# 第 1 篇

# 基础知识

本篇是 Go 语言从入门到项目实践的基础知识篇。从 Go 语言开发环境的搭建及开发工具的使用讲起，结合 Go 语言语法、程序的编写和语法结构的剖析，带领读者快速步入 Go 语言的世界。读者在学完本篇内容后将会了解到 Go 语言程序元素的构成、基本数据类型及流程控制语句等内容。

项目基础篇的主要内容就是了解 Go 语言的概念、开发环境的搭建、基本数据类型、流程控制语句等基础内容，为后面更深入地学习 Go 语言打下坚实的基础。

- **第 1 章** 走进 Go 语言的世界
- **第 2 章** Go 语言程序元素的构成
- **第 3 章** 基本数据类型
- **第 4 章** 流程控制

# 第1章

# 走进 Go 语言的世界

## 本章概述

Go 语言是一门新生的、开源的编程语言，它的出现受到程序开发人员的广泛喜爱。随着 Go 语言的不断发展和完善，其应用范围逐渐扩大，为了让更多的人了解并掌握 Go 语言，本章将依次介绍 Go 语言的诞生背景、语言特性、Go 语言开发环境的部署、开发工具的使用方法、Go 语言的基本结构、第一个 Go 语言程序等基础内容。

## 知识导读

本章要点（已掌握的在方框中打钩）：

- ☐ Go 语言的特性。
- ☐ Go 语言的使用。
- ☐ Go 语言开发环境的搭建。
- ☐ Go 语言开发工具的使用。
- ☐ Go 语言的基本结构。

## 1.1 初识 Go 语言

学习 Go 语言之前，首先需要了解 Go 语言的诞生背景、语言特性及使用方法等。

### 1.1.1 Go 语言的诞生

Go 语言是一门新型的静态类型的编译型语言。Go 语言的诞生可能给大多数人带来了一个疑虑：目前已经有了多种编程语言，为什么还要发明 Go 语言？为什么还要学习 Go 语言？

在程序开发人员看来，尽管已经出现多种编程语言，但每种语言都有其独特的应用领域，在某个领域使用某种语言能达到收益/投入的最大化。例如，在嵌入式领域，汇编语言和 C 语言是首选；在操作系统领域，C 语言是首选；在系统级服务编程领域，C++是首选；在企业级应用程序和 Web 应用领域，Java 是首选。

最近几年，由于 C 和 C++在计算领域没有得到很好的发展，也没有出现新的、好用的系统编程语言，因此使得开发程度和系统效率等在很多情况下不能兼容。当执行效率较高时，就存在低效的开发和编译，如 C++；当执行效率低时，但拥有有效的编译，如.NET、Java 等。根据以上情况，就需要一种既有较高效的执行速度，又有高效的编译速度和开发速度的编程语言，因此 Go 语言就诞生了。

Go 语言是由 Google 公司推出的一个开源项目（系统开发语言），它是基于编译、垃圾收集和并发的编程语言。Go 语言最早是在 2007 年 9 月由 Robert Griesemer、Rob Pike 和 Ken Thompson 联合开发的，2009 年 11 月，Google 正式发布 Go 语言，并将其开源在 BSD 许可证下发行。

Go 语言不仅支持 Linux 和 Mac OS 平台，还支持 Windows 平台。Go 语言就是为了解决当下编程语言对并发支持不友好、编译速度慢、编程复杂等问题而诞生的。

## 1.1.2　Go 语言的特性

Go 语言是由 Google 公司开发的一种静态型、编译型并自带垃圾回收和并发的编程语言。

Go 语言与当前的传统开发语言（如 Java、PHP）相比具备许多新特性。例如，Go 语言拥有自动垃圾回收功能，同时也允许开发人员干预回收操作；Go 语言有着更加丰富的内置类型，在错误处理方面语法更加精简高效。在 Go 语言中，函数支持多个返回值，而且函数也是一种值类型，可以作为参数传递。

Go 语言的特性主要有以下几点：

### 1. 简单、易学

对于刚接触 Go 语言的读者来说，对该语言的熟悉过程为 1～2 天，之后就可以通过 Go 语言来解决一些简单的问题，一周左右读者就可以使用 Go 语言来完成一些既定的任务。

