<<Go语言编程>>

图书基本信息

书名：<<Go语言编程>>

13位ISBN编号：9787115290366

10位ISBN编号：7115290369

出版时间：2012-8

出版时间：人民邮电出版社

作者：许式伟

前言

为什么我们需要一门新语言　　编程语言已经非常多，偏性能敏感的编译型语言有C、C++、Java、C#、Delphi和Objective-C等，偏快速业务开发的动态解析型语言有PHP、Python、Perl、Ruby、JavaScript和Lua等，面向特定领域的语言有Erlang、R和MATLAB等，那么我们为什么需要Go这样一门新语言呢？

在2000年前的单机时代，C语言是编程之王。
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C的位置。
尽管看起来Java已经深获人心，但Java编程的体验并未尽如人意。
历年来的编程语言排行榜（如图0-1所示）显示，Java语言的市场份额在逐步下跌，并趋近于C语言的水平，显示了这门语言后劲不足。

图0-1编程语言排行榜①　　Go语言官方自称，之所以开发Go语言，是因为“近10年来开发程序之难让我们有点沮丧”。
这一定位暗示了Go语言希望取代C和Java的地位，成为最流行的通用开发语言。
Go希望成为互联网时代的C语言。
多数系统级语言（包括Java和C#）的根本编程哲学来源于
——————————　　① 据来源：http：//www.tiobe.com/index.php/content/paperinfo/tpci/index.html。

C++，将C++的面向对象进一步发扬光大。
但是Go语言的设计者却有不同的看法，他们认为C++真的没啥好学的，值得学习的是C语言。
C语言经久不衰的根源是它足够简单。
因此，Go语言也要足够简单！

那么，互联网时代的C语言需要考虑哪些关键问题呢？
首先，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。
作为一个互联网时代的C语言，必须要让这门语言操作多核计算机与计算机集群如同操作单机一样容易。
其次，软件工程支持。
工程规模不断扩大是产业发展的必然趋势。
单机时代语言可以只关心问题本身的解决，而互联网时代的C语言还需要考虑软件品质保障和团队协作相关的话题。
最后，编程哲学的重塑。
计算机软件经历了数十年的发展，形成了面向对象等多种学术流派。
什么才是最佳的编程实践？
作为互联网时代的C语言，需要回答这个问题。
接下来我们来聊聊Go语言在这些话题上是如何应对的。

并发与分布式　　多核化和集群化是互联网时代的典型特征，那语言需要哪些特性来应对这些特征呢？
第一个话题是并发执行的“执行体”。
执行体是个抽象的概念，在操作系统层面有多个概念与之对应，比如操作系统自己掌管的进程（process）、进程内的线程（thread）以及进程内的协程　　（coroutine，也叫轻量级线程）。
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的协程的功能也并不完整，比如仅仅提供协程的创建、销毁与切换等能力。
如果在这样的协程中调用一个同步IO操作，比如网络通信、本地文件读写，都会阻塞其他的并发执行协程，从而无法真正达到协程本身期望达到的目标。

