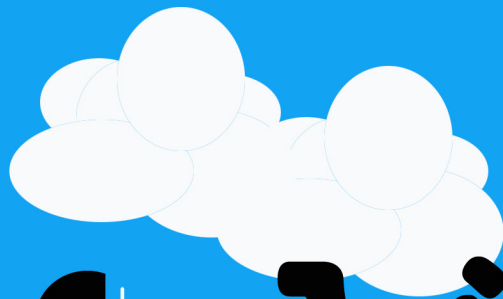




图灵社区 [www.ituring.com.cn](http://www.ituring.com.cn)  
图灵电子书

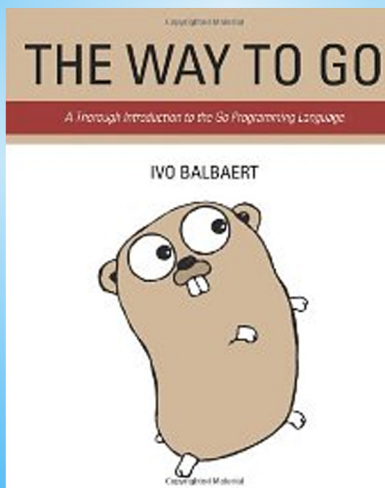


# Go 入门指南

——The way to go (中文版)



【比利时】Ivo Balbaert 著  
陈佳桦 译



技术改变世界  
阅读塑造人生

# 版权声明

本书由北京图灵文化发展有限公司全球范围内出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

版权所有，侵权必究。

书 名 《Go 入门指南——The way to go（中文版）》

著 【比利时】Ivo Balbaert

译 陈佳桦

封面设计 董苗苗

版本信息 2013 年

定 价 0.00 元

---

阅读、书评、交流、勘误、意见或建议，欢迎您来到图灵社区！

<http://www.ituring.com.cn/>

反侵权热线 （010）51095186



# 译者序

在接触 Go 语言之后，对这门编程语言非常着迷，期间也陆陆续续开始一些帮助国内编程爱好者了解和发展 Go 语言的工作，比如开始录制视频教程《Go 编程基础》。但由于目前国内并没有比较好的 Go 语言书籍，而国外的优秀书籍因为英文的缘故在一定程度上也为不少 Go 语言爱好者带来了一些学习上的困扰，不仅为了加快扩散 Go 爱好者的国内群体，同时充分贯彻 Asta 谢 的为己为人精神，本人在完成阅读这本名叫《The Way to Go》之后，决定每天抽出一一点时间来进行翻译的工作，并且以开源的形式免费分享给有需要的 Go 语言爱好者。

尽管该书对目前 Go 语言版本来说有小部分内容相对过时，但是为当下不可多得的好书，部分内容已获得作者同意根据当前 Go 语言版本进行修改。

该翻译版本已获得原作者（Ivo Balbaert）本人授权，并表示支持开源事业的发展！

本书全面地讲解了 Go 学习者需要了解的知识，不仅涵盖了基础概念，同时也包含一些项目开发中所涉及到的高级技巧。每个知识点的讲解和示例都非常完备，部分章节在最后还会提供练习题以便学习者巩固及加深印象，是 Go 学习者不可错过的一本经典书籍。

# 前言

**用更少的代码，更短的编译时间，创建运行更快的程序，享受更多的乐趣**

对于学习 Go 编程语言的爱好者来说，这本书无疑是最适合你的一本书籍，这里包含了当前最全面的学习资源。本书通过对官方的在线文档、名人博客、书籍、相关文章以及演讲的资料收集和整理，并结合我自身在软件工程、编程语言和数据库开发的授课经验，将这些零碎的知识点组织成系统化的概念和技术分类来进行讲解。

随着软件规模的不断扩大，诸多的学者和谷歌的开发者们在公司内部的软件开发过程中开始经历大量的挫折，在诸多问题上都不能给出令人满意的解决方案，尤其是在使用 C++ 来开发大型的服务端软件时，情况更是不容乐观。由于二进制文件一般都是非常巨大的，因此需要耗费大量的时间在编译这些文件上，同时编程语言的设计思想也已经非常陈旧，这些情况都充分证明了现有的编程语言已不符合时下的生产环境。尽管硬件在过去的几十年中有了飞速的发展，但人们依旧没有找到机会去改变 C++ 在软件开发的重要地位，并在实际开发过程中忍受着它所带来的令人头疼的一些问题。因此学者们坐下来总结出了现在生产环境与软件开发之间的主要矛盾，并尝试设计一门全新的编程语言来解决这些问题。

以下就是他们讨论得出的对编程语言的设计要求：

- 能够以更快的速度开发软件
- 开发出的软件能够很好地在现代的多核计算机上工作
- 开发出的软件能够很好地在网络环境下工作
- 使人们能够享受软件开发的过程

Go 语言就在这样的环境下诞生了，它让人感觉像是 Python 或 Ruby 这样的动态语言，但却又拥有像 C 或者 Java 这类语言的高性能和安全性。

Go 语言出现的目的是希望在编程领域创造最实用的方式来进行软件开发。它并不是要用奇怪的语法和晦涩难懂的概念来从根本上推翻已有的编程语言，而是建立并改善了 C、Java、C# 中的许多语法风格。它提倡通过接口来针对面向对象编程，通过 goroutine 和 channel 来支持并发和并行编程。

这本书是为那些想要学习 Go 这门全新的，迷人的和充满希望的编程语言的开发者量身定做的。当然，你在学习 Go 语言之前需要具备一些关于编程的基础知识和经验，并且拥有合适的学习环境，但你并不需要对 C 或者 Java 或其它类似的语言有非常深入的了解。

对于那些熟悉 C 或者面向对象编程语言的开发者，我们将会在本书中用 Go 和一些编程语言的相关概念进行比较（书中会使用大家所熟知的缩写“OO”来表示面向对象）。

本书将会从最基础的概念讲起，同时也会讨论一些类似在应用 goroutine 和 channel 时有多少种不同的模式，如何在 Go 语言中使用谷歌 API，如何操作内存，如何在 Go 语言中进行程序测试和如何使用模板来开发 Web 应用这些高级概念和技巧。

在本书的第一部分，我们将会讨论 Go 语言的起源（第 1 章），以及如何安装 Go 语言（第 2 章）和开发环境（第 3 章）。

在本书的第二部分，我们将会带领你贯穿 Go 语言的核心思想，譬如简单与复杂类型（第 4，7，8 章），控制结构（第 5 章），函数（第 6 章），结构与方法（第 10 章）和接口（第 11 章）。我们会对 Go 语言的函数式和面向对象编程进行透彻的讲解，包括如何使用 Go 语言来构造大型项目（第 9 章）。

在本书的第三部分，你将会学习到如何处理不同格式的文件（第 12 章）和如何在 Go 语言中巧妙地使用错误处理机制（第 13 章）。然后我们会对 Go 语言中最值得称赞的设计 goroutine 和 channel 进行并发和多核应用的基本技巧的讲解（第 14 章）。最后，我们会讨论如何将 Go 语言应用到分布式和 Web 应用中的相关网络技巧（第 15 章）。

我们会在本书的第四部分向你展示许多 Go 语言的开发模式和一些编码规范，以及一些非常有用的代码片段（第 18 章）。在前面章节完成对所有的 Go 语言技巧的学习之后，你将会学习如何构造一个完整 Go 语言项目（第 19 章），然后我们会介绍一些关于 Go 语言在云（Google App Engine）方面的应用（第 20 章）。在本书的最后一章（第 21 章），我们会讨论一些在全世界范围内已经将 Go 语言投入实际开发的公司和组织。本书将会在最后给出一些对 Go 语言爱好者的引用，Go 相关包和工具的参考，以及章节练习的答案和所有参考资源和文献的清单。

Go 语言有一个被称之为“没有废物”的宗旨，就是将一切没有必要的东西都去掉，不能去掉的就无底线地简化，同时追求最大程度的自动化。他完美地诠释了敏捷编程的 KISS 秘诀：短小精悍！

Go 语言通过改善或去除在 C、C++ 或 Java 中的一些所谓“开放”特性来让开发者们的工作更加便利。这里只举例其中的几个，比如对于变量的默认初始化，内存分配与自动回收，以及更简洁却不失健壮的控制结构。同时我们也会发现 Go 语言旨在减少不必要的编码工作，这使得 Go 语言的代码更加简洁，从而比传统的面向对象语言更容易阅读和理解。

与 C++ 或 Java 这些有着庞大体系的语言相比，Go 语言简洁到可以将它整个的装入你的大脑中，而且比学习 Scala（Java 的并发语言）有更低的门槛，真可谓是 21 世纪的 C 语言！

作为一门系统编程语言，你不应该为 Go 语言的大多数代码示例和练习都和控制台有着密不可分的关系而感到惊奇，因为提供平台依赖性的 GUI（用户界面）

框架并不是一个简单的任务。有许多由第三方发起的 GUI 框架项目正在如火如荼地进行中，或许我们会在不久的将来看到一些可用的 Go 语言 GUI 框架。不过现阶段的 Go 语言已经提供了大量有关 Web 方面的功能，我们可以通过它强大的 http 和 template 包来达到 Web 应用的 GUI 实现。

我们会经常涉及到一些关于 Go 语言的编码规范，了解和使用这些已经被广泛认同的规范应该是你学习阶段最重要的实践。我会在书中尽量使用已经讲解的概念或者技巧来解释相关的代码示例，以避免你在不了解某些高级概念的情况下而感到迷茫。

我们通过 227 个完整的代码示例和书中的解释说明来对所有涉及到的概念和技巧进行彻底的讲解，你可以下载这些代码到你的电脑上运行，从而加深对概念的理解。

本书会尽可能地将前后章节的内容联系起来，当然这也同时要求你通过学习不同的知识来对一个问题提出尽可能多的解决方案。记住，学习一门新语言的最佳方式就是实践，运行它的代码，修改并尝试更多的方案。因此，你绝对不可以忽略书中的 130 个代码练习，这将对你学习好 Go 语言有很大的帮助。比如，我们就为斐波那契算法提供了 13 个不同的版本，而这些版本都使用了不同的概念和技巧。

你可以通过访问本书的[官方网站](#)下载书中的代码，并获得有关本书的勘误情况和内容更新。

为了让你在成为 Go 语言大师的道路上更加顺利，我们会专注于一些特别的章节以提供 Go 语言开发模式的最佳实践，同时也会帮助初学者逃离一些语言的陷阱。第 18 章可以作为你在开发时的一个参考手册，因为当中包含了众多的有价值的代码片段以及相关的解释说明。

最后要说明的是，你可以通过完整的索引来快速定位你需要阅读的章节。书中所有的代码都在 Go1 版本下测试通过。（译者注：所有代码经作者同意将会根据需要进行相关修改以在 Go1.1 版本下运行）

