的在这些问题上的处理

✓**编程哲学的重塑**

Go语言用批判吸收的眼光，融合众家之长，但时刻警惕特性复杂化，极力维持语言特性的简洁，力求小而精。

- Go语言反对函数和操作符重载
- Go语言放弃构造和析构函数
- Go语言支持类、类成员方法、类的组合，但反对继承，反对虚函数和虚函数重载

➢Go语言的目标

✓提升现有编程语言对程序库等依赖性(dependency)的管理。

✓解决多处理器的任务

➢Go语言的特色

简洁、快速、安全、并行、有趣 、开源、支持泛型编程、内存管理、数组安全、编译迅速。

1 为什么我们需要Go语言

2 **Go语言的简介**

3 Go语言语法

4 面向对象编程

5 并发编程

➢Go语言简史

Go语言是由贝尔实验室包括肯·汤普森在内的Plan 9原班人马开发。Go语言的第一个版本在2009年11月正式对外发布，并在此后的两年内快速迭代，发展迅猛。第一个正式版本在2012年3月28正式发布。

➢Go语言特性

- ✓自动垃圾回收
- ✓更丰富的内置类型
- ✓函数多返回值
- ✓错误处理
- ✓匿名函数和闭包
- ✓类型和接口
- ✓并发编程
- ✓反射
- ✓语言交互性

1 为什么我们需要Go语言

2 Go语言的简介

3 **Go语言语法**

4 面向对象编程

5 并发编程

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

➢变量

✓变量声明

```
var v1 int
var v2 string
var v3 [10] int
var (
     v1 int
     v2 string
)
```

✓变量初始化

```
var v1 int = 10     //ok
var v2 = 10         //ok
v3 := 10            //ok
v2 := 10            //error
```

➢变量

✓变量赋值

```
var v1 int
v1 = 123
i, j = j, i     //多重赋值，交换
```

✓匿名变量

```
func GetName () (firstName, lastName, nickName string) {
     return “May”, “Chan”, “Chibi Maruko”
}

_, _, nickName := GetName ()//仅获取nickName, _为匿名变量占位符
```

➢常量

- ✓字面常量
- ✓常量定义
- ✓预定义常量

Go语言预定义了true， false，iota

```
const (                 //iota重置为0
        c0 = iota       //c0为0
        c1 = iota       //c1为1
        c2 = iota       //c2为2
)
```

- ✓枚举

```
const (
        Sunday = iota
        Monday
        numberDays      //这个常量未导出
)
```

➢类型

✓基础类型

- ➢布尔类型 bool
- ➢整型 int8 byte int16 int uint uintptr
- ➢浮点类型 float32 float64
- ➢复数类型 complex64 complex128
- ➢字符串 string
- ➢字符类型 rune
- ➢错误类型 error

✓复合类型

- ➢指针（pointer）
- ➢数组（array）
- ➢切片（slice）
- ➢字典（map）
- ➢通道（chan）
- ➢结构体（struct）
- ➢接口（interface）

➢流程控制

✓条件语句

```
if a < 5 {
    return 0
} else {
    return 1
}
```

✓选择语句

```
switch i {
    case 0 :
        fmt.Printf (“0”)
    case 1 :
        fallthrough
    case 2, 3 :
        fmt.Printf (“2,3”)
    default:
        fmt.Printf (“default”)
}
```

➢流程控制

✓选择语句

```
switch  {
    case 0 <= Num && Num <= 3 :
         fmt.Printf (“0-3”)
    case 4 <= Num && Num <= 6 :
         fmt.Printf (“4-6”)
}
```

✓循环语句

```
//case 1
sum := 0
for i := 0; i < 10; i++ {//条件表达式中也支持多重赋值
    sum += i
}
//case2
sum := 0
for {          //相当于while，do-while
    sum++      //支持按标签break
}
```

➢流程控制

✓跳转语句

支持continue、break、goto，break可按标签选择中断到哪一个循环

➢函数

✓函数定义

函数的基本组成为：关键字func、函数名、参数列表、返回值、函数体和返回语句。

```
func Add (a int, b int) (ret int, err error) {
    if a < 0 || b < 0 {
        err = errors.New (“error”)
        return
    }
    return a + b , nil //多重返回值
}
```