Go 语言的风格类似于 C 语言。其语法在 C 语言的基础上进行了大幅简化，去掉了不需要的表达式括号，循环也只有 for 一种表示方法，就可以实现数值、键值等各种遍历。因此，Go 语言非常容易上手。

### 2. 类型系统和抽象

每个编程语言都有自己的类型系统，当然 Go 语言也不例外。从 struct 关键字来说，Go 语言的类型定义参考了 C 语言中的结构（struct），但是 Go 语言并不像 C++和 Java 那样设计一个庞大而又复杂的类型系统，而是仅支持最基本的类型组合，不支持继承和重载。虽然 Go 语言没有类和继承的概念，但是它可以通过接口（interface）的概念来实现多态性。

### 3. Go 语言工程结构简单

Go 语言不像 C 语言那样需要头文件才能运行，Go 语言编译的文件都来自扩展名为 go 的源码文件，Go 语言还不需要解决方案、工程文件及 Make File。由于 Go 语言遵循 GOPATH 规则，因此，只需要将 Go 语言的工程文件按照 GOPATH 的规则进行填充即可，最后使用 go build 或 go install 进行编译。

### 4. 快速编译

Go 语言和其他语言一样，拥有一个健全的包管理机制，同时得益于包之间的树状依赖，Go 语言的初次编译速度可以和 C/C++相媲美，甚至二次编译的速度明显快于 C/C++，同时又拥有接近 Python 等解释语言的简洁和开发效率。Go 语言在执行速度、编译速度和开发效率之间做了权衡，尽量达到了快速编译、高效执行、易于开发的目标。

同时，Go 语言还支持交叉编译，可以在运行 Linux 系统的计算机上开发 Windows 下的应用程序。Go 语言源码文件格式默认都是使用 UTF-8 编码的。

### 5. 原生支持并发

Go 语言最有特色的特性就是从语言层支持并发，不需要第三方库、开发者的编程技巧及开发经验就可

以轻松地在 Go 语言运行时来帮助开发者决定如何使用 CPU 资源。Go 语言在语言层可以通过 goroutine 对函数实现并发执行。goroutine 类似于线程但是并非线程，goroutine 会在 Go 语言运行时进行自动调度。因此，Go 语言非常适合用于高并发网络服务的编写。

Go 语言对多核处理器的编程进行了优化，Go 语言从程序与结构方面来实现并发编程，这是 Go 语言最重要的特性之一。

**6. 开源免费**

由于 Go 语言是基于 BSD 协议完全开源的，因此能免费被任何人用于适合的商业目的。

### 1.1.3 Go 语言的使用

编程语言对于开发人员来说只是一种工具，不是选择最好的，而是选择最适合的，那么 Go 语言适用于哪些场景？

使用 Go 语言，可以让 Web 服务器端的开发变得更高效，能够充分发挥多核计算机的性能，拥有更出色的网络环境兼容能力。自动垃圾回收、类型安全、依赖严格、编译快速等特点都是 Go 语言的魅力所在。很显然，Go 语言的目标就是针对服务器端的 Web 开发领域。

Go 语言凭借其出色的并发能力，在高性能分布式系统领域如鱼得水，像集群系统、游戏服务器端等场景都可以把 Go 语言作为首选开发语言。但是，Go 语言并不适合开发强实时性的软件，垃圾回收和自动内存分配等因素导致 Go 语言在实时性上有些力不从心。

对于 Go 语言最初的构想是把它作为一个系统编程语言，但目前也被用于像 Web Server、存储架构等这类分布式、高并发系统中；同时还可以用于一般的文字处理和作为脚本程序使用。

Go 语言的编译器作为 Native Client 被内嵌到 Chrome 浏览器中，可以被 Web 应用程序用来执行本地代码；同时 Go 语言也可以运行在 Intel 和 ARM 处理器上。

目前，Go 语言已被 Google 集成到 Google APP Engine 中，在基于 Google App Engine 基础设施的 Web 应用中也得到了很好的应用。