这里有一段来自在 C++、Java 和 Python 领域众所周知的专家 Bruce Eckel 的评论：

“作为一个有着 C/C++ 背景的开发者的，我在使用 Go 语言时仿佛呼吸到了新鲜空气一般，令人心旷神怡。我认为使用 Go 语言进行系统编程开发比使用 C++ 有着更显著的优势，因为它在解决一些很难用 C++ 解决的问题的同时，让我的工作变得更加高效。我并不是说 C++ 的存在是一个错误，相反地，我认为这是历史发展的必然结果。当我深陷在 C 语言这门略微比汇编语言好一点的泥潭时，我坚信任何语言的构造都不可能支持大型项目的开发。像垃圾回收或并发语言支持这类东西，在当时都是极其荒谬的主意，根本没有人会在乎。C++ 向大型项目开发迈出了重要的第一步，带领我们走进这个广袤无垠的世界。很庆幸



Stroustrup 做了让 C++ 兼容 C 语言以能够让其编译 C 程序这个正确的决定。我们当时需要 C++ 的出现。”

“之后我们学到了更多。我们毫无疑问地接受了垃圾回收，异常处理和虚拟机这些当年人们认为只有疯子才会想的东西。C++ 的复杂程度（新版的 C++ 甚至更加复杂）极大地影响了软件开发的高效性，这使得它再也不再适合这个时代。人们不再像过往那样认同在 C++ 中兼容使用 C 语言的方法，认为这些工作只是在浪费时间，牺牲人们的努力。就在此时，Go 语言已经成功地解决了 C++ 中那些本打算解决却未能解决的关键问题。”

我非常想要向发明这门精湛的语言的 Go 开发团队表示真挚的感谢，尤其是团队的领导者 Rob Pike、Russ Cox 和 Andrew Gerrand，他们阐述的例子和说明都非常的完美。同时，我还要感谢 Miek Gieben、Frank Muller、Ryenne Dolan 和 Satish V.J. 给予我巨大的帮助，还有那些 Golang-nuts 邮件列表里的所有的成员。

欢迎来到 Go 语言开发的奇妙世界！

版权声明.....	1 -
译者序.....	2 -
前言.....	3 -
1 Go 语言的起源，发展与普及.....	8 -
1.1 起源与发展.....	8 -
1.2 语言的主要特性与发展的环境和影响因素.....	10 -
1.2.2 为什么要创造一门编程语言.....	11 -
1.2.3 Go 语言的发展目标.....	12 -
1.2.4 指导设计原则.....	13 -
1.2.5 语言的特性.....	13 -
1.2.6 语言的用途.....	14 -
1.2.7 关于特性丢失.....	15 -
1.2.8 使用 Go 语言编程.....	15 -
1.2.9 小结.....	16 -
2 安装与运行环境.....	17 -
2.1 平台与架构.....	17 -
2.2 Go 环境变量.....	19 -
2.3 在 Linux 上安装 Go.....	20 -
2.4 在 Mac OS X 上安装 Go.....	24 -
2.5 在 Windows 上安装 Go.....	24 -
2.6 安装目录清单.....	25 -
2.7 Go 类虚拟机 (runtime).....	26 -
2.8 Go 解释器.....	26 -
3 编辑器、集成开发环境与其它工具.....	27 -
3.1 Go 开发环境的基本要求.....	27 -
3.2 编辑器和集成开发环境.....	28 -
3.2.1 Golang LiteIDE.....	31 -
3.2.2 GoClipse.....	32 -
3.3 调试器.....	33 -
3.4 构建并运行 Go 程序.....	33 -
3.5 格式化代码.....	33 -
3.6 生成代码文档.....	34 -
3.7 其它工具.....	35 -
3.8 Go 性能说明.....	35 -
3.9 与其它语言进行交互.....	37 -
3.9.1 与 C 进行交互.....	37 -
3.9.2 与 C++ 进行交互.....	38 -



# 1 Go 语言的起源，发展与普及

本章主要介绍 Go 语言的开发缘由，过程以及现状。

## 1.1 起源与发展

Go 语言起源 2007 年，并于 2009 年正式对外发布。它从 2009 年 9 月 21 日开始作为谷歌公司 20% 兼职项目，即相关员工利用 20% 的空余时间来参与 Go 语言的研发工作。该项目的三位领导者均是著名的 IT 工程师：Robert Griesemer，参与开发 Java HotSpot 虚拟机；Rob Pike，Go 语言项目总负责人，贝尔实验室 Unix 团队成员，参与的项目包括 Plan 9，Inferno 操作系统和 Limbo 编程语言；Ken Thompson，贝尔实验室 Unix 团队成员，C 语言、Unix 和 Plan 9 的创始人之一，与 Rob Pike 共同开发了 UTF-8 字符集规范。自 2008 年 1 月起，Ken Thompson 就开始研发一款以 C 语言为目标结果的编译器来拓展 Go 语言的设计思想。

这是一个由计算机领域“发明之父”所组成的黄金团队，他们对系统编程语言，操作系统和并行都有着非常深刻的见解



图 1.1 Go 语言设计者：Griesemer、Thompson 和 Pike

在 2008 年年中，Go 语言的设计工作接近尾声，一些员工开始以全职工作状态投入到这个项目的编译器和运行实现上。Ian Lance Taylor 也加入到了开发团队中，并于 2008 年 5 月创建了一个 gcc 前端。

Russ Cox 加入开发团队后着手语言和类库方面的开发，也就是 Go 语言的标准包。在 2009 年 10 月 30 日，Rob Pike 以 Google Techtalk 的形式第一次向人们宣告了 Go 语言的存在。

直到 2009 年 11 月 10 日，开发团队将 Go 语言项目以 BSD-style 授权（完全开源）正式公布在 Linux 和 Mac OS X 平台上的版本。Hector Chu 于同年 11 月 22 日公布了 Windows 版本。

作为一个开源项目，Go 语言借助开源社区的有生力量达到快速地发展，并吸引更多的开发者来使用并改善它。自该开源项目发布以来，超过 200 名非谷歌员工的贡献者对 Go 语言核心部分提交了超过 1000 个修改建议。在过去的 18 个月里，又有 150 开发者贡献了新的核心代码。这俨然形成了世界上最大的开源团队，并使该项目跻身 [Ohloh](#) 前 2% 的行列。大约在 2011 年 4 月 10 日，谷歌开始抽调员工进入全职开发 Go 语言项目。开源化的语言显然能够让更多的开发者参与其中并加速它的发展速度。Andrew Gerrand 在 2010 年加入到开发团队中成为共同开发者与支持者。

在 Go 语言在 2010 年 1 月 8 日被 [Tiobe](#)（闻名于它的编程语言流行程度排名）宣布为“2009 年年度语言”后，引起各界很大的反响。目前 Go 语言在这项排名中的最高记录是在 2010 年 2 月创下的第 13 名，流行程度 1778%。

#### 时间轴：

- 2007 年 9 月 21 日：雏形设计
- 2009 年 11 月 10 日：首次公开发布
- 2010 年 1 月 8 日：当选 2009 年年度语言
- 2010 年 5 月：谷歌投入使用
- 2011 年 5 月 5 日：Google App Engine 支持 Go 语言

从 2010 年 5 月起，谷歌开始将 Go 语言投入到后端基础设施的实际开发中，例如开发用于管理后端复杂环境的项目。有句话叫“吃你自己的狗粮”，这也体现了谷歌确实想要投资这门语言，并认为它是有生产价值的。

Go 语言的官方网站是 <https://golang.org>，这个站点采用 Python 作为前端，并且使用 Go 语言自带的工具 godoc 运行在 Google App Engine 上来作为 Web 服务器提供文本内容。在官网的首页有一个功能叫做 Go-playground，是一个 Go 代码的简单编辑器的沙盒，它可以在没有安装 Go 语言的情况下在你的浏览器中编译并运行 Go，它提供了一些示例，其中包括国际惯例“Hello, World!”。

更多的信息详见 <http://code.google.com/p/go/>，Go 项目 Bug 追踪和功能预期详见 <http://code.google.com/p/go/issues/list>。

Go 通过以下的 Logo 来展示它的速度，并以囊地鼠（Gopher）作为它的吉祥物。

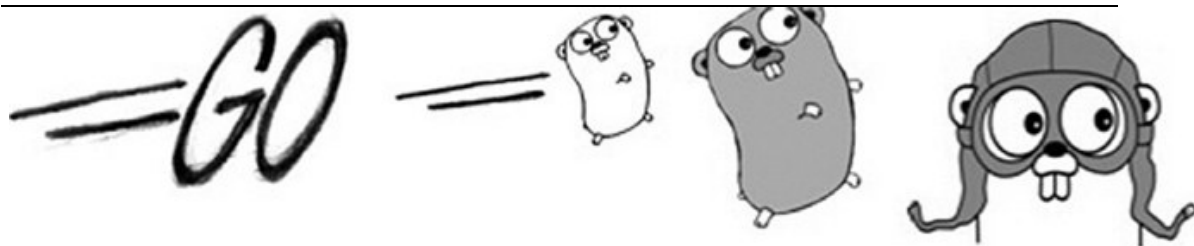


图 1.2 Go 语言 Logo

谷歌邮件列表 [golang-nuts](mailto:golang-nuts) 非常活跃，每天的讨论和问题解答数以百计。

关于 Go 语言在 Google App Engine 的应用，这里有一个单独的邮件列表 [google-appengine-go](mailto:google-appengine-go)，不过 2 个邮件列表的讨论内容并不是分得很清楚，都会涉及到相关的话题。[go-lang.cat-v.org/](http://go-lang.cat-v.org/) 是 Go 语言开发社区的资源站，[#go-nuts](http://irc.freenode.net) 是官方的 Go IRC 频道。

[http://twitter.com/#!/go\\_nuts](http://twitter.com/#!/go_nuts) 是 Go 语言在 Twitter 的官方帐号，大家一般使用 #golang 作为话题标签。

这里还有一个在 Linked-in 的小组：

[http://www.linkedin.com/groups?gid=2524765&trk=myg\\_ugrp\\_ovr](http://www.linkedin.com/groups?gid=2524765&trk=myg_ugrp_ovr)。

Go 编程语言的维基百科：

[http://en.wikipedia.org/wiki/Go\\_\(programming\\_language\)](http://en.wikipedia.org/wiki/Go_(programming_language))

Go 语言相关资源的搜索引擎页面：<http://go-lang.cat-v.org/go-search>

Go 语言还有一个运行在 Google App Engine 上的 [Go Tour](#)，你也可以通过执行命令 `go install go-tour.googlecode.com/hg/gotour` 安装到你的本地机器上。对于中文读者，可以访问该指南的[中文版本](#)，或通过命令 `go install https://bitbucket.org/mikespook/go-tour-zh/gotour` 进行安装。