✓匿名函数和闭包

匿名函数由一个不带函数名的函数声明和函数体组成。匿名函数可以直接赋值给一个变量或直接执行。
闭包是可以包含自由变量的代码块。相当于Java中的嵌套匿名类

➢错误处理

✓大多数函数将error作为最后一个返回值

✓defer

defer语句的含义是不管程序是否出现异常，均在函数退出时自动执行相关代码。

```
func CopyFile (dst, src string) (w int64, err error)
{
   srcFile, err := os.Open (src)
   if err != nil {
      return
   }
   defer srcFile.Close ()
}

//清理多语句，使用匿名函数
defer func () {
    //清理工作
} ()
```

1 为什么我们需要Go语言

2 Go语言的简介

3 Go语言语法

4 **面向对象编程**

5 并发编程

➢类型系统

Go语言中的大多数类型都是值语义，并且都可以包含对应的操作方法。在需要的时候，你可以给任何类型（包括内置类型）增加新方法。而在实现某个接口时，无需从从该接口继承，只需要实现该接口要求的所有方法即可。

```
type Integer int
func (a Integer) Less (b Integer) bool {
    return a < b
}
func main () {
    var a Integer  = 1
    if a.Less (2) {
        fmt.Println (a, “Less 2”)
    }
}
```

➢可见性

Go语言中要使某个符号对其他包（package）可见，只需要将该符号定义为以大写字母开头，否则不可见。成员方法和成员变量的可见性遵循同样的规则。Go语言中符号的可访问性是包一级的而不是类型一级的。

```
type Rect struct {
    X, Y float64
    Width, Height float64
}

func (r *Rect) area () float64 {
    return r.Width * r.Height
}
```

➢接口

✓非侵入式接口

在Go语言中，一个类只要实现了接口要求的所有函数，我们就说这个类实现了该接口。
这种方式有如下几种好处：

- Go语言不需要绘制类库的继承树图
- 实现类的时候，只需要关心自己应该提供哪些方法，不用纠结接口需要拆的多细才合适。
- 不用为了实现一个接口而导入一个包，减少耦合。

```
type File struct {
      // …
}
func (f *File) Read (buf [] byte) (n int, err error)
func (f *File) Write (buf [] byte) (n int, err error)
func (f *File) Close () error
```

➢接口

```
type IFile interface {
    Read (buf [] byte) (n int, err error)
    Write (buf [] byte) (n int, err error)
    Close () error
}
type  IReader interface {
    Read (buf [] byte) (n int, err error)
}
type IWriter interface {
    Write (buf [] byte) (n int, err error)
}
type ICloser interface {
    Close () error
}
var file1 IFile = new (File)
var file2 IReader = new (File)
var file3 IWriter = new (File)
var file4 ICloser = new (File)
```

1. 为什么我们需要Go语言
2. Go语言的简介
3. Go语言语法
4. 面向对象编程
5. **并发编程**

➢协程

Go语言在语言级别支持轻量级线程（即协程），叫goroutine。Go语言提供的所有系统调用操作都会出让CPU给其他goroutine。这让事情变得非常简单，让轻量级线程的切换管理不依赖于系统的线程和进程，也不依赖于CPU的核心数量。

Go语言中最重要的一个特性是go关键字

```
func Add (x, y int) {
      z := x + y
}

go Add (2, 1)     //并发执行
```

➢并发通信

工程上两种最常见的并发通信模型：

- ✓共享数据
- ✓消息

一个大的系统中具有无数的锁、无数的共享变量、无数的业务逻辑与错误处理分支。采用共享数据将是一场噩梦。Go语言已并发编程作为最核心优势，提供了以消息机制而非共享内存作为通信方式的通信模型（channel）。

channel是类型相关的的，一个channel只能传递一种类型的值，这个类型需要在声明的channel时指定。channel相当于一种类型安全的管道。

➢并发通信

✓channel

一般channel的声明形式：var chanName chan ElementType
定义一个channel：ch := make (chan int)
将一个数据写入channel：ch <- value
从channel中读取数据：value := <- ch

✓select

Go语言直接在语言级别支持select关键字，用于处理异步IO问题。
select有比较多的限制，其中最大的限制就是每个case语句里必须是一个IO操作。

```
select {
    case <- chan1 :
    case chan2 <- 1 :
    default :
}
```

➢并发通信

✓缓冲机制

创建一个带缓冲的channel： c := make (chan int, 1024)