## 1.2 部署 Go 语言的开发环境

Go 语言主要支持 Windows、Linux 及 Mac OS 操作系统。本节将详细讲解在 Windows 操作系统中安装 Go 语言环境的具体过程。

Go 安装包的下载地址为 https://golang.google.cn/dl/。

Go 安装包的下载页面如图 1-1 所示。方框中标注的是官方推荐下载的版本，版本的描述如表 1-1 所示。

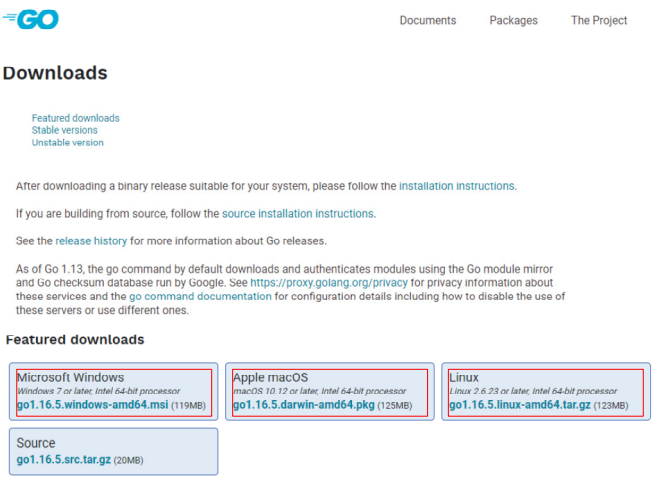


图 1-1 Go 安装包的下载页面

表 1-1　Go 安装包的命名及对应的操作平台

| 文　件　名 | 说　　明 |
| --- | --- |
| go1.16.5.windows-amd64.msi | Windows 操作平台及安装包 |
| go1.16.5.linux-amd64.tar.gz | Linux 操作平台及安装包 |
| go1.16.5.darwin-amd64.pkg | Mac OS 操作平台及安装包 |

## 1.2.1　Go 语言的环境变量

与 Java 等编程语言一样，安装 Go 语言开发环境需要设置全局的操作系统环境变量（除非使用包管理工具直接安装）。

主要的系统级别的环境变量有以下两个：

（1）GOROOT：表示 Go 语言环境在计算机上的安装位置，它的值可以是任意的位置，这个变量只有一个值，值的内容必须是绝对路径。

（2）GOPATH：表示 Go 语言的工作目录，可以有多个，类似于工作空间。一般不建议将 GOPATH 和 GOROOT 设置为同一个目录。

## 1.2.2　在 Windows 上安装 Go 语言环境

### 1. 下载

下载 Windows 版本的安装包 go1.16.5.windows-amd64.msi。Go 语言的 Windows 版本安装包的一般格式为 msi，可以直接安装到 Windows 系统中。

（1）1.16.5：表示 Go 语言安装包的版本。

（2）windows：表示这是一个 Windows 版本的安装包。

（3）amd64：表示匹配的 CPU 版本，这里匹配的是 64 位 CPU。

### 2. 安装

下载的 Windows 版本的 Go 语言安装包是一个可执行文件，直接双击进行安装即可。默认安装路径是 C 盘的 Go 目录下，直接单击 Next 按钮进行下一步，如图 1-2 所示。