## 1.2 语言的主要特性与发展的环境和影响因素

正如“21 世纪的 C 语言”这句话所说，Go 语言并不是凭空而造的，而是和 C++、Java 和 C# 一样属于 C 系。不仅如此，设计者们还汲取了其它编程语言的精粹部分融入到 Go 语言当中。

在声明和包的设计方面，Go 语言受到 Pascal、Modula 和 Oberon 系语言的影响；在并发原理的设计上，Go 语言从同样受到 Tony Hoare 的 CSP（通信序列进程 *Communicating Sequential Processes*）理论影响的 Limbo 和 Newsqueak 的实践中借鉴了一些经验，并使用了和 Erlang 类似的机制。

这是一门完全开源的编程语言，因为它使用 BSD 授权许可，所以任何人都可以进行商业软件的开发而不需要支付任何费用。

尽管为了能够让目前主流的开发者们能够对 Go 语言中的类 C 语言的语法感到非常亲切而易于转型，但是它在极大程度上简化了这些语法，使得它们比 C/C++ 的语法更加简洁和干净。同时，Go 语言也拥有一些动态语言的特性，这使得使用 Python 和 Ruby 的开发者们在使用 Go 语言的时候感觉非常容易上手。

下图展示了一些其它编程语言对 Go 语言的影响：

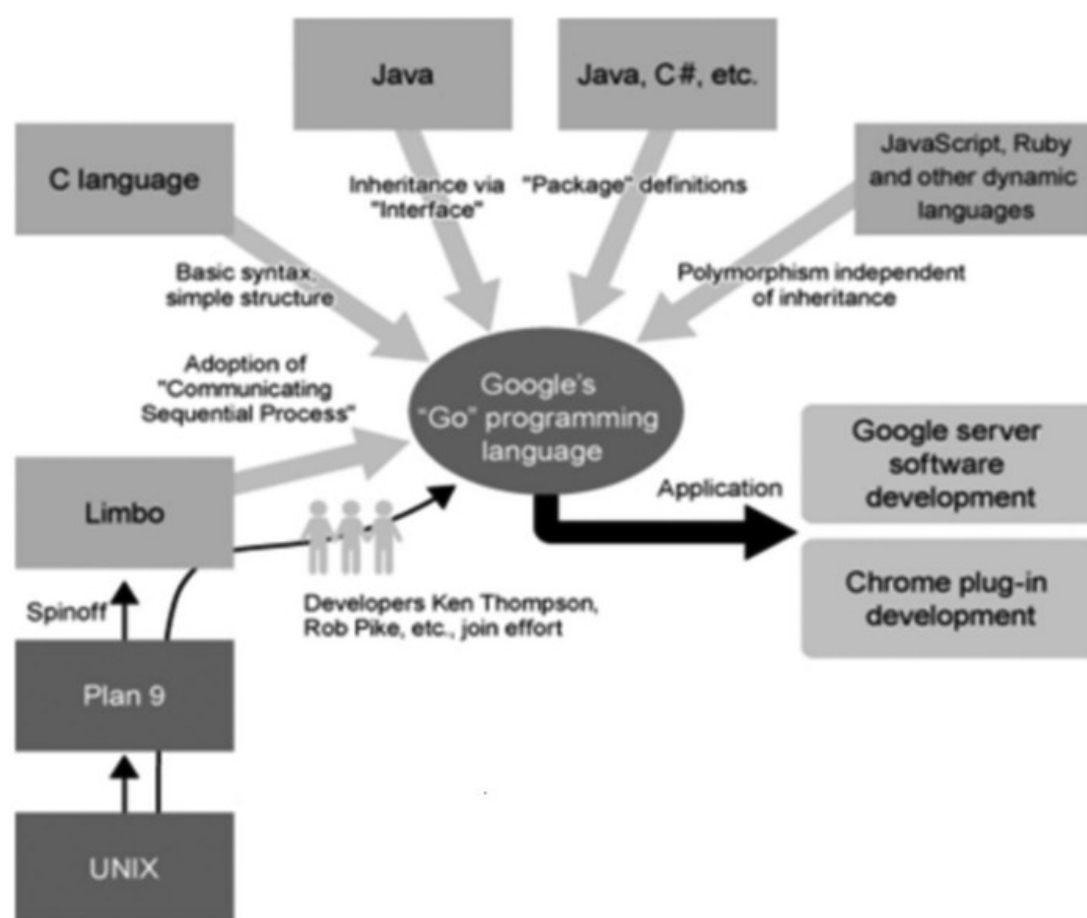


图 1.3 其它编程语言对 Go 语言的影响

### 1.2.2 为什么要创造一门编程语言

- C/C++的发展速度无法跟上计算机发展的脚步，十多年来也没有出现一门与时代相符的主流系统编程语言，因此人们需要一门新的系统编程语言来弥补这个空缺，尤其是在计算机信息时代。
- 对比计算机性能的提升，软件开发领域不被认为发展地足够快或者比硬件发展更加成功（有许多项目均以失败告终），同时应用程序的体积始终在不断地扩大，这就迫切地需要一门具备更高层次概念的低级语言来突破现状。

- 在 Go 语言出现之前，开发者们总是面临非常艰难的抉择，究竟是使用执行速度快但是编译速度并不理想的语言（如：C++），还是使用编译速度较快但执行效率不佳的语言（如：.NET、Java），或者说开发难度较低但执行速度一般的动态语言呢？显然，Go 语言在这 3 个条件之间做到了最佳的平衡：快速编译，高效执行，易于开发。

### 1.2.3 Go 语言的发展目标

Go 语言的主要目标是将静态语言的安全性和高效性与动态语言的易开发性进行有机结合，达到完美平衡，从而使编程变得更加有趣，而不是在艰难抉择中痛苦前行。

因此，Go 语言是一门类型安全和内存安全的编程语言。虽然 Go 语言中仍有指针的存在，但并不允许进行指针运算。

Go 语言的另一个目标是对于网络通信，并发和并行编程的极佳支持，从而更好地利用大量的分布式和多核的计算机，这一点对于谷歌内部的使用来说就非常重要了。设计者通过 goroutine 这种轻量级线程的概念来实现这个目标，然后通过 channel 来实现各个 goroutine 之间的通信。他们实现了分段栈增长和 goroutine 在线程基础上多路复用技术的自动化。

这个特性显然是 Go 语言最强有力的部分，不仅支持了日益重要的多核与多处理器计算机，也弥补了现存编程语言在这方面所存在的不足。

Go 语言中另一个非常重要的特性就是它的构建速度（编译和链接到机器代码的速度），一般情况下构建一个程序的时间只需要数百毫秒到几秒。作为大量使用 C++ 来构建基础设施的谷歌来说，无疑从根本上摆脱了 C++ 在构建速度上非常不理想的噩梦。这不仅极大地提升了开发者的生产力，同时也使得软件开发过程中的代码测试环节更加紧凑，而不必浪费大量的时间在等待程序的构建上。

依赖管理是现今软件开发的一个重要组成部分，但是 C 语言中“头文件”的概念却导致越来越多因为依赖关系而使得构建一个大型的项目需要长达几个小时的时间。人们越来越需要一门具有严格的、简洁的依赖关系分析系统从而能够快速编译的编程语言。这正是 Go 语言采用包模型的根本原因，这个模型通过严格的依赖关系检查机制来加快程序构建的速度，提供了非常好的可量测性。

整个 Go 语言标准库的编译时间一般都在 20 秒以内，其它的常规项目也只需要半秒钟的时间来完成编译工作。这种闪电般的编译速度甚至比编译 C 语言或者 Fortran 更加快，使得编译这一环节不再成为在软件开发中困扰开发人员的问题。在这之前，动态语言将快速编译作为自身的一大亮点，像 C++ 那样的静态语言一般都有非常漫长的编译和链接工作。而同样作为静态语言的 Go 语言，通过自身优良的构建机制，成功地去除了这个弊端，使得程序的构建过程变得微不足道，拥有了像脚本语言和动态语言那样的高效开发的能力。



另外，Go 语言在执行速度方面也可以与 C/C++ 相提并论。

由于内存问题（通常称为内存泄漏）长期以来一直伴随着 C++ 的开发者们，Go 语言的设计者们认为内存管理不应该是开发人员所需要考虑的问题。因此尽管 Go 语言像其它静态语言一样执行本地代码，但它依旧运行在某种意义上的虚拟机，以此来实现高效快速的垃圾回收（使用了一个简单的标记-清除算法）。

尽管垃圾回收并不容易实现，但考虑这将是未来并发应用程序发展的一个重要组成部分，Go 语言的设计者们还是完成了这项艰难的任务。

Go 语言还能够运行时进行反射相关的操作。

使用 `go install` 能够很轻松地对第三方包进行部署。

此外，Go 语言还支持调用由 C 语言编写的海量库文件（第 3.9 节），从而能够将过去开发的软件进行快速迁移。

### 1.2.4 指导设计原则

Go 语言通过减少关键字的数量（25 个）来简化编码过程中的混乱和复杂度。干净、整齐和简洁的语法也能够提高程序的编译速度，因为这些关键字在编译过程中少到甚至不需要符号表来协助解析。

这些方面的工作都是为了减少编码的工作量，甚至可以与 Java 的简化程度相比较。

Go 语言有一种极简抽象艺术家的感觉，因为它只提供了一到两种方法来解决某个问题，这使得开发者们的代码都非常容易阅读和理解。众所周知，代码的可读性是软件工程里最重要的一部分（**译者注：代码是写给人看的，不是写给机器看的**）。

这些设计理念没有建立其它概念之上，所以并不会因为牵扯到一些概念而将某个概念复杂化，他们之间是相互独立的。

Go 语言有一套完整的编码规范，你可以在 [Go 语言编码规范](#) 页面进行查看。

它不像 Ruby 那样通过实现过程来定义编码规范。作为一门具有明确编码规范的语言，它要求可以采用不同的编译器如 `gc` 和 `gccgo`（第 2.1 节）进行编译工作，这对语言本身拥有更好的编码规范起到很大帮助。

[LALR](#) 是 Go 语言的语法标准，你也可以在 `src/cmd/gc/go.y` 中查看到，这种语法标准在编译时不需要符号表来协助解析。

### 1.2.5 语言的特性

Go 语言从本质上（程序和结构方面）来实现并发编程。

因为 Go 语言没有类和继承的概念，所以它和 Java 或 C++ 看起来并不相同。但是它通过接口（interface）的概念来实现多态性。Go 语言有一个清晰易懂的轻量级类型系统，在类型之间也没有层级之说。因此可以说这是一门混合型的语言。