Go语言在语言级别支持协程，叫goroutine。
Go语言标准库提供的所有系统调用（syscall）操作，当然也包括所有同步IO操作，都会出让CPU给其

他goroutine，这让事情变得非常简单。
我们对比一下Java和Go，近距离观摩下两者对“执行体”的支持。
为了简化，我们在样例中使用的是 Java标准库中的线程，而不是协程，具体代码如下： public class MyThread implements Runnable { String arg； public MyThread(String a) { arg = a； } public void run() { // ... } public static void main(String[] args) { new Thread(new MyThread(“”test“”)).start()； // ... } } 相同功能的代码，在 Go语言中是这样的： func run(arg string) { // ... } func main() { go run(“”test“”) ... } 对比非常鲜明。
我相信你已经明白为什么 Go语言会叫 Go语言了：Go语言献给这个时代最好的礼物，就是加了 go这个关键字。
当然也有人会说，叫 Go语言是因为它是 Google出的。
好吧，这也是个不错的闲聊主题。
第二个话题是“执行体间的通信”。
执行体间的通信包含几个方式： ·执行体之间的互斥与同步 ·执行体之间的消息传递 先说·执行体之间的互斥与同步·。
当执行体之间存在共享资源（一般是共享内存）时，为保证内存访问逻辑的确定性，需要对访问该共享资源的相关执行体进行互斥。
当多个执行体之间的逻辑存在时序上的依赖时，也往往需要在执行体之间进行同步。
互斥与同步是执行体间最基础的交互方式。
多数语言在库层面提供了线程间的互斥与同步支持，那么协程之间的互斥与同步呢？
呃，不好意思，没有。
事实上多数语言标准库中连协程都是看不到的。
再说·执行体之间的消息传递·。
在并发编程模型的选择上，有两个流派，一个是共享内存模型，一个是消息传递模型。
多数传统语言选择了前者，少数语言选择后者，其中选择·消息传递模型·的最典型代表是 Erlang语言。
业界有专门的术语叫“ Erlang风格的并发模型”，其主体思想是两点：一是“轻量级的进程（ Erlang中‘进程’这个术语就是我们上面说的‘执行体’）”，二是“消息乃进程间通信的唯一方式”。
当执行体之间需要相互传递消息时，通常需要基于一个消息队列（message queue）或者进程邮箱（process mail box）这样的设施进行通信。
Go语言推荐采用“ Erlang风格的并发模型”的编程范式，尽管传统的“共享内存模型”仍然被保留，允许适度地使用。
在 Go语言中内置了消息队列的支持，只不过它叫通道（ channel）。
两个goroutine之间可以通过通道来进行交互。
软件工程 单机时代的语言可以只关心问题本身的解决，但是随着工程规模的不断扩大，软件复杂度的不断增加，软件工程也成为语言设计层面要考虑的重要课题。
多数软件需要一个团队共同去完成，在团队协作的过程中，人们需要建立统一的交互语言来降低沟通的成本。
规范化体现在多个层面，如： ·代码风格规范 ·错误处理规范 ·包管理 ·契约规范（接口） ·单元测试规范 ·功能开发的流程规范
Go语言很可能是第一个将代码风格强制统一的语言，例如 Go语言要求 public的变量必须以大写字母开头，private变量则以小写字母开头，这种做法不仅免除了public、private关键字，更重要的是统一了命名风格。
另外，Go语言对 { }应该怎么写进行了强制，比如以下风格是正确的： if expression { ... } 但下面这个写法就是错误的： if expression { ... } 而C和Java语言中则对花括号的位置没有任何要求。