出现图 1-3 所示的页面，直接单击 Install 按钮进行安装。

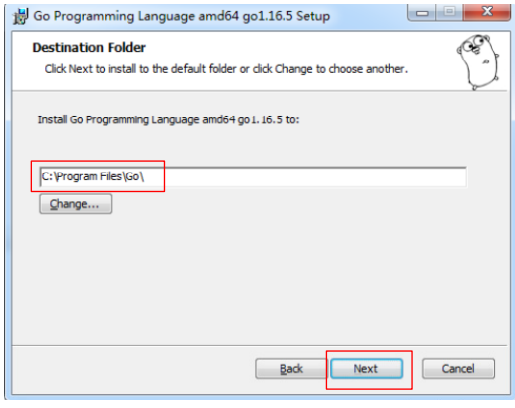


图 1-2　Windows 下 Go 安装包的安装目录的选择

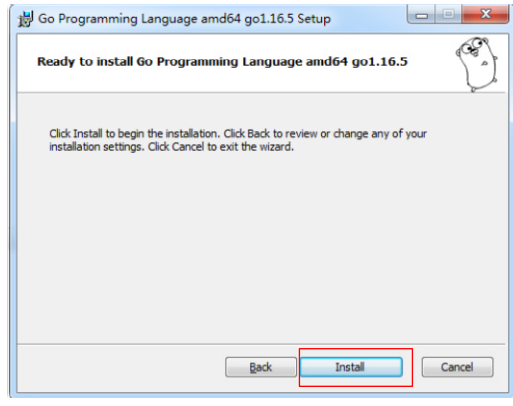


图 1-3　Windows 下 Go 安装包的安装

安装完成后，在安装路径 C 盘的 Go 目录下将生成一些目录文件，如图 1-4 所示。


图 1-4　Windows 下 Go 安装包的安装目录及文件

Go 安装包的安装目录及其说明如表 1-2 所示。

表 1-2　Go 安装包的安装目录及其说明

| 目　录　名 | 说　　明 |
|---|---|
| api | 每个版本的 api 变更差异 |
| bin | go 源码包编译出的编译器（go）、文档工具（godoc）、格式化工具（gofmt） |
| doc | 英文版的 Go 语言文档 |
| lib | 引用的库文件 |
| misc | 多项用途，如 Android 平台的编译、git 的提交钩子等 |
| pkg | Windows 平台编译完成的中间文件 |
| src | 标准库的源码 |
| test | 测试用例 |

### 3. 配置

Go 语言的安装包安装完成后需要配置环境变量才能正常使用。

右击“计算机”图标，在弹出的快捷菜单中选择“属性”命令，进入系统的控制面板主页，如图 1-5 所示。

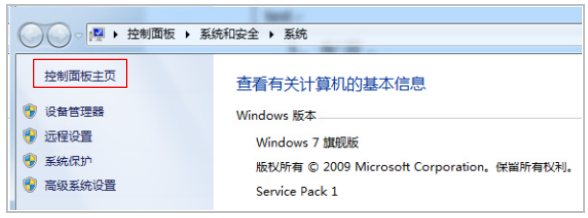


图 1-5　控制面板主页

在控制面板主页中单击“高级系统设置”选项，在弹出的对话框中单击“环境变量”按钮，弹出“环境变量”对话框，如图 1-6 所示。

在“系统变量”选项组中单击“新建”按钮，在“变量”文本框中输入 GOROOT，在“值”文本框中输入安装 Go 语言的路径，单击“确定”按钮，即系统变量配置完成，如图 1-7 所示。


图 1-6　“环境变量”对话框

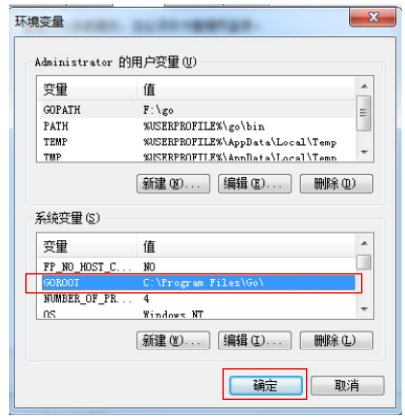


图 1-7　配置环境变量

另外，还要修改系统变量中的 PATH 变量，在变量值的最后添加“%%GOROOT\bin”路径，与其他 PATH 变量以“;”分隔，如图 1-8 所示。

环境变量配置完成后，还要查看环境变量是否全部配置正确。打开 cmd 终端，在终端中输入命令 go version，查看是否输出 Go 语言安装包的版本号，如果输出正确的版本号，则证明环境变量配置成功，如图 1-9 所示。

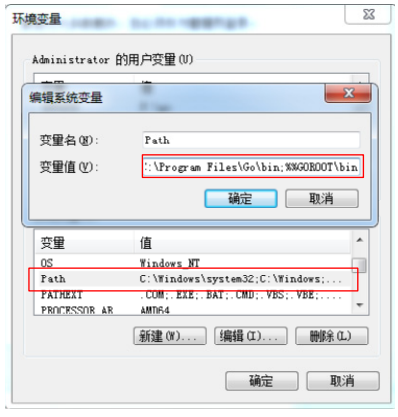


图 1-8　修改 PATH 变量

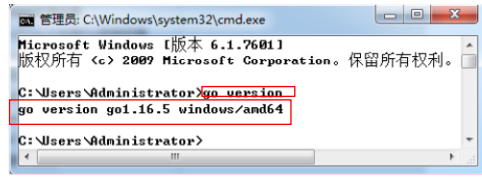


图 1-9　查看环境变量是否配置成功

## 1.2.3　在 Linux 上安装 Go 语言环境

首先，需要在图 1-1 所示的页面中下载 Linux 版本的安装包 go1.16.5.linux-amd64.tar.gz。

下载完成后，将该安装包解压到/usr/local/go 目录下，可以使用如下命令来完成：

```
tar -C /usr/local -xzf go1.16.5.linux-amd64.tar.gz
```