✓超时机制

Go语言没有提供直接的超时处理机制，但我们可以利用select机制实现一套

```
timeout := make (chan bool, 1)
go func () {
     time.Sleep (1e9) //等待1秒
     timeout <- true
} ()
select {
    case <- ch :
    case <- timeout :
    default :
}
```

➢并发通信

✓channel的传递

Go语言中channel本身是一个原生类型，因此channel可以传递。
下面我们利用这个特性来实现*nix常见的管道特性

```
type PipeData struct {
    value int
    handler func (int) int
    next chan int
}

func handler (queue chan *PipeData) {
    for data := range queue {
        data.next <- data.handler (data.value)
    }
}
```

➢并发通信

✓单向channel

我们在将一个channel变量传递到一个函数时，可以通过将其制定为单向channel变量，从而限制该函数中可以对此channel的操作。

```
var ch1 chan int        //ch1是一个正常的channel，不是单向的
var ch2 chan<- float64 //ch2是单向的，只用于写float64数据
var ch3 <-chan int      //ch3是单向的，只用于读取int数据

//单向channel与双向channel之间的转换
ch4 := make (chan int)
ch5 := <-chan int (ch4)     //ch5为单向的读取channel
ch6 := chan<- int (ch4)     //ch6为单向的写入channel
```

➢多核并行化

我们可以通过设置环境变量GOMAXPROCS的值来控制使用多少个CPU核心。

runtime.GOMAXPROCS (16)

runtime包中还提供了一个函数NumCPU ()来获取CPU核心数。

➢全局唯一性操作

Go语言提供了一个Once类型来保证全局的唯一性操作。Once的Do ()方法可以保证在全局范围内只调用指定的函数一次，而且所有其他goroutine在调用到此语句时，将会先被阻塞，直至全局唯一的Once.Do ()调用结束才继续。

GO学习结束语

快速简单的编译：

go语言编译的很快，事实上，他快的甚至可以作为脚本语言了。几个使他编译很快的原因有：

- 他不使用头文件
- 当A依赖B，B又依赖C时，那么首先会编译C，然后是B和A；但是如果A依赖B，但是A并不直接依赖于C，而是存在依赖传递，这时会把所有B需要从C拿到的信息放在B的对象代码里。这样，当编译A的时候，就不需要再管C了。
- 在编译程序时，只需将类型信息沿着依赖关系树向上遍历即可，如果到达树的顶端，则只需编译紧邻的依赖，而不用管其它层级的依赖了。

GO学习结束语+

通过多返回值的错误处理：

现代的编程语言基本上有两种错误处理办法，例如在C语言里是使用返回值，而在Java等面向对象语言里使用异常处理返回值，因为返回值的状态码总是可能跟 需要返回的结果有冲突。Go语言允许多返回值，从某种程度上解决了这个问题。你可以为你的函数的执行结果状态定义返回值，任何调用的时候都可以来检查，很 方便。

简单的组合：

可以使用interface为对象指定一些类型的成员，还可以像Java一样给他们指定操作（行为）。例如在标准库的io包中定义了一个Writer，就 有一个带有字节数组作为参数（输入）一个integer值和错误码作为返回值（输出）的方法。而实现了io.Writer接口中的Write方法的类型才 是实际被执行的。这个设计能够非常优雅的分离代码，还简化了单元测试过程，例如，如果你想测试一个数据库对象的一个方法，在传统的语言中你必须创建一个数 据库对象，然后做很多协议初始化工作。在Go语言中，你可在接口下创建任何对象。

简单的并发：

在Go中并发变得非常的简单，在任何函数前方上go两个字母，这个函数就将以他自己的go-routine（一个非常轻量级的线程）来运行，Go- routines之间通过channels来通信。我们通常会有一些需要线程同步和互斥的需求，在Go中非常简单，Go只是启动并发任务，各个任务之间通 过channels来通信，从而协调同步和互斥。

GO学习结束语+

优秀的错误提示：

我从没见过别的语言有Go语言这么高的错误诊断质量。例如如果你的程序死锁了，Go的运行时可以通知你，而且，他甚至可以告诉你是哪个线程出了问题。当然编译错误也是很详细很有用的。

其他特性：

Go语言还有其他非常吸引人的特性：高阶函数、垃圾回收、哈希映射、可扩展的数组等等。

参考及鸣谢

Go语言代码分享：http://www.sharejs.com/codes/go/

Go指南：http://tour.golang.tc/#1

根据网络资料整理多谢相关作者

THE END

HTANK YOU