链滴

Golang 简介

作者：feiwo

原文链接：https://ld246.com/article/1609664748617

来源网站：链滴

许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

Go语言介绍

Go是一门开源的由Google公司推出的通用型编程语言，目的在于降低构建简单，可靠，高效软件的槛。**Go**借鉴了诸多语言的一些特性，以及现代语言中常见的一些高级特性，且拥有自己独特特性的门编程语言，**Go**语言可以用来构建非常快捷，高性能且有足够控制力的编程环境。

Go具有足够少的关键字，内置并发机制，没有特定的线程库(Java的Thread,PHP的thread扩展)，开一个线程只需要一个关键字go，即可享受多线程带来的高性能，内置了**CAS**特性的atomic库，以及规的锁机制sync包，**Go**主张通过通信来共享内存，而不是通过共享内存来通信，通过chan这种数据构来进行线程之间通信，共享内存资源。**Go**开发并不需要特别关心内存的使用，因为其自身自带了圾回收器，帮助我们很好的管理了内存的使用。

Go的面向对象模式与其他传统面向对象语言不同，没有class关键字，而是采用了struct关键字，也有继承(extends)这种方式，采用组合模式对struct进行扩展，达到类似继承实现的方式。**Go**的接口与其他传统面向对象不同，采用的是Duck Type实现方式，也就是所谓的鸭子模型，即这个东西看起像个鸭子，那么它就是一只鸭子，接口的实现也是非显式实现的，也不存在implements关键字，只某个类型实现了这个接口所有方法，那么这个类型就实现了这个接口。与其他语言相比，**Go**的方法并不包含在对象里面，而是采用函数绑定的方式，将函数挂载在某个类型上。

Go被称为21纪的C语言，以其语法简单，高性能，高并发，跨平台，部署方便等著称，在云计算领域展拳脚，拥有Docker,K8s等多个杀手级项目。

发展历程

Go诞生在2007年，2008年核心团队组建，2009年进行开源，随后在2012年**Go**的1.0版本发布，进三年的开发，**Go**在2015的1.5版本实现了自举。此后**Go**团队决定一年发布两个版本，2016年发布了1.6和.7，语法分析器从yacc改为硬编码实现，GC延迟更新，x86编程生成SSA(static single assignment)间代码，2017发布了1.8和1.9， 对struct语言规范微调，增加了类型别名，sync包增加了并发安全的ap，2018年发布了1.10和1.11，加快了build构建的速度，在SDK中内置了包管理器，go mod，并供了把**Go**程序编译成可以兼容四大主流浏览器的二进制格式的能力。2019年，发布了1.12和1.13，进了sync包下的pool，重写了逃逸分析，减少**Go**程序中在堆内存申请的空间。2020年，发布了1.14和

.15， 提高了defer的使用性能，新增了maphash,unicode11.0升级到unicode12.0,新增了时区包tzdata,refect包将不允许访问未导出的字段和方法。

设计哲学

少即是多

世界是并行的

组合优于继承

非侵入式接口

少即是多

Go语言简洁，只有25个关键字，支持类型推导，语句结尾不用分号;，可编译成直接运行的二进制文件

类型推导:

```
var name = "feiwo" // 自动推导变量为 string类型

func show() {
  age := 18 // 自动推导变量为 int类型
}
```

世界是并行的

Go要想使用并发能力很简单，一个go关键字就可以了

```
func fetch(url string) {
  resp, err := http.Get(url)
  if err != nil {
    log.Printf("fetch error %s", err.Error())
  }
  ...
}

// 每个Go程序都会有一个主的goroutine，称之为g0
func main() {
  // 是用go关键字单独启动一个goroutine, Go的线程并不是系统线程，Go线程与系统线程的并比为m
n，且go线程由的runtime进行调度与管理
  go fetch("https://www.feiwo.xyz")

  select{}
}
```

组合优于继承

Go采用组合的方式来实现多个类型的聚合

```
type Person struct {
  Name string
  Age  int
}
```

```
type Student struct {
  Person // 组合Person struct, 拥有Person的字段
  ClassRoom string
  Score       float32
}

type Teacher struct {
  Person // 组合 Person struct, 拥有Person的字段
  Salary  float32
}

func main() {
  stu := Student {
    Person: Person {
      Name: "xiao qi",
      Age: 10,
    },
    ClassRoom: "三年级一班",
    Score: 98.2,
  }
  // 直接通过实例拿到组合结构体中定义的字段
  fmt.Println(stu.Name) // out: xiao qi

  teacher := Teacher {
    Person: Person{
      Name: "feiwo",
      Age: 38,
    }
    Salary: 9000.2,
  }

  fmt.Println(teacher.Name) // out: feiwo
}
```

非侵入式接口

在**Go**中，提倡小接口，组合成大接口

```
type Jwt interface {
  Gen(data interface{}) (string, error)
}

type jwt struct {

}

// jwt结构体实现了Jwt接口，不需要显示指定实现了什么接口
// 只要该结构体中方法包含实现接口中所有的方法即可，方法的签名与接口中定义的方法签名一致
func (j *jwt) Gen(data interface{}) (string, error) {
  // 生成jwt token
  token, err := ...
  if err != nil {
    return "", err
```

```
  }
  return token, nil
}

// 在Go中，可以通过实现接口的方式，实现多态的效果
type Reader interface {
  Read(p []byte) (int, error)
}

type ReadFile struct {

}

func (r *ReadFile) Read(p []byte) (int, error) {
  // TODO 具体实现
}


type ReadNet struct {

}

func (r *ReadNet) Read(p []byte) (int, error) {
  // TODO 具体实现
}


func Processer(r Reader) {
  // TODO 具体实现
}

// use
func main() {
  readFile := new(ReadFile)
  readNet := new(ReadNet)

  // 使用接口实现多态
  Processer(readFile)
  Processer(readNet)
}
```