系统编程语言明日之星——Go

■ 文 / 余锋

作者简介：

余锋，就职于淘宝核心系统研发部。多年C和网络开发经验，专注于高性能、容错、分布式服务器的研究和实现。

Go语言由Google提供支持，于2007年9月开始浮出水面，出自Rob Pike、Ken Thomason等大师之手，是一种支持并发和垃圾回收的编译型编程语言。

Go语言是基于BSD许可发行的开源项目，目前只支持Linux、Mac OS X和Windows平台。由于团队资源及社区规模的限制，目前Go还无法支持更多的操作系统，因此需要更多的参与者来进行移植。

Go语言创建伊始，就目标明确——创建一种高性能、开发迅速、安全且富有趣味的系统编程语言。各种高性能的网络服务器、文件存储系统、数据库等，都非常适合使用Go进行开发。当然Go也可以运用在其他领域，Go的官方网站golang.org就是使用gdoc实现的。

Go 已经在Google公司内部尝试使用，但是并没有大规模地部署，因此仍处于试验阶段，需要丰富的测试和考验。

Go已经给了我们一个华丽的亮相，在工程层面，Go语言官方目前还没有自己的编辑器， 但是发布包带了Emacs、VI以及Xcode的插件，方便用户的编辑。它目前有2个编译器：源于Plan9的Native 8/6g编译器和以GCC为后端的Gccgo。 Gccgo能编译出来更高效的代码，还可以和其他的gcc库相链接。但是遗憾的是Gccgo尚未实现垃圾收集，Goroutine通过pthread实现，调度开销大且栈空间占用多，而且编译时间较长，无法在实际项目中使用。 排错器和调优器目前还不大可用。

我看好Go的原因是，这个语言不是凭空或者实验室设计的，而是填补过去10年在系统编程软件方面的不足。作为系统软件的编写者，经常会深深地感到目前常见系统语言在面对大规模网络应用的力不从心，在不断涌现的多核心硬件前的无奈，在多线程和各种锁中间的痛苦，在不停地制造各种各样的轮子的困惑， 以及在使用很原始的排错器的低效。Go拥有的这些现代特性都是系统用户梦寐以求的，既结合了动态语言的弹性，又有了静态语言的安全性和高性能。Rob Pike指出Go代码的编译基本上可以达到C语言的速度，几乎可以在瞬间完成。而在运行效率上已经和C语言很接近，同时支持多核心并发运算。来自语言性能比较网站Computer Language Benchmarks Game的数据也验证了这一点。目前这个Go社区非常活跃，新的特性和包在不断地被提出和加入，加上有Google在背后强力支持和推广，个人认为应用前景非常广阔。

Go能解决什么问题呢?

我们在编写系统程序时总是会面临以下挑战：第一，硬件发展得很快，软件技术总是突破得很慢；第二，复杂的系统依赖性很强，平台、库、管理这些依赖相当麻烦；第三，目前可用的系统语言如C++、Java太笨拙，也太复杂，在处理复杂情况时感觉力不从心；第四，复杂的内存管理和资源管理；第五，多核心机器的涌现，在未来几年里面128核心的机器应该会很常见，我们难道还是用最原始的线程机制和同步原语来面对?

大部分系统语言并不是设计来解决这些问题的，而且新涌现的大多数程序库也并没有改善这些问题，相反加剧了复杂度。正如Rob Pike说：“我们开发Go，是因为近10年来开发系统程序之难让我们有点沮丧。 这时候Go来拯救我们啦！”

设计目标

- 系统编程语言
- 快速编译和执行
- 类型安全性和内存使用安全
- 很好的支持并发计算
- 高效低延迟的垃圾回收算法

设计原则

- 更少的关键词，减少无谓的输入

- 保持概念正交
- 保持简单
- 减少类型，无类型层次，避免罗嗦

语言特性

Go语言脱胎于C语言，同时还从社区的其他语言如Pascal、 CSP、Occam借鉴了很多特性，添加了如包管理、反射、 垃圾回收器、动态类型、并发以及并行机制等。