在传统的面向对象语言中，使用面向对象编程技术显得非常的臃肿，它们总是通过复杂的模式来构建庞大的类型层级，这违背了编程语言应该提升生产力的宗旨。

函数是 Go 语言中的基本构件，它们的使用方法非常灵活。在第六章，我们会看到 Go 语言在函数式编程方面的基本概念。

Go 语言使用静态类型，所以它是类型安全的一门语言，加上通过构建到本地代码，程序的执行速度也非常快。

作为强类型语言，隐式的类型转换是不被允许的，记住一条原则：让所有的东西都是显式的。

Go 语言其实也有一些动态语言的特性（通过关键字 `var`），所以它对那些逃离 Java 和 .Net 世界而使用 Python、Ruby、PHP 和 JavaScript 的开发者们也具有很大的吸引力。

Go 语言支持交叉编译，比如说你可以在运行 Linux 系统的计算机上开发运行下 Windows 下运行的应用程序。这是第一门完全支持 UTF-8 的编程语言（译者注：.NET 好像也支持吧？），这不仅体现在它可以处理使用 UTF-8 编码的字符串，就连它的源码文件格式都是使用的 UTF-8 编码。Go 语言做到了真正的国际化！

### 1.2.6 语言的用途

Go 语言被设计成一门应用于搭载 Web 服务器，存储集群或类似用途的巨型中央服务器的系统编程语言。对于高性能分布式系统领域而言，Go 语言无疑比大多数其它语言有着更高的开发效率。它提供了海量并行的支持，这对于游戏服务端的开发而言是再好不过了。

Go 语言一个非常好的目标就是实现所谓的复杂事件处理（[CEP](#)），这项技术要求海量并行支持，高度的抽象化和高性能。当我们进入到物联网时代，CEP 必然会成为人们关注的焦点。

但是 Go 语言同时也是一门可以用于实现一般目标的语言，例如对于文本的处理，前端展现，甚至像使用脚本一样使用它。

值得注意的是，因为垃圾回收和自动内存分配的原因，Go 语言不适合用来开发对实时性要求很高的软件。



越来越多的谷歌内部的大型分布式应用程序都开始使用 Go 语言来开发，例如谷歌地球的一部分代码就是由 Go 语言完成的。

如果你想知道一些其它组织使用 Go 语言开发的实际应用项目，你可以到这个页面进行查看：<http://go-lang.cat-v.org/organizations-using-go>。出于隐私保护的考虑，许多公司的项目都没有展示在这个页面。我们将会在第 21 章讨论到一个使用 Go 语言开发的大型存储区域网络（SAN）案例。

在 Chrome 浏览器中内置了一款 Go 语言的编译器用于本地客户端（NaCl（译者注：为什么我觉得这是“氯化钠”？）），这很可能会被用于在 Chrome OS 中执行 Go 语言开发的应用程序。

Go 语言可以在 Intel 或 ARM 处理器上运行，因此它也可以在安卓系统下运行，例如 Nexus 系列的产品。

在 Google App Engine 中使用 Go 语言：2011 年 5 月 5 日，官方发布了用于开发运行在 Google App Engine 上的 Web 应用的 Go SDK，在此之前，开发者们只能选择使用 Python 或者 Java。这主要是 David Symonds 和 Nigel Tao 努力的成果。目前最新的稳定版是基于 r60.3 的 SDK 1.6.1，于 2011 年 12 月 13 日发布。当前 Go 语言的稳定版本是 Go 1（译者注：目前最新的稳定版是 Go1.1）。

### 1.2.7 关于特性丢失

许多能够在大多数面向对象语言中使用的特性 Go 语言都没有支持，但其中的一部分可能会在未来被支持。

- 为了简化设计，不支持函数重载和操作符重载
- 为了避免在 C/C++ 开发中的一些 Bug 和混乱，不支持隐式转换
- Go 语言通过另一种途径实现面向对象设计（第 10 - 11 章）来放弃类和类型的继承
- 尽管在接口的使用方面（第 11 章）可以实现类似变体类型的功能，但本身不支持变体类型
- 不支持动态加载代码
- 不支持动态链接库
- 不支持泛型
- 通过 `recover` 和 `panic` 来替代异常机制（第 13.2 - 3 节）
- 不支持断言
- 不支持静态变量

关于 Go 语言开发团队对于这些方面的讨论，你可以通过这个页面查看：[http://golang.org/doc/go\\_faq.html](http://golang.org/doc/go_faq.html)

### 1.2.8 使用 Go 语言编程

如果你有其它语言的编程经历（面向对象编程语言，如：Java、C#、Object-C、Python、Ruby），在你进入到 Go 语言的世界之后，你将会像迷恋你的 X 语言一样无法自拔。Go 语言使用了与其它语言不同的设计模式，所以当你尝试将你的 X 语言的代码迁移到 Go 语言时，你将会非常失望，所以你需要从头开始，用 Go 的理念来思考。

如果你在至高点使用 Go 的理念来重新审视和分析一个问题，你通常会找到一个适用于 Go 语言的优雅解决方案。

### 1.2.9 小结

这里列举一些 Go 语言的必杀技：

- 简化问题，易于学习
- 内存管理，简洁语法，易于使用
- 快速编译，高效开发
- 高效执行
- 并发支持，轻松驾驭
- 静态类型
- 标准类库，规范统一
- 易于部署
- 文档全面
- 免费开源

## 2 安装与运行环境

本章主要介绍 Go 语言的开发环境的搭建以及开发工具的选择。

### 2.1 平台与架构

（译者注：由于 Go 语言版本更替，本节中的相关内容经原作者同意将被直接替换而不作另外说明）

Go 语言开发团队开发了适用于以下操作系统的编译器：

- Linux
- FreeBSD
- Mac OS X（也称为 Darwin）

目前有 2 个版本的编译器：Go 原生编译器 `gc` 和非原生编译器 `gccgo`，这两款编译器都是在类 Unix 系统下工作。其中，`gc` 版本的编译器已经被移植到 Windows 平台上，并集成在主要发行版中，你也可以通过安装 MinGW 从而在 Windows 平台下使用 `gcc` 编译器。这两个编译器都是以单通道的方式工作。

你可以获取以下平台上的 Go 1.1 源码和二进制文件：

- FreeBSD 7+：amd64 和 386 架构
- Linux 2.6+：amd64、386 和 arm 架构
- Mac OS X (Snow Leopard + Lion)：amd64 和 386 架构
- Windows 2000+：amd64 和 386 架构

对于非常底层的纯 Go 语言代码或者包而言，在各个操作系统平台上的可移植性是非常强的，只需要将源码拷贝到相应平台上进行编译即可，或者可以使用交叉编译来构建目标平台的应用程序（第 2.2 节）。但如果你打算使用 `cgo` 或者类似文件监控系统的软件，就需要根据实际情况进行相应地修改了。

#### 1. Go 原生编译器 `gc`：

主要基于 Ken Thompson 先前在 Plan 9 操作系统上使用的 C 工具链。

Go 语言的编译器和链接器都是使用 C 语言编写并产生本地代码，Go 不存在自我引导之类的功能。因此如果使用一个有不同指令集的编译器来构建 Go 程序，就需要针对操作系统和处理器架构（32 位操作系统或 64 位操作系统）进行区别对待。

这款编译器使用非世代、无压缩和并行的方式进行编译，它的编译速度要比 `gccgo` 更快，产生更好的本地代码，但编译后的程序不能够使用 `gcc` 进行链接。

编译器目前支持以下基于 Intel 或 AMD 处理器架构的程序构建。

<i>No of bits</i>	<i>Processor name</i>	<i>Compiler</i>	<i>Linker</i>
<b>64 bit</b> implementation	<b>amd64</b> (also named <b>x86-64</b> )	<b>6g</b>	<b>6l</b>
<b>32 bit</b> implementation	<b>386</b> (also named <b>x86</b> or <b>x86-32</b> )	<b>8g</b>	<b>8l</b>
<b>32 bit RISC</b> implementation	<b>arm (ARM)</b>	<b>5g</b>	<b>5l</b>

图 2.1 gc 编译器支持的处理器架构

当你第一次看到这套命名系统的时候你会觉得很奇葩，不过这些命名都是来自于 Plan 9 项目。

**g** = 编译器：将源代码编译为项目代码（程序文本）  
**l** = 链接器：将项目代码链接到可执行的二进制文件（机器代码）

（相关的 C 编译器名称为 6c、8c 和 5c，相关的汇编器名称为 6a、8a 和 5a）

**标记（Flags）**是指可以通过命令行设置可选参数来影响编译器或链接器的构建过程或得到一个特殊的目标结果。

可用的编译器标记如下：

```
flags:
-I 针对包的目录搜索
-d 打印声明信息
-e 不限制错误打印的个数
-f 打印栈结构
-h 发生错误时进入恐慌（panic）状态
-o 指定输出文件名 // 详见第 3.4 节
-S 打印产生的汇编代码
-V 打印编译器版本 // 详见第 2.3 节
-u 禁止使用 unsafe 包中的代码
-w 打印归类后的语法解析树
-x 打印 lex tokens
```

从 Go 1.0.3 版本开始，不再使用 8g, 8l 之类的指令进行程序的构建，取而代之的是统一的 `go build` 和 `go install` 等命令，而这些指令会自动调用相关的编译器或链接器。

如果你想获得更深层次的信息，你可以在目录 `$GOROOT/src/cmd` 下找到编译器和链接器的源代码。Go 语言本身是由 C 语言开发的，而不是 Go 语言。词法分析程序是 GNU bison，语法分析程序是名为 `$GOROOT/src/cmd/gc/go.y` 的 yacc 文件，它会在同一目录输出 `y.tab.{c,h}` 文件。如果你想知道更多有关构建过程的信息，你可以查看相同目录下的

Makefile 文件，另一个版本的构建过程的概述可以在 `$GOROOT/src/make.bash` 中找到。

大部分的目录都包含了名为 `doc.go` 的文件，这个文件提供了更详细的信息。

## 2. gccgo 编译器：

一款相对于 `gc` 而言更加传统的编译器，使用 GCC 作为后端。GCC 是一款非常流行的 GNU 编译器，它能够构建基于众多处理器架构的应用程序。编译速度相对 `gc` 较慢，但产生的本地代码运行要稍微快一点。它同时也提供一些与 C 语言之间的互操作性。

从 Go 1 版本开始，`gc` 和 `gccgo` 在编译方面都有等价的功能。

## 3. 文件扩展名与包（package）：

Go 语言源文件的扩展名很显然就是 `.go`。

C 文件使用后缀名 `.c`，汇编文件使用后缀名 `.s`。所有的源代码文件都是通过包（packages）来组织。包含可执行代码的包文件在被压缩后使用扩展名 `.a`（AR 文档）。