接着，需要将/usr/local/go/bin 目录添加到 PATH 环境变量中，可以使用如下命令来完成：

```
export PATH=$PATH:/usr/local/go/bin
```

最后，需要使用 go env 命令查看 Linux 版本的 Go 安装包是否安装成功。

## 1.2.4　在 Mac OS 上安装 Go 语言环境

在图 1-1 所示的页面中下载 Mac OS 版本的安装包 go1.16.5.darwin-amd64.pkg，双击安装包进行安装，

根据安装指引完成安装即可。Mac OS 版本的 Go 安装包默认安装到/usr/local/go 目录下。

Mac OS 设置变量的方法和 Linux 一样，都需要将/usr/local/go/bin 目录添加到 PATH 环境变量中，使用如下命令来完成：

```
export PATH=$PATH:/usr/local/go/bin
```

安装完成之后，使用 go version 命令查看 Mac OS 版本的 Go 安装包是否安装成功。

**注意：**如果 Mac OS 上之前已经安装过 Go 语言环境，则需要卸载原来的版本后再进行新版本的安装，即删除/etc/paths.d/go 文件。

# 1.3 Go 语言开发工具的使用

Go 语言和 Java、C/C++和 Python 等语言相比还显得较为“年轻”，成熟的 Go 语言集成开发工具也并不多，本节将介绍几个主要用于 Go 语言开发的工具。

## 1.3.1 LiteIDE

LiteIDE 是一款专门为 Go 语言开发的跨平台轻量级的集成开发环境，同时也是一个开源的工具。LiteIDE 支持主流的操作系统，如 Windows、Linux 及 Mac OS 操作系统等；还支持对 Go 语言的编译环境进行管理和切换，能够管理和切换多个 Go 语言编译环境，界面支持 Go 语言交叉编译。另外，LiteIDE 提供了一个基于 GOPATH 的包浏览器和一个基于 GOPATH 的编译系统，能够通过 API 文档检索相应信息。

在编辑 Go 语言程序方面，LiteIDE 拥有类浏览器和大纲显示功能，完美支持 Gocode（代码自动完成工具）与 Gdb 断点和调试，自动格式化代码。

本书就是采用 LiteIDE 集成开发工具完成各个程序及项目的开发，接下来将详细介绍 LiteIDE 集成开发工具的安装及配置。

LiteIDE 开发工具的下载地址为 https://sourceforge.net/projects/liteide/files/latest/download。

下载完成后，找到下载的 LiteIDE 压缩包进行解压，解压完成后，找到文件夹中的 liteide.exe 可执行文件，如图 1-10 所示，双击即可打开该工具进行使用。