哪种更有利，这个见仁见智。
但很显然的是，所有的 Go代码的花括号位置肯定是非常统一的。
最有意思的其实还是 Go语言首创的错误处理规范： f, err ：= os.Open(filename) if err ！= nil { log.Println(“”Open file failed：“”, err) return } defer f.Close() ... // 操作已经打开的 f文件 这里有两个关键点。
其一是 defer关键字。
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。
在上面的例子中，正是因为有了 defer，才使得无论后续是否会出现异常，都可以确保文件被正确关闭。
其二是 Go语言的函数允许返回多个值。
大多数函数的最后一个返回值会为 error类型，以在错误情况下返回详细信息。
error类型只是一个系统内置的interface，如下： type error interface { Error() string } 有了error类型，程序出现错误的逻辑看起来就相当统一。
在Java中，你可能这样写代码来保证资源正确释放： Connection conn = ...； try { Statement stmt = ...； try { ResultSet rset = ...； try { ... // 正常代码 } finally { rset.close()； } } finally { stmt.close()； } } finally { conn.close()； } 完成同样的功能，相应的 Go代码只需要写成这样： conn ：= ... defer conn.Close() stmt ：= ... defer stmt.Close() rset ：= ... defer rset.Close() ... // 正常代码 对比两段代码，Go语言处理错误的优势显而易见。
当然，其实 Go语言带给我们的惊喜还有很多，后续有机会我们可以就某个更具体的话题详细展开来谈一谈。
编程哲学 计算机软件经历了数十年的发展，形成了多种学术流派，有面向过程编程、面向对象编程、函数式编程、面向消息编程等，这些思想究竟孰优孰劣，众说纷纭。
C语言是纯过程式的，这和它产生的历史背景有关。
Java语言则是激进的面向对象主义推崇者，典型表现是它不能容忍体系里存在孤立的函数。
而 Go语言没有去否认任何一方，而是用批判吸收的眼光，将所有编程思想做了一次梳理，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。
从编程范式的角度来说，Go语言是变革派，而不是改良派。
对于C++、Java和C#等语言为代表的面向对象（OO）思想体系，Go语言总体来说持保守态度，有限吸收。
首先，Go语言反对函数和操作符重载（overload），而C++、Java和C#都允许出现同名函数或操作符，只要它们的参数列表不同。
虽然重载解决了一小部分面向对象编程（OOP）的问题，但 同样给这些语言带来了极大的负担。
而 Go语言有着完全不同的设计哲学，既然函数重载带来了负担，并且这个特性并不对解决任何问题有显著的价值，那么 Go就不提供它。
其次，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（virtual function）和虚函数重载。
确切地说，Go也提供了继承，只不过是采用了组合的文法来提供： type Foo struct { Base ... } func (foo *Foo) Bar() { ... } 再次，Go语言也放弃了构造函数（constructor）和析构函数（destructor）。
由于Go语言中没有虚函数，也就没有 vptr，支持构造函数和析构函数就没有太大的价值。
本着“如果一个特性并不对解决任何问题有显著的价值，那么 Go就不提供它”的原则，构造函数和析构函数就这样被Go语言的作者们干掉了。
在放弃了大量的 OOP特性后，Go语言送上了一份非常棒的礼物：接口（interface）。
你可能会说，除了 C这么原始的语言外，还有什么语言没有接口呢？
是的，多数语言都提供接口，但它们的接口都不同于 Go语言的接口。
Go语言中的接口与其他语言最大的一点区别是它的非侵入性。
在 C++、Java和C#中，为了实现一个接口，你需要从该接口继承，具体代码如下： class Foo