Go 语言的标准库（第 9.1 节）包文件在被安装后就是使用这种格式的文件。

**注意：**当你在创建目录时，文件夹名称永远不应该包含空格，而应该使用下划线“`_`”或者其它一般符号代替。

## 2.2 Go 环境变量

（译者注：由于 Go 语言版本更替，本节中的相关内容经原作者同意将被直接替换而不作另外说明）

Go 开发环境依赖于一些操作系统环境变量，你最好在安装 Go 之间就已经设置好他们。如果你使用的是 Windows 的话，你完全不用进行手动设置，Go 将被默认安装在目录 `c:/go` 下。这里列举几个最为重要的环境变量：

- `$GOROOT` 表示 Go 在你的电脑上的安装位置，它的值一般都是 `$HOME/go`，当然，你也可以安装在别的地方。
- `$GOARCH` 表示目标机器的处理器架构，它的值可以是 386，amd64 或 arm。
- `$GOOS` 表示目标机器的操作系统，它的值可以是 darwin，freebsd，linux 或 windows

- `$GOBIN` 表示编译器和链接器的安装位置，默认是 `$GOROOT/bin`，如果你使用的是 Go 1.0.3 及以后的版本，一般情况下你可以将它的值设置为空，Go 将会使用前面提到的默认值。

目标机器是指你打算运行你的 Go 应用程序的机器。

Go 编译器支持交叉编译，也就是说你可以在一台机器上构建运行在具有不同操作系统和处理器架构上运行的应用程序，也就是说编写源代码的机器可以和目标机器有完全不同的特性（操作系统与处理器架构）。

为了区分本地机器和目标机器，你可以使用 `$GOHOSTOS` 和 `$GOHOSTARCH` 设置目标机器的参数，这两个变量只有在进行交叉编译的时候才会用到，如果你不进行显示设置，他们的值会和本地机器（`$GOOS` 和 `$GOARCH`）一样。

- `$GOPATH` 默认采用和 `$GOROOT` 一样的值，但从 Go 1.1 版本开始，你必须修改为其它路径。它可以包含多个包含 Go 语言源码文件、包文件和可执行文件的路径，而这些路径下又必须分别包含三个规定的目录：`src`，`pkg` 和 `bin`，这三个目录分别用于存放源码文件、包文件和可执行文件。
- `$GOARM` 专门针对基于 arm 架构的处理器，它的值可以是 5 或 6，默认为 6。
- `$GOMAXPROCS` 用于设置应用程序可使用的处理器个数与核数，详见第 14.1.3 节

在接下来的章节中，我们将会讨论如何在 Linux, Mac OS X 和 Windows 上安装 Go 语言。在 FreeBSD 上的安装和 Linux 非常类似。开发团队正在尝试将 Go 语言移植到其它例如 OpenBSD, DragonFlyBSD, NetBSD, Plan 9, Haiku 和 Solaris 操作系统上，你可以在这个页面找到最近的动态：<http://go-lang.cat-v.org/os-ports>

## 2.3 在 Linux 上安装 Go

（译者注：由于 Go 语言版本更替，本节中的相关内容经原作者同意将被直接替换而不作另外说明）

如果你能够自己下载并编译 Go 的源代码来说是非常有教育意义的，你可以根据这个页面找到安装指南和下载地址：<http://golang.org/doc/install.html>。

我们接下来也会带你一步步的完成安装过程。

### 1. 设置 Go 环境变量

我们在 Linux 系统下一般通过文件 `$HOME/.bashrc` 配置自定义环境变量，根据不同的发行版也可能是文件 `$HOME/.profile`，然后使用 `gedit` 或 `vi` 来编辑文件内容。

```
export GOROOT=$HOME/go
export GOBIN=$GOROOT/bin
export GOARCH=386
export GOOS=linux
```

（译者注：目前的 Go 版本一般情况下已不需要设置 \$GOBIN ）

为了确保相关文件在文件系统的任何地方都能被调用，你还需要添加以下内容：

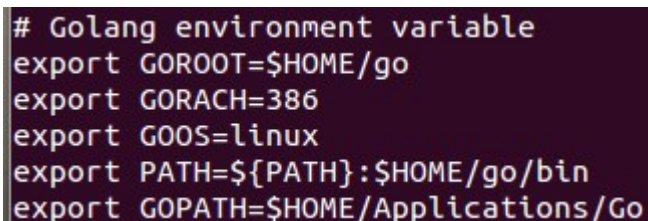
```
export PATH=$GOROOT/bin:$PATH
```

在开发 Go 项目时，你还需要一个环境变量来保存你的工作目录。

```
export GOPATH=$HOME/Applications/Go
```

\$GOPATH 可以包含多个工作目录，取决于你的个人情况。如果你设置了多个工作目录，那么当你在之后使用 `go get`（远程包安装命令）时远程包将会被安装在第一个目录下。

在完成这些设置后，你需要在终端输入指令 `source .bashrc` 以使这些环境变量生效。然后重启终端，输入 `go env` 和 `env` 来检查环境变量是否设置正确。



```
# Golang environment variable
export GOROOT=$HOME/go
export GOARCH=386
export GOOS=linux
export PATH=${PATH}:$HOME/go/bin
export GOPATH=$HOME/Applications/Go
```

图 2.2 在终端输入 `go env` 后打印的信息

## 2. 安装 C 工具

Go 的工具链是用 C 语言编写的，因此在安装 Go 之前你需要先安装相关的 C 工具。如果你使用的是 Ubuntu 的话，你可以在终端输入以下指令（译者注：由于网络环境的特殊性，你可能需要将每个工具分开安装）。

```
sudo apt-get install bison ed gawk gcc libc6-dev make
```

你可以在其它发行版上使用 RPM 之类的工具。

## 3. 安装 Mercurial

Go 源代码是通过 Mercurial 来进行版本管理的，因此你必须在你下载 Go 源代码之前安装这款工具。你可以使用指令 `hg` 来检查你的计算机上是否已经安装该工具，如果没有，请使用以下指令来完成安装：



```
sudo apt-get install python-setuptools
sudo apt-get install python-dev
sudo apt-get install build-essential
sudo apt-get install mercurial
```

#### 4. 获取 Go 源代码

通过以下指令来从服务器获取 Go 源代码到你的计算机上，这里我们直接使用 `$GOROOT` 的值，在获取源代码之前，你不能手动创建相关目录，否则将导致获取失败。

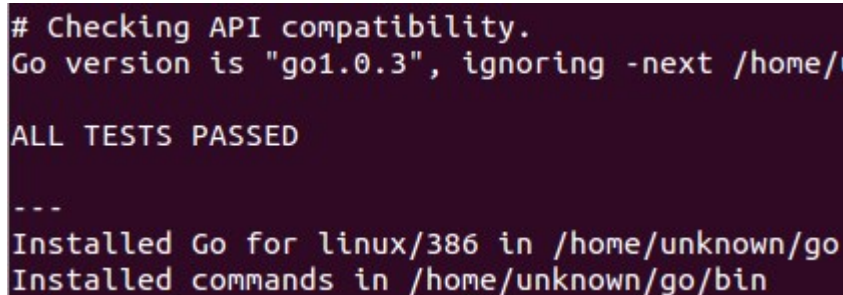
```
hg clone -r release https://go.googlecode.com/hg/ $GOROOT
```

#### 5. 构建 Go

在终端使用以下指令来进行编译工作。

```
cd $GOROOT/src
./all.bash
```

在完成编译之后（通常在 1 分钟以内，如果你在 B 型树莓派上编译，一般需要 1 个小时），你会在终端看到如下信息被打印：



```
# Checking API compatibility.
Go version is "go1.0.3", ignoring -next /home/
ALL TESTS PASSED
---
Installed Go for linux/386 in /home/unknown/go
Installed commands in /home/unknown/go/bin
```

图 2.3 完成编译后在终端打印的信息

#### 注意事项

如果你在编译过程中发生错误，请尝试使用指令 `hg pull -u` 来更新源代码，然后进行第 5 步。

#### 注意事项

在测试 `net/http` 包时有一个测试会尝试连接 `google.com`，你可能会看到如下所示的一个无厘头的错误报告：

```
'make[2]: Leaving directory `/localusr/go/src/pkg/net'
```

如果你正在使用一个带有防火墙的机器，我建议你可以在编译过程中暂时关闭防火墙，以避免不必要的错误。

解决这个问题的另一个办法是通过设置环境变量 `$DISABLE_NET_TESTS` 来告诉构建工具忽略 `net/http` 包的相关测试：

```
export DISABLE_NET_TESTS=1
```

如果这样还是没有办法阻止错误的发生的话，你可以通过添加 `net` 包到忽略测试列表，这个列表在 `$GOROOT/src/pkg` 下的 Makefile 中。

如果你完全不想运行包的测试，你可以直接运行 `./make.bash` 来进行单纯的构建过程。

## 6. 测试安装

使用你最喜爱的编辑器来输入以下内容，并保存为文件名 `test.go`。

Example 2.1 [hello\\_world1.go](#)

```
package main
func main() {
    println("Hello", "world")
}
```

切换相关目录到下，然后执行指令 `go run hello_world1.go`，将会打印信息：  
`Hello, world。`

## 7. 验证安装版本

你可以通过在终端输入指令 `go version` 来打印 Go 的版本信息。

如果你想要通过 Go 代码在运行时检测版本，可以通过以下例子实现。

Example 2.2 [version.go](#)

```
package main

import (
    "fmt"
    "runtime"
)

func main() {
    fmt.Printf("%s", runtime.Version())
}
```

这段代码将会输出 `go1.0.3` 或 `go1.1`。

## 8. 更新源代码版本

在终端输入以下指令来完成源代码的更新：

```
cd $GOROOT
hg pull
hg update release
```

```
cd src
sudo ./all.bash
```

你可以在这个页面查看到最新的稳定版：

<http://golang.org/doc/devel/release.html>

当前最新的稳定版 Go 1 系列于 2012 年 3 月 28 日发布。

Go 的源代码有以下三个分支：

- Go release: 最新稳定版，实际开发最佳选择
- Go weekly: 包含最近更新的版本，一般每周更新一次
- Go tip: 永远保持最新的版本，相当于内测版