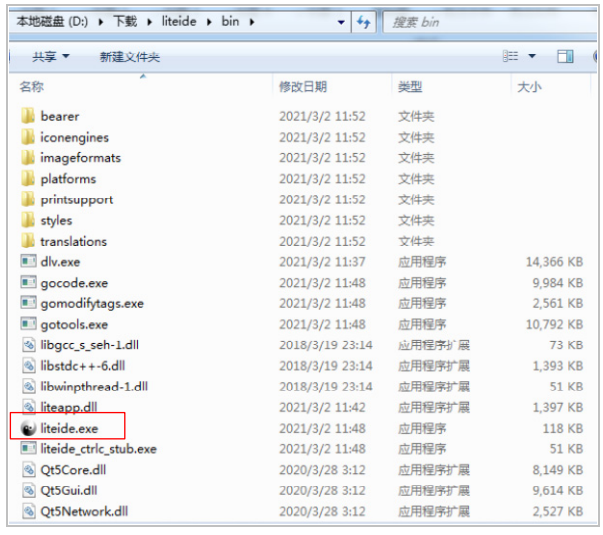


图 1-10　找到 liteide.exe 可执行文件

LiteIDE 的打开界面如图 1-11 所示。

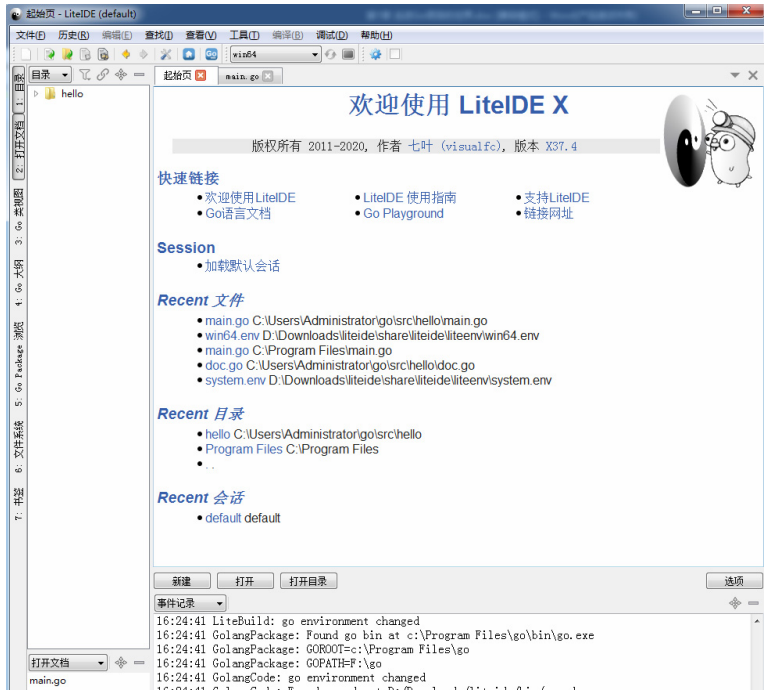


图 1-11　LiteIDE 界面

下载并安装 LiteIDE 集成开发工具之后，要想该开发工具能够正常使用，还需要进行环境的配置。

### 1. 配置环境

LiteIDE 提供了多种环境供开发者使用，目的是让开发者能够将程序编译成不同的系统所能执行的文件。例如，开发者使用的是 Windows 64 位系统，并且服务器也是 Windows 64 位，那么就需要选择 win64 的编译环境，这样在执行编译后，编译器将会自动生成在 Windows 中可执行的.exe 文件，如图 1-12 所示。

### 2. 编译当前环境

配置当前运行环境，单击“工具”按钮，在下拉菜单中选择“编辑当前环境”，在打开的 win64.env 文件中找到“GOROOT=xxx”，并将其修改为环境变量中 GOROOT 对应的值，如图 1-13 所示。

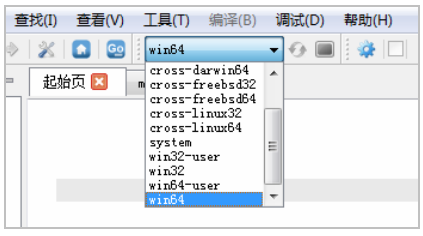


图 1-12　选择 LiteIDE 的系统环境

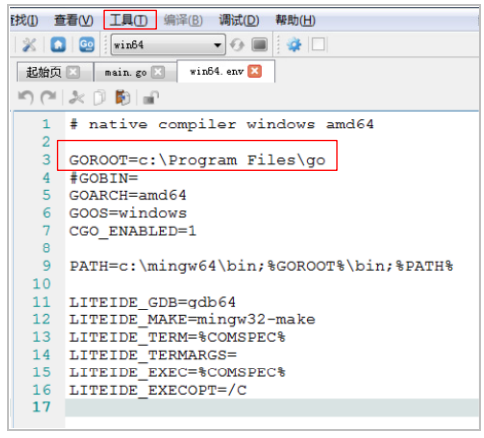


图 1-13　编辑当前环境

3. 配置管理 GOPATH/Modules/GOPROXY

在项目需要使用 GOPATH 或 Modules 时，可以通过单击图 1-14 所示的下三角按钮，on 表示使用 mod，off 表示不使用，auto 表示根据检测，有 mod 的情况可以使用。