implements IFoo { // Java文法 ... } class Foo ： public IFoo { // C++文法 ... } IFoo* foo = new Foo ；
在Go语言中，实现类的时候无需从接口派生，具体代码如下： type Foo struct { // Go 文法 ... }
var foo IFoo = new(Foo) 只要Foo实现了接口IFoo要求的所有方法，就实现了该接口，可以进行赋值。
Go语言的非侵入式接口，看似只是做了很小的文法调整，实则影响深远。
其一，Go语言的标准库再也不需要绘制类库的继承树图。
你只需要知道这个类实现了哪些 方法，每个方法是啥含义就足够了。
其二，不用再纠结接口需要拆得多细才合理，比如我们实现了 File类，它有下面这些方法： 前言 ：为什么我们需要一门新语言 Read(buf []byte) (n int， err error) Write(buf []byte) (n int， err error) Seek(off int64， whence int) (pos int64， err error) Close() error 那么，到底是应该定义一个 IFile接口，还是应该定义一系列的 IReader、IWriter、 ISeeker和ICloser接口，然后让File从它们派生好呢？
事实上，脱离了实际的用户场景，讨论这两个设计哪个更好并无意义。
问题在于，实现 File类的时候，我怎么知道外部会如何用它呢？
其三，不用为了实现一个接口而专门导入一个包，而目的仅仅是引用其中的某个接口的定义。
在Go语言中，只要两个接口拥有相同的方法列表，那么它们就是等同的，可以相互赋值，如对于以下两个接口，第一个接口： package one type ReadWriter interface { Read(buf [] byte) (n int， err error) Write(buf [] byte) (n int， err error) } 第二个接口： package two type IStream interface { Write(buf [] byte) (n int， err error) Read(buf [] byte) (n int， err error) } 这里我们定义了两个接口，一个叫 one.ReadWriter，一个叫 two.IStream，两者都定义了Read()和Write()方法，只是定义的次序相反。
one.ReadWriter先定义了 Read()再定义 Write()，而two.IStream反之。
在Go语言中，这两个接口实际上并无区别，因为： ·任何实现了 one.ReadWriter接口的类，均实现了 two.IStream； ·任何one.ReadWriter接口对象可赋值给 two.IStream，反之亦然； ·在任何地方使用 one.ReadWriter接口，与使用 two.IStream并无差异。
所以在Go语言中，为了引用另一个包中的接口而导入这个包的做法是不被推荐的。
因为多引用一个外部的包，就意味着更多的耦合。
除了OOP外，近年出现了一些小众的编程哲学， Go语言对这些思想亦有所吸收。
例如， Go语言接受了函数式编程的一些想法，支持匿名函数与闭包。
再如， Go语言接受了以 Erlang语言为代表的面向消息编程思想，支持 goroutine和通道，并推荐使用消息而不是共享内存来进行并发编程。
总体来说， Go语言是一个非常现代化的语言，精小但非常强大。
小结 在十余年的技术生涯中，我接触过、使用过、喜爱过不同的编程语言，但总体而言，Go语言的出现是最让我兴奋的事情。
我个人对未来 10年编程语言排行榜的趋势判断如下： □·Java语言的份额继续下滑，并最终被 C和Go语言超越； □·C语言将长居编程榜第二的位置，并有望在 Go取代Java前重获语言榜第一的宝座； □·Go语言最终会取代 Java，居于编程榜之首。
由七牛云存储团队编著的这本书将尽可能展现出 Go语言的迷人魅力。
希望本书能够让更多人理解这门语言，热爱这门语言，让这门优秀的语言能够落到实处，把程序员从以往繁杂的语言细节中解放出来，集中精力开发更加优秀的系统软件。
许式伟 2012年3月7日

<<Go语言编程>>

内容概要

在C语言和Unix操作系统发布40年后，肯·汤普森等贝尔实验室原班人马终于推出了一门全新的编程语言，它就是Go语言。
Go语言凝聚了该团队将近半个世纪对计算机工程的思考成果，被称为互联网时代的C语言。
自Go语言第一次发布以来，七牛云存储团队就非常密切地关注这门语言的发展，并率先在七牛的产品中进行大面积的应用，而开发效率和系统稳定性等客观数据也在持续证明我们选择Go语言的正确性。
因此，我们迫不及待地希望向同行们分享这门语言，大家一起来享受Go语言所带来的极大乐趣，也一起来促进这门语言的发展吧！

《Go语言编程》首先概览了Go语言的诞生和发展历程，从面向过程编程特性入手介绍Go语言的基础用法，让有一定C语言基础的读者可以非常迅速地入门并开始上手用Go语言来解决实际问题，之后介绍了Go语言简洁却又无比强大的面向对象编程特性和并发编程能力，至此读者已经可以理解为什么Go语言是为互联网时代而生的语言。
从实用性角度出发，本书还介绍了Go语言标准库和配套工具的用法，包括安全编程、网络编程、工程管理工具等。
对于希望对Go语言有更深入了解的读者，我们也特别组织了一系列进阶话题，包括语言交互性、链接符号、goroutine机理和接口机制等。
《Go语言编程》适合所有层次的开发者阅读。