它有一些有趣的特性：

- 融合多种语言的特性，各个社区的人都感觉会很熟悉，特别是C、Lua、Python社区的用户非常容易上手。
- 支持面向对象编程，但是不支持层次继承。
- 清晰、精确的语法。
- 大写符号导出公共符号，小写是私有的。
- 函数或者块开始注释是文档，方便gdoc生成文档。
- 改进的If、Case复合语句，允许在条件判定前，执行初始语句。Case不支持自动fall through，支持多个判断条件和表达式。
- 轻量的类型系统，强类型，没有隐式的转换，显式类型转化。
- 指针运算由slice代替，提高安全性的同时也有很高的效率。
- 内置基本数据类型数组和字典。
- type：相当灵活的struct，支持匿名字段，复用看起来很优雅。
- 灵活但独特的interface：接口类型， 接口实现分离。若一个struct实现了interface声明的方法则视为实现此interface。通用类型interface{}用来表示同样类型，用于实现包容器和用于参数省略的场合。
- range：for语句支持通用的迭代，支持数组、字典、通讯管道。 特别是通讯管道结合Goroutine，由生产者负责在一端输入数据，range充当消费者，从另外一端消费数据。
- reflection：用于实现多参数机制和实现比如XML、JSON库，很直观地把结构的字段名和值映射起来，但是执行效率不高。
- defer负责资源的自动释放，有效地避免资源泄漏。

语言亮点

闭包：函数是第一类对象。有了闭包，我们很容易把数据和函数结合在一起，形成一个独立的执行体，无需关心数据的泄漏，是并发编程的基础。

Channel：CSP的核心思想是通过消息共享，而不是内存共享。而消息共享机制就是通过Channel来传递消息，有同步和异步之分。Channel在生产者和消费者之间架设起沟通的桥梁，承担起传统语言消息队列的作用。

Goroutine：Goroutine是使并发编程变得容易的核心。实现上是通过系统的线程来多路派遣协程，使得函数的执行看起来独立。当一个协程阻塞的时候，比如说费时的系统调用，调度器就会自动把其他协程安排到另外的线程去执行，从而保证了系统的不间断运行。这个调度对于程序员是透明的，从用户的角度来看，协程在一直运行。而且这个调度的开销非常小，典型的CPU每秒钟可以调度百万次，协程还支持堆栈的自增。这两个设计使得我们可以创建大量的goroutine，模拟现实世界对象的行为，大大简化了程序的设计和实现。

Network支持：目前支持Tcp/Udp协议，有统一的编程接口，方便未来支持更多的协议，通过内核POLL支持成千上万的并发连接。在网络读写暂时无效的时候，网络模块会自动把该句柄注册到系统的poll set，并且让出执行权，等待读写事件的发生。一旦读写事件得到通知，网络模块内部通过Channel通知刚才被阻塞的操作继续执行。从用户使用角度来看，网络的读写都是顺序执行的，极大地简化了编程。

正是 Network+闭包+Goroutine+Channel让Go语言这么突出，突破了用户在编写大规模系统程序时的局限。

库实现

- 完全用Go语言实现， 不依赖其他系统库。
- 模块管理。一个Package可以分散在多个文件里，通过包名整合在一起。支持包层面的初始化操作。
- 接口库如XML、Asn1、JSON、IO库、压缩、加密库等，比较齐全，极大地方便了用户。
- 支持Unicode。

与其他语言的交互

Cgo 支持C和Go语言编程混合，使得Go语言的扩展非常容易，让Cgo来做参数和类型转换，直接连接用户的C函数。

目前存在的问题

- 语言的面世时间比较短，实现和社区都不够成熟，一些关键的语言特性如select timeout还没有实现，编译器也不大完善。
- 语言库少，主要体现在缺少高级网络、GUI、数据库访和媒体处理库。
- 基于Go的应用还比较少，甚至在Google内部也还是个试验项目，还没有大规模的应用。