你可以通过 gofix 这个工具来进行 Go 源代码的版本迁移，将旧版本的代码升级到最新版。

当你在使用不同的版本时，注意官方博客发布的信息，因为你所查阅的文档可能和你正在使用的版本不相符。

## 2.4 在 Mac OS X 上安装 Go

（译者注：由于 Go 语言版本更替，本节中的相关内容经原作者同意将被直接替换而不作另外说明）

如果你想要在你的 Mac 系统上安装 Go，则必须使用 Intel 64 位处理器，Go 不支持 PowerPC 处理器。

你可以通过该页面查看有关在 PowerPC 处理器上的移植进度：

<https://codedr-go-ppc.googlecode.com/hg/>。

### 注意事项

在 Mac 系统下使用到的 C 工具链是 Xcode 的一部分，因此你需要通过安装 Xcode 来完成这些工具的安装。你并不需要安装完整的 Xcode，而只需要安装它的命令行工具部分。

你可以在这个页面下载到 Mac 系统下的一键安装包或源代码自行编译：

<https://code.google.com/p/go/downloads/list>

通过源代码编译安装的过程与环境变量的配置与在 Linux 系统非常相似，因此不再赘述。

## 2.5 在 Windows 上安装 Go

（译者注：由于 Go 语言版本更替，本节中的相关内容经原作者同意将被直接替换而不作另外说明）

你可以在这个页面下载到 Windows 系统下的一键安装包：

<http://code.google.com/p/go/downloads/list>

前期的 Windows 移植工作由 Hector Chu 完成，但目前的发行版已经由 Joe Poirier 全职维护。

在完成安装包的安装之后，你只需要配置 `$GOPATH` 这一个环境变量就可以开始使用 Go 语言进行开发了，其它的环境变量安装包均会进行自动设置。在默认情况下，Go 将会被安装在目录 `c:\go` 下，但如果你在安装过程中修改安装目录，则可能需要手动修改所有的环境变量的值。

如果你想要测试安装，则可以使用指令 `go run` 运行 [hello\\_world1.go](#)。

如果发生错误 `fatal error: can't find import: fmt` 则说明你的环境变量没有配置正确。

如果你想要在 Windows 下使用 `cgo`（调用 C 语言写的代码），则需要安装 [MinGW](#)，如果你使用的是 64 位操作系统，请务必安装 64 位版本的 MinGW。安装完成进行环境变量等相关配置即可使用。

在 Windows 下运行在虚拟机里的 Linux 系统上安装 Go：

如果你想要在 Windows 下的虚拟机里的 Linux 系统上安装 Go，你可以选择使用虚拟机软件 [VMware](#)，下载 [VMware player](#)，搜索并下载一个你喜欢的 Linux 发行版镜像，然后安装到虚拟机里，安装 Go 的流程参考第 2.3 节中的内容。

## 2.6 安装目录清单

你的 Go 安装目录（`$GOROOT`）的文件夹结构应该如下所示：

README, AUTHORS, CONTRIBUTORS, LICENSE

- `\bin` 包含可执行文件，如：编译器，Go 工具
- `\doc` 包含示例程序，代码工具，本地文档等
- `\include` 包含 C/C++ 头文件
- `\lib` 包含文档模版
- `\misc` 包含与支持 Go 编辑器有关的配置文件以及 `cgo` 的示例
- `\pkg\os_arch` 包含标准库的包的对象文件（`.a`）
- `\src` 包含源代码构建脚本
- `\src\cmd` 包含 Go 和 C 的编译器和命令行脚本
- `\src\lib9` `\src\libbio` `\src\libmach` 包含 C 文件
- `\src\pkg` 包含 Go 标准库的包的完整源代码（Go 是一门开源语言）

## 2.7 Go 类虚拟机 (runtime)

尽管 Go 编译器产生的是本地可执行代码，这些代码仍旧运行在 Go 的 runtime（这部分的代码可以在 runtime 包中找到）当中。这个 runtime 类似 Java 和 .NET 语言所用到的虚拟机，它负责管理包括内存分配、垃圾回收（第 10.8 节）、栈处理、goroutine、channel、切片 (slice)、map 和反射 (reflection) 等等。

runtime 主要由 C 语言编写，并且是每个 Go 包的最顶级包。你可以在目录 `$GOROOT/src/pkg/runtime/` 中找到相关内容（主要看 `mgc0.c` 和其它以 `m` 开头的文件）。

**垃圾回收器** Go 拥有简单却高效的标记-清除回收器。它的主要思想来源于 IBM 的可复用垃圾回收器，旨在打造一个高效、低延迟的并发回收器。目前 `gccgo` 还没有回收器，同时适用 `gc` 和 `gccgo` 的新回收器正在研发中。使用一门具有垃圾回收功能的编程语言不代表你可以避免内存分配所带来的问题，分配和回收内容都是消耗 CPU 资源的一种行为。

Go 的可执行文件都比相对应的源代码文件要大很多，这恰恰说明了 Go 的 runtime 嵌入到了每一个可执行文件当中。当然，在部署到数量巨大的集群时，较大的文件体积也是比较头疼的问题。但总得来说，Go 的部署工作还是要比 Java 和 Python 轻松得多。因为 Go 不需要依赖任何其它文件，它只需要一个单独的静态文件，这样你也不会像使用其它语言一样在各种不同版本的依赖文件之间混淆。

## 2.8 Go 解释器

因为 Go 具有像动态语言那样快速编译的能力，自然而然地就有人会问 Go 语言能否在 REPL (real-eval-print loop) 编程环境下实现。Sebastien Binet 已经使用这种环境实现了一个 Go 解释器，你可以在这个页面找到：

<https://bitbucket.org/binet/igo>

## 3 编辑器、集成开发环境与其它工具

因为 Go 语言还是一门相对年轻的编程语言，所以不管是在集成开发环境（IDE）还是相关的插件方面，发展都不是很成熟。不过目前还是有一些 IDE 能够较好地支持 Go 的开发，有些开发工具甚至是跨平台的，你可以在 Linux、Mac OS X 或者 Windows 下工作。

你可以通过查阅该页面来获取 Go 开发工具的最新信息：<http://go-lang.cati.v.org/text-editors/>

### 3.1 Go 开发环境的基本要求

这里有一个你可以期待你用来开发 Go 的集成开发环境有哪些特性的列表，从而替代你使用文本编辑器写代码和命令行编译与链接程序的方式。

1. 语法高亮是必不可少的功能，这也是为什么每个开发工具都提供配置文件来实现自定义配置的原因。
2. 可以自动保存代码，至少在每次编译前都会保存。
3. 可以显示代码所在的行数。
4. 拥有较好的项目文件纵览和导航能力，可以同时编辑多个源文件并设置书签，能够匹配括号，能够跳转到某个函数或类型的定义部分。
5. 完美的查找和替换功能，替换之前最好还能预览结果。
6. 可以注释或取消注释选中的一行或多行代码。
7. 当有编译错误时，双击错误提示可以跳转到发生错误的位置。
8. 跨平台，能够在 Linux、Mac OS X 和 Windows 下工作，这样就可以专注于一个开发环境。
9. 最好是免费的，不过有些开发者还是希望能够通过支付一定金额以获得更好的开发环境。
10. 最好是开源的。
11. 能够通过插件架构来轻易扩展和替换某个功能。
12. 尽管集成开发环境本身就是非常复杂的，但一定要让人感觉操作方便。
13. 能够通过代码模版来简化编码过程从而提升编码速度。
14. 使用 Go 项目的概念来浏览和管理项目中的文件，同时还要拥有构建系统的概念，这样才能更加方便的构建、清理或运行我们建立的程序或项目。构建出的程序需要能够通过命令行或 IDE 内部的控制台运行。
15. 拥有断点、检查变量值、单步执行、逐过程执行标识库中代码的能力。
16. 能够方便的存取最近使用过的文件或项目。
17. 拥有对包、类型、变量、函数和方法的智能代码补全的功能。
18. 能够对项目或包中的代码建立抽象语法树视图（AST-view）。
19. 内置 Go 的相关工具。
20. 能够方便完整地查阅 Go 文档。
21. 能够方便地在不同的 Go 环境之间切换。
22. 能够导出不同格式的代码文件，如：PDF，HTML 或格式化后的代码。



23. 针对一些特定的项目有项目模板，如：Web 应用，App Engine 项目，从而能够更快地开始开发工作。
24. 具备代码重构的能力。
25. 集成像 hg 或 git 这样的版本控制工具。
26. 集成 Google App Engine 开发及调试的功能。

## 3.2 编辑器和集成开发环境

（译者注：由于 Go 语言版本更替，本节中的相关内容经原作者同意将被直接替换而不作另外说明）

这些编辑器包含了代码高亮和其它与 Go 有关的一些使用工具：Emacs、Vim、Xcode3、KD Kate、TextWrangler、BBEdit、McEdit、TextMate、TextPad、JEdit、SciTE、Nano、Notepad++、Geany、SlickEdit 和 Sublime Text 2。

你可以将 Linux 的文本编辑器 GEdit 改造成一个很好的 Go 开发工具，详见页面：<http://gohelp.wordpress.com/>。

[Sublime Text](#) 是一个革命性的跨平台（Linux、Mac OS X、Windows）文本编辑器，它支持编写非常多的编程语言代码。对于 Go 而言，它有一个插件叫做 [GoSublime](#) 来支持代码补全和代码模板。

这里还有一些更加高级的 Go 开发工具，其中一些是以插件的形式利用本身是作为开发 Java 的工具。

NetBeans [Go For NetBeans](#) 插件是收费的，它支持语法高亮和代码模板；另外免费的插件正在开发中，你可以通过这个页面获取最新进展：  
<http://www.tunnelvisionlabs.com/downloads/go/>

[gogo](#) 可以运行在 Linux 和 Mac 上的一个非常基础的开发工具。

[GoIde](#) 是一个 IntelliJ IDEA 的插件，具有很好的操作体验和代码补全功能，你可以从这个页面下载：<https://github.com/mtoader/google-go-lang-idea-plugin>

[LiteIDE \(golangide\)](#) 这是一款专门针对 Go 开发的集成开发环境，在编辑、编译和运行 Go 程序和项目方面都有非常好的支持。同时还包括了对源代码的抽象语法树视图和一些内置工具。（译者注：此开发环境由国人 vfc 大叔开发）

[GoClipse](#) 是一款 Eclipse IDE 的插件，拥有非常多的特性以及通过 GoCode 来实现代码补全功能。



如果你对集成开发环境都不是很熟悉，那就使用 LiteIDE 吧，另外使用 GoClique 或者 GoIde 也是不错的选择。

**代码补全** 一般都是通过内置 GoCode 实现的（如：LieteIDE、GoClique），如果需要手动安装 GoCode，在命令行输入指令 `go get -u github.com/nsf/gocode` 即可。  
（译者注：务必事先配置好 Go 环境变量）