作者简介

许式伟，七牛云存储CEO，曾任盛大创新院资深研究员、金山软件技术总监、WPS Office 2005首席架构师。
开源爱好者，发布过WINX、TPL等十余个C++开源项目，拥有超过15年的C/C++开发经验。
在接触Go语言后即被其大道至简、少即是多的设计哲学所倾倒。
七牛云存储是国内第一个吃螃蟹的团队，核心服务完全采用Go语言实现。
吕桂华，七牛云存储联合创始人，曾在金山软件、盛大游戏等公司担任架构师和部门经理等职务，在企业级系统和大型网游平台领域有较多涉猎。
拥有十余年的C/C++大型项目开发经验，也曾在Java和.NET平台上探索多年。
同样被Go语言的魅力所吸引而不可自拔，希望能为推广这门优秀的语言尽自己的绵薄之力。

<<Go语言编程>>

书籍目录

目 录 第1章 初识Go语言 11.1 语言简史 11.2 语言特性 21.2.1 自动垃圾回收 31.2.2 更丰富的内置类型 41.2.3 函数多返回值 51.2.4 错误处理 61.2.5 匿名函数和闭包 61.2.6 类型和接口 71.2.7 并发编程 81.2.8 反射 91.2.9 语言交互性 101.3 第一个Go程序 111.3.1 代码解读 111.3.2 编译环境准备 121.3.3 编译程序 121.4 开发工具选择 131.5 工程管理 131.6 问题追踪和调试 181.6.1 打印日志 181.6.2 GDB调试 181.7 如何寻求帮助 181.7.1 邮件列表 191.7.2 网站资源 191.8 小结 19第2章 顺序编程 202.1 变量 202.1.1 变量声明 202.1.2 变量初始化 212.1.3 变量赋值 212.1.4 匿名变量 222.2 常量 222.2.1 字面常量 222.2.2 常量定义 232.2.3 预定义常量 232.2.4 枚举 242.3 类型 242.3.1 布尔类型 252.3.2 整型 252.3.3 浮点型 272.3.4 复数类型 282.3.5 字符串 282.3.6 字符类型 302.3.7 数组 312.3.8 数组切片 322.3.9 map 362.4 流程控制 382.4.1 条件语句 382.4.2 选择语句 392.4.3 循环语句 402.4.4 跳转语句 412.5 函数 412.5.1 函数定义 422.5.2 函数调用 422.5.3 不定参数 432.5.4 多返回值 452.5.5 匿名函数与闭包 452.6 错误处理 472.6.1 error接口 472.6.2 defer 482.6.3 panic()和recover() 492.7 完整示例 502.7.1 程序结构 512.7.2 主程序 512.7.3 算法实现 542.7.4 主程序 572.7.5 构建与执行 592.8 小结 61第3章 面向对象编程 623.1 类型系统 623.1.1 为类型添加方法 633.1.2 值语义和引用语义 663.1.3 结构体 673.2 初始化 683.3 匿名组合 683.4 可见性 713.5 接口 713.5.1 其他语言的接口 713.5.2 非侵入式接口 733.5.3 接口赋值 743.5.4 接口查询 763.5.5 类型查询 783.5.6 接口组合 783.5.7 Any类型 793.6 完整示例 793.6.1 音乐库 803.6.2 音乐播放 823.6.3 主程序 843.6.4 构建运行 863.6.5 遗留问题 863.7 小结 87第4章 并发编程 884.1 并发基础 884.2 协程 904.3 goroutine 904.4 并发通信 914.5 channel 944.5.1 基本语法 954.5.2 select 954.5.3 缓冲机制 964.5.4 超时机制 974.5.5 channel的传递 984.5.6 单向channel 984.5.7 关闭channel 994.6 多核并行化 1004.7 出让时间片 1014.8 同步 1014.8.1 同步锁 1014.8.2 全局唯一性操作 1024.9 完整示例 1034.9.1 简单IPC框架 1054.9.2 中央服务器 1084.9.3 