下表展示了目前三个最好的集成开发环境在功能上的对比，其中 + 表示具备此项功能，++ 表示支持度很好，空白表示不支持此功能。

	Golang LiteIDE	GoClipse	Golde
Syntax highlighting	++	+	+
Automatic saving before building	+		
Line numbering	+	+	
Bookmarks			
Bracket matching		+	
Find / Replace	+	++	
Go to definition			
(Un)Comment		+	
Compiler error double click	++	+	
Cross platform	+	+	
Free	+	+	
Open source	+	+	
Plugin-architecture	+	+	
Easy to use	++	+	
Code snippets			
Project concept	+	+	
Code Folding			
Build system	+	+	
Debugging	+	+	

Recent files and projects	+	+	
Code completion	+	++	
AST-view of code	++	+	
Built-in go tools	+	+	
Browsing of go documentation	+		
Easy switching different Go-environments	++	+	
Exporting code to different formats	++	+	
Project templates			
Integration with version control-systems			
Integration with Google App Engine		++	

接下来会对这三个集成开发环境做更加详细的说明。

### 3.2.1 Golang LiteIDE

这款 IDE 的当前最新版本号为 X17，你可以从该页面下载到：

<http://code.google.com/p/golangide/>

LiteIDE 是一款非常好用的轻量级 Go 集成开发环境（基于 QT、Kate 和 SciTE），包含了跨平台开发及其它必要的特性，对代码编写、自动补全和运行调试都有极佳的支持。它采用了 Go 项目的概念来对项目文件进行浏览和管理，它还支持在各个 Go 开发环境之间随意切换以及交叉编译的功能。

同时，它具备了抽象语法树视图的功能，可以清楚地纵览项目中的常量、变量、函数、不同类型以及他们的属性和方法。

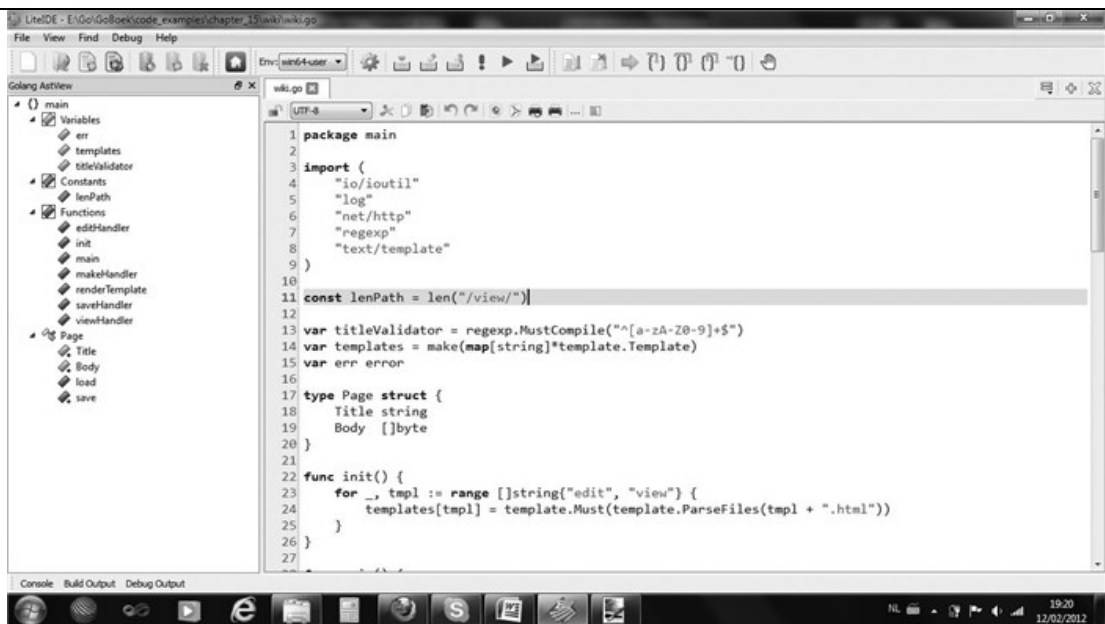


图 3.1 LiteIDE 代码编辑界面和抽象语法树视图

### 3.2.2 GoClipse

该款插件的当前最新版本号为 0.7.6，你可以从该页面下载到：

<http://code.google.com/p/goclipse/>

其依附于著名的 Eclipse 这个大型开发环境，虽然需要安装 JVM 运行环境，但却可以很容易地享有 Eclipse 本身所具有的诸多功能。这是一个非常好的编辑器，完善的代码补全、抽象语法树视图、项目管理和程序调试功能。

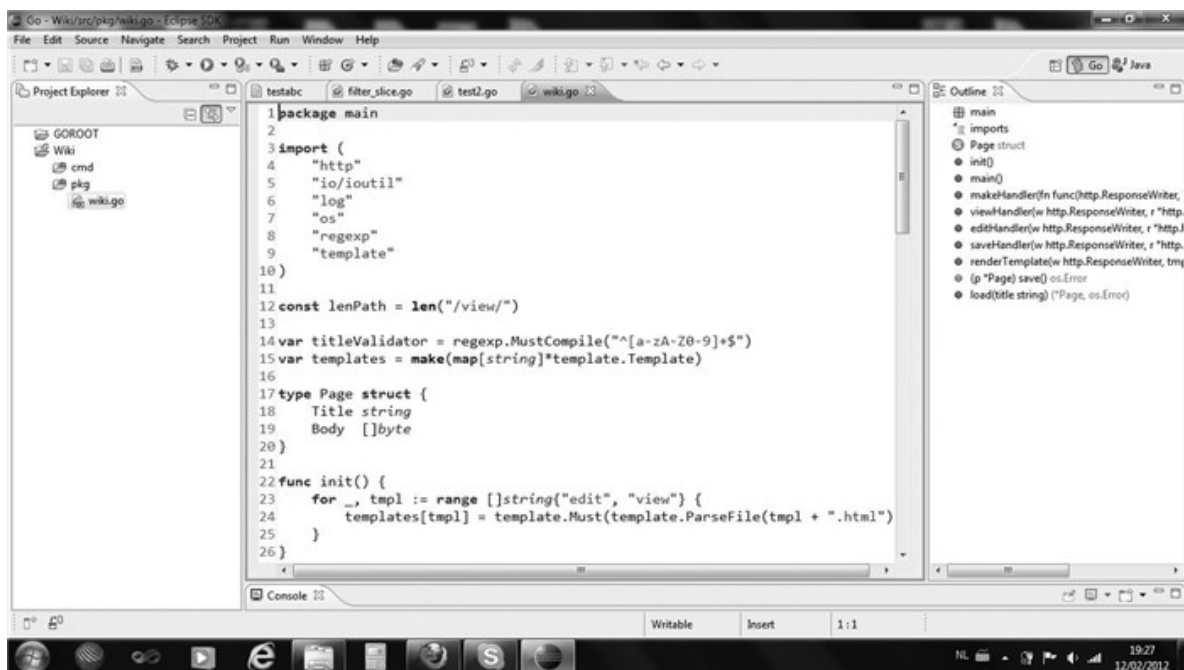


图 3.2 GoClipse 代码编辑界面、抽象语法树视图和项目管理

### 3.3 调试器

（译者注：由于 Go 语言版本更替，本节中的相关内容经原作者同意将被直接替换而不作另外说明）

应用程序的开发过程中调试是必不可少的一个环节，因此有一个好的调试器是非常重要的，可惜的是，Go 在这方面的发展还不是很完善。目前可用的调试器是 gdb，最新版均以内置在集成开发环境 LiteIDE 和 GoClipse 中，但是该调试器的调试方式并不灵活且操作难度较大。

如果你不想使用调试器，你可以按照下面的一些有用的方法来达到基本调试的目的：

1. 在合适的位置使用打印语句输出相关变量的值（`print/println` 和 `fmt.Print/fmt.Println/fmt.Printf`）。
2. 在 `fmt.Printf` 中使用下面的说明符来打印有关变量的相关信息：
  - `%+v` 打印包括字段在内的实例的完整信息
  - `%#v` 打印包括字段和限定类型名称在内的实例的完整信息
  - `%T` 打印某个类型的完整说明
3. 使用 `panic` 语句（第 13.2 节）来获取栈跟踪信息（直到 `panic` 时所有被调用函数的列表）。
4. 使用关键字 `defer` 来跟踪代码执行过程（第 6.4 节）。

### 3.4 构建并运行 Go 程序

（译者注：由于 Go 语言版本更替，本节中的相关内容经原作者同意将被直接替换而不作另外说明）

在大多数 IDE 中，每次构建程序之前都会自动调用源码格式化工具 `gofmt` 并保存格式化后的源文件。如果构建成功则不会输出任何信息，而当发生编译时错误时，则会指明源码中具体第几行出现了什么错误，如：`a declared and not used`。一般情况下，你可以双击 IDE 中的错误信息直接跳转到发生错误的那一行。

如果程序执行一切顺利并成功退出后，将会在控制台输出 `Program exited with code 0`。

从 Go 1 版本开始，使用 Go 自带的更加方便的工具来构建应用程序：

- `go build` 编译并安装自身包和依赖包
- `go install` 安装自身包和依赖包

### 3.5 格式化代码

Go 开发团队不想要 Go 语言像许多其它语言那样总是在为代码风格而引发无休止的争论，浪费大量宝贵的开发时间，因此他们制作了一个工具：`go fmt`

(`gofmt`)。这个工具可以将你的源代码格式化成符合官方统一标准的风格，属于语法风格层面上的小型重构。遵循统一的代码风格是 Go 开发中无可撼动的铁律，因此你必须在编译或提交版本管理系统之前使用 `gofmt` 来格式化你的代码。

尽管这种做法也存在一些争论，但使用 `gofmt` 后你不再需要自成一套代码风格而是和所有人使用相同的规则。这不仅增强了代码的可读性，而且在接手外部 Go 项目时，可以更快地了解其代码的含义。此外，大多数开发工具也都内置了这一功能。

Go 对于代码的缩进层级方面使用 `tab` 还是空格并没有强制规定，一个 `tab` 可以代表 4 个或 8 个空格。在实际开发中，1 个 `tab` 应该代表 4 个空格，而在本身的例子当中，每个 `tab` 代表 8 个空格。至于开发工具方面，一般都是直接使用 `tab` 而不替换成空格。

在命令行输入 `gofmt -w program.go` 会格式化该源文件的代码然后将格式化后的代码覆盖原始内容（如果不加参数 `-w` 则只会打印格式化后的结果而不重写文件）；`gofmt -w *.go` 会格式化并重写所有 Go 源文件；`gofmt map1` 会格式化并重写 `map1` 目录及其子目录下的所有 Go 源文件。

`gofmt` 也可以通过在参数 `-r` 后面加入用双引号括起来的替换规则实现代码的简单重构，规则的格式：`<原始内容> -> <替换内容>`。

实例：

```
gofmt -r "(a) -> a" -w *.go
```

上面的代码会将源文件中没有意义的括号去掉。

```
gofmt -r "a[n:len(a)] -> a[n:]" -w *.go
```

上面的代码会将源文件中多余的 `len(a)` 去掉。（译者注：了解 `slice` 之后就明白这为什么是多余的了）

```
gofmt -r 'A.Func1(a,b) -> A.Func2(b,a)' -w *.go
```

上面的代码会将源文件中符合条件的函数的参数调换位置。

如果想要了解有关 `gofmt` 的更多信息，请访问该页面：

<http://golang.org/cmd/gofmt/>

## 3.6 生成代码文档

`go doc` 工具会从 Go 程序和包文件中提取顶级声明的首行注释以及每个对象的相关注释，并生成相关文档。

它也可以作为一个提供在线文档浏览的 web 服务器，<http://golang.org> 就是通过这种形式实现的。

### 一般用法

- `go doc package` 获取包的文档注释，例如：`go doc fmt` 会显示使用 `godoc` 生成的 `fmt` 包的文档注释。
- `go doc package/subpackage` 获取子包的文档注释，例如：`go doc container/list`。
- `go doc package function` 获取某个函数在某个包中的文档注释，例如：`go doc fmt.Printf` 会显示有关 `fmt.Printf()` 的使用说明。

这个工具只能获取在 Go 安装目录下 `.../go/src/pkg` 中的注释内容。此外，它还可以作为一个本地文档浏览 web 服务器。在命令行输入 `godoc -http=:6060`，然后使用浏览器打开 <http://localhost:6060> 后，你就可以看到本地文档浏览服务器提供的页面。

`godoc` 也可以用于生成非标准库的 Go 源码文件的文档注释（第 9.6 章）。

如果想要获取更多有关 `godoc` 的信息，请访问该页面：

<http://golang.org/cmd/godoc/>

## 3.7 其它工具

Go 自带的工具集主要使用脚本和 Go 语言自身编写的，目前版本的 Go 实现了以下三个工具：

- `go install` 是安装 Go 包的工具，类似 Ruby 中的 `rubygems`。主要用于安装非标准库的包文件，将源代码编译成对象文件。
- `go fix` 用于将你的 Go 代码从旧的发行版迁移到最新的发行版，它主要负责简单的、重复的、枯燥无味的修改工作，如果像 API 等复杂的函数修改，工具则会给出文件名和代码行数的提示以便让开发人员快速定位并升级代码。Go 开发团队一般也使用这个工具升级 Go 内置工具以及谷歌内部项目的代码。`go fix` 之所以能够正常功能是因为 Go 在标准库就提供生成抽象语法树和通过抽象语法树对代码进行还原的功能。该工具会尝试更新当前目录下的所有 Go 源文件，并在完成代码更新后在控制台输出相关的文件名称。
- `go test` 是一个轻量级的单元测试框架（第 13 章）。

## 3.8 Go 性能说明



根据 Go 开发团队和基本的算法测试，Go 的性能大概在 C 语言的 10%~20% 之间（译者注：由于出版时间限制，该数据应为 2013 年 3 月 28 日之前产生）。虽然没有官方的性能标准，但是与其它各个语言相比已经拥有非常出色的表现。

如果说 Go 语言的执行效率大约比 C++ 慢 20% 也许更有实际意义。保守估计在相同的环境和执行目标的情况下，Go 程序比 Java 或 Scala 应用程序要快上 2 倍，并比这两门语言使用少占用 70% 的内存。在很多情况下这种比较是没有意义的，因为像谷歌这样拥有成千上万台服务器的公司都抛弃 C++ 而开始将 Go 用于生产环境已经足够说明它本身所具有的优势。

时下流行的语言大都是运行在虚拟机上，如：Java 和 Scala 使用的 JVM，C# 和 VB.NET 使用的 .NET CLR。尽管虚拟机的性能已经有了很大的提升，但任何使用 JIT 编译器和脚本语言解释器的编程语言（Ruby、Python、Perl 和 JavaScript）在 C 和 C++ 的绝对优势下甚至都无法在性能上望其项背。

如果说 Go 比 C++ 要慢 20%，那么 Go 就要比任何非静态和编译型语言快 2 到 10 倍，并且能够更加高效地使用内存。

其实比较多门语言之间的性能是一种非常猥琐的行为，因为任何一种语言都有其所长和薄弱的方面。例如在处理文本方面，那些只处理纯字节的语言显然要比处理 Unicode 这种更为复杂编码的语言要出色的多。有些人可能认为使用两种不同的语言实现同一个目标能够得出正确的结论，但是很多时候测试者可能对一门语言非常了解而对另一门语言只是大概明白，测试者对程序编写的手法在一定程度也会影响结果的公平性，因此测试程序应该分别由各自语言的擅长者来编写，这样才能得到具有可比性的结果。另外，像在统计学方面，人们很难避免人为因素对结果的影响，所以这在严格意义上并不是科学。还要注意的，测试结果的可比性还要根据测试目标来区别，例如很多发展十多年的语言已经针对各类问题拥有非常成熟的类库，而作为一门新生语言的 Go 语言，并没有足够的时间来推导各类问题的最佳解决方案。如果你想获取更多有关性能的资料，请访问 [Computer Language Benchmark Game](#)（详见引用 27）。

这里有一些评测结果：

- 比较 Go 和 Python 在简单的 web 服务器方面的性能，单位为传输量每秒：

原生的 Go http 包要比 web.py 快 7 至 8 倍，如果使用 web.go 框架则稍微差点，比 web.py 快 6 至 7 倍。在 Python 中被广泛使用的 tornado 异步服务器和框架在 web 环境下要比 web.py 快很多，Go 大概只比它快 1.2 至 1.5 倍（详见引用 26）。

- Go 和 Python 在一般开发的平均水平测试中，Go 要比 Python 3 快 25 倍左右，少占用三分之二的内存，但比 Python 大概多写一倍的代码（详见引用 27）。

- 根据 Robert Hundt（2011 年 6 月，详见引用 28）的文章对 C++、Java、Go 和 Scala，以及 Go 开发团队的反应（详见引用 29），可以得出以下结论：
  - Go 和 Scala 之间具有更多的可比性（都使用更少的代码），而 C++ 和 Java 都使用非常冗长的代码。
  - Go 的编译速度要比绝大多数语言都要快，比 Java 和 C++ 快 5 至 6 倍，比 Scala 快 10 倍。
  - Go 的二进制文件体积是最大的（每个可执行文件都包含 runtime）。
  - 在最理想的情况下，Go 能够和 C++ 一样快，比 Scala 快 2 至 3 倍，比 Java 快 5 至 10 倍。
  - Go 在内存管理方面也可以和 C++ 相媲美，几乎只需要 Scala 所使用的一半，比 Java 少 4 倍左右。

## 3.9 与其它语言进行交互

### 3.9.1 与 C 进行交互

工具 `cgo` 提供了对 FFI（外部函数接口）的支持，能够使用 Go 代码安全地调用 C 语言库，你可以访问 `cgo` 文档主页：<http://golang.org/cmd/cgo>。`cgo` 会替代 Go 编译器来产生可以组合在同一个包中的 Go 和 C 代码。在实际开发中一般使用 `cgo` 创建单独的 C 代码包。

如果你想要在你的 Go 程序中使用 `cgo`，则必须在单独的一行使用 `import "C"` 来导入，一般来说你可能还需要 `import "unsafe"`。

然后，你可以在 `import "C"` 之前使用注释（单行或多行注释均可）的形式导入 C 语言库（甚至有效的 C 语言代码），它们之间没有空行，例如：

```
// #include <stdio.h>
// #include <stdlib.h>
import "C"
```

名称 `"C"` 并不属于标准库的一部分，这只是 `cgo` 集成的一个特殊名称用于引用 C 的命名空间。在这个命名空间里所包含的 C 类型都可以被使用，例如 `C.uint`、`C.long` 等等，还有 `libc` 中的函数 `C.random()` 等也可以被调用。

当你想要使用某个类型作为 C 中函数的参数时，必须将其转换为 C 中的类型，反之亦然，例如：

```
var i int
C.uint(i) // 从 Go 中的 int 转换为 C 中的无符号 int
int(C.random()) // 从 C 中 random() 函数返回的 long 转换为 Go 中的 int
```

下面的 2 个 Go 函数 `Random()` 和 `Seed()` 分别调用了 C 中的 `C.random()` 和 `C.srandom()`。

### Example 3.2 [c1.go](#)

```
package rand
// #include <stdlib.h>
import "C"
func Random() int {
    return int(C.random())
}
func Seed(i int) {
    C.srandom(C.uint(i))
}
```

C 当中并没有明确的字符串类型，如果你想要将一个 `string` 类型的变量从 Go 转换到 C，可以使用 `C.CString(s)`；同样，可以使用 `C.GoString(cs)` 从 C 转换到 Go 中的 `string` 类型。

Go 的内存管理机制无法管理通过 C 代码分配的内存。

开发人员需要通过手动调用 `C.free` 来释放变量的内存：

```
defer C.free(unsafe.Pointer(Cvariable))
```

这一行最好紧跟在使用 C 代码创建某个变量之后，这样就不会忘记释放内存了。下面的代码展示了如何使用 `cgo` 创建变量、使用并释放其内存：

### Example 3.3 [c2.go](#)

```
package print
// #include <stdio.h>
// #include <stdlib.h>
import "C"
import "unsafe"
func Print(s string) {
    cs := C.CString(s)
    defer C.free(unsafe.Pointer(cs))
    C.fputs(cs, (*C.FILE)(C.stdout))
}
```

## 构建 `cgo` 包

你可以在使用将会在第 9.5 节讲到的 `Makefile` 文件（因为我们使用了一个独立的包），除了使用变量 `GOFILES` 之外，还需要使用变量 `CGOFILES` 来列出需要使用 `cgo` 编译的文件列表。例如，Example 3.2 中的代码就可以使用包含以下内容的 `Makefile` 文件来编译，你可以使用 `gomake` 或 `make`：

```
include $(GOROOT)/src/Make.inc
TARG=rand
CGOFILES=\
c1.go\
include $(GOROOT)/src/Make.pkg
```

## 3.9.2 与 C++ 进行交互

SWIG（简化封装器和接口生成器）支持在 Linux 系统下使用 Go 代码调用 C 或者 C++ 代码。这里有一些使用 SWIG 的注意事项：

- 编写需要封装的库的 SWIG 接口。
- SWIG 会产生 C 的存根函数。
- 这些库可以使用 cgo 来调用。
- 相关的 Go 文件也可以被自动生成。

这类接口支持方法重载、多重继承以及使用 Go 代码实现 C++ 的抽象类。

目前使用 SWIG 存在的一个问题是它无法支持所有的 C++ 库，比如说它就无法解析 TObject.h。