主程序 1134.9.4 运行程序 1164.10 小结 117第5章 网络编程 1185.1 Socket编程 1185.1.1 Dial()函数 1185.1.2 ICMP示例程序 1195.1.3 TCP示例程序 1215.1.4 更丰富的网络通信 1225.2 HTTP编程 1245.2.1 HTTP客户端 1245.2.2 HTTP服务端 1305.3 RPC编程 1325.3.1 Go语言中的RPC支持与处理 1325.3.2 Gob简介 1345.3.3 设计优雅的RPC接口 1345.4 JSON处理 1355.4.1 编码为JSON格式 1365.4.2 解码JSON数据 1375.4.3 解码未知结构的JSON数据 1385.4.4 JSON的流式读写 1405.5 网站开发 1405.5.1 最简单的网站程序 1415.5.2 net/http包简介 1415.5.3 开发一个简单的相册网站 1425.6 小结 157第6章 安全编程 1586.1 数据加密 1586.2 数字签名 1586.3 数字证书 1596.4 PKI体系 1596.5 Go语言的哈希函数 1596.6 加密通信 1606.6.1 加密通信流程 1616.6.2 支持HTTPS的Web服务器 1626.6.3 支持HTTPS的文件服务器 1656.6.4 基于SSL/TLS的ECHO程序 1666.7 小结 169第7章 工程管理 1707.1 Go命令行工具 1707.2 代码风格 1727.2.1 强制性编码规范 1727.2.2 非强制性编码风格建议 1737.3 远程import支持 1757.4 工程组织 1757.4.1 GOPATH 1767.4.2 目录结构 1767.5 文档管理 1777.6 工程构建 1807.7 跨平台开发 1807.7.1 交叉编译 1817.7.2 Android支持 1827.8 单元测试 1837.9 打包分发 1847.10 小结 184第8章 开发工具 1868.1 选择开发工具 1868.2 gedit 1878.2.1 语法高亮 1878.2.2 编译环境 1878.3 Vim 1888.4 Eclipse 1898.5 Notepad++ 1928.5.1 语法高亮 1928.5.2 编译环境 1928.6 LiteIDE 1938.7 小结 195第9章 进阶话题 1969.1 反射 1969.1.1 基本概念 1969.1.2 基本用法 1979.1.3 对结构的反射操作 1999.2 语言交互性 1999.2.1 类型映射 2009.2.2 字符串映射 2019.2.3 C程序 2019.2.4 函数调用 2029.2.5 编译Cgo 2039.3 链接符号 2039.4 goroutine机理 2049.4.1 协程 2049.4.2 协程的C语言实现 2059.4.3 协程库概述 2059.4.4 任务 2089.4.5 任务调度 2109.4.6 上下文切换 2119.4.7 通信机制 2159.5 接口机理 2169.5.1 类型赋值给接口 2179.5.2 接口查询 2239.5.3 接口赋值 224附录A 225

媒体关注与评论

这本书除了完整介绍Go语言特性之外，还深入剖析了语言实现机制。
作为服务器软件开发者和编程语言爱好者，我强烈推荐此书。
——李杰，盛大文学首席架构师《Go语言编程》正是这样一份猛料，能够带领越来越多的人了解Go，学习Go，用Go来实现自己的梦想。
——何晓杰，国内知名Android研究者，安居客移动事业部高级开发经理《Go语言编程》这本书应当说是作者多年编程经验的沉淀和反思。
本书一方面通过展示和分析大量Go语言代码，阐明了Go语言基本的使用方式，另一方面通过和C语言代码进行比较，进一步剖析了语言的内在设计思想，乃至底层实现原理，让各个层次的读者都能从书中汲取到大量的知识，使人读后必有所得。
——邢星，Go语言社区积极推动者，39健康网技术部副总监

<<Go语言编程>>

编辑推荐

国内第一本Go语言编程书原盛大创新院研究员执笔Go语言领域技术大牛作品作者Go语言开发实战项目帮助您快速入手全新语言

版权说明