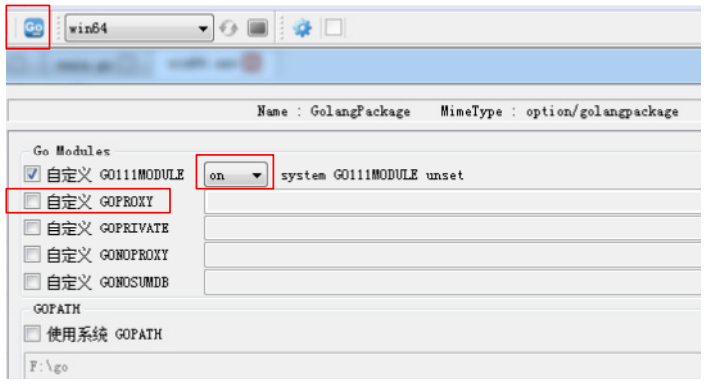


图 1-14　配置 GOPATH/Modules/GOPROXY

GOPROXY 可以设置代理，如果确定使用代理，则可勾选 GOPROXY 左边的复选框，在右边的文本框中输入代理名称即可，一般使用最多的是阿里云的代理，即 GOPROXY=https://mirrors.aliyun.com/goproxy/。

4. 添加工作空间

LiteIDE 一般通过使用 mod 来管理程序的运行，因此需要设置 GOPATH 工作空间。在配置管理 GOPATH/Modules/GOPROXY 的页面中单击“添加目录”按钮，将本地的 GOPATH 工作空间添加进去，如图 1-15 所示。如果有多个工作空间，可以添加多个。

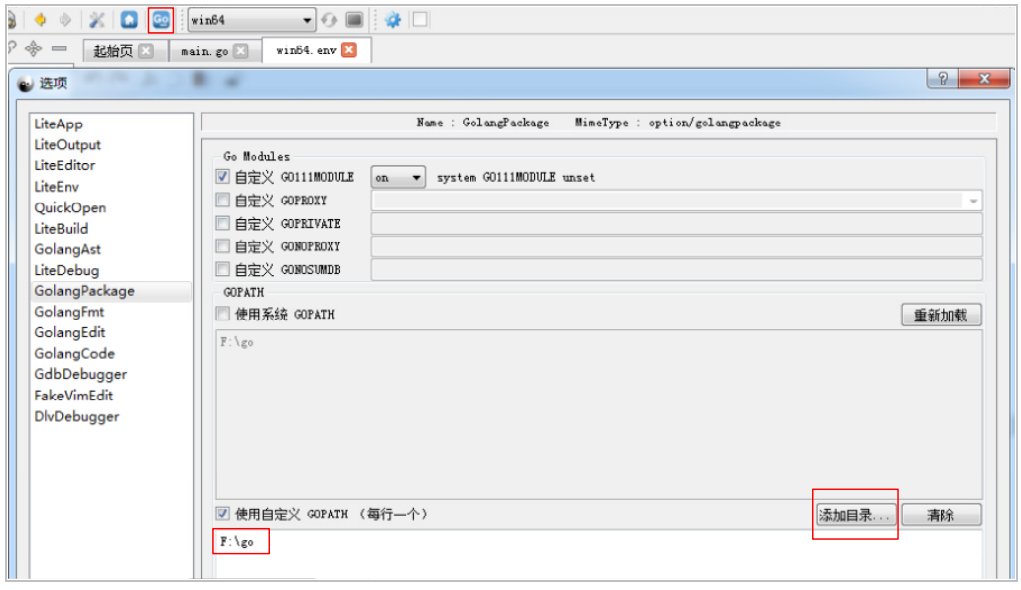


图 1-15　设置 GOPATH 工作空间

## 1.3.2　Gogland

Gogland 是由著名的 Jetbrains 公司专门为 Go 语言开发而设计的 IDE。Gogland 是 Jebrains 家族的一员，因此它也拥有 Jebrains 家族的传统特色。

Gogland 还处于开发阶段，并且还未发布正式的版本，但是开发者仍然可以在 Gogland 官网免费下载最新的测试版本。Gogland 集成了代码检查、自动补全、快速导航、格式化等编码助手与一系列集成工具，最重要的是还支持一系列的 IntelliJ 插件。

Gogland 的下载地址为 https://www.jetbrains.com/go/download，单击 DOWNLOAD 超链接即可下载，下载后解压，把文件夹移到适当的位置，最后执行 bin 目录中的 gogland.sh 脚本即可启动。

由于 Gogland 是 Jebrains 家族的一员，所以，要使 Gogland 能够正常工作，必须安装 Java 并配置 JDK 或 JRE 环境，同时还需要手动配置 GOPATH，目前该软件没有汉化版本。Gogland 的运行界面如图 1-16 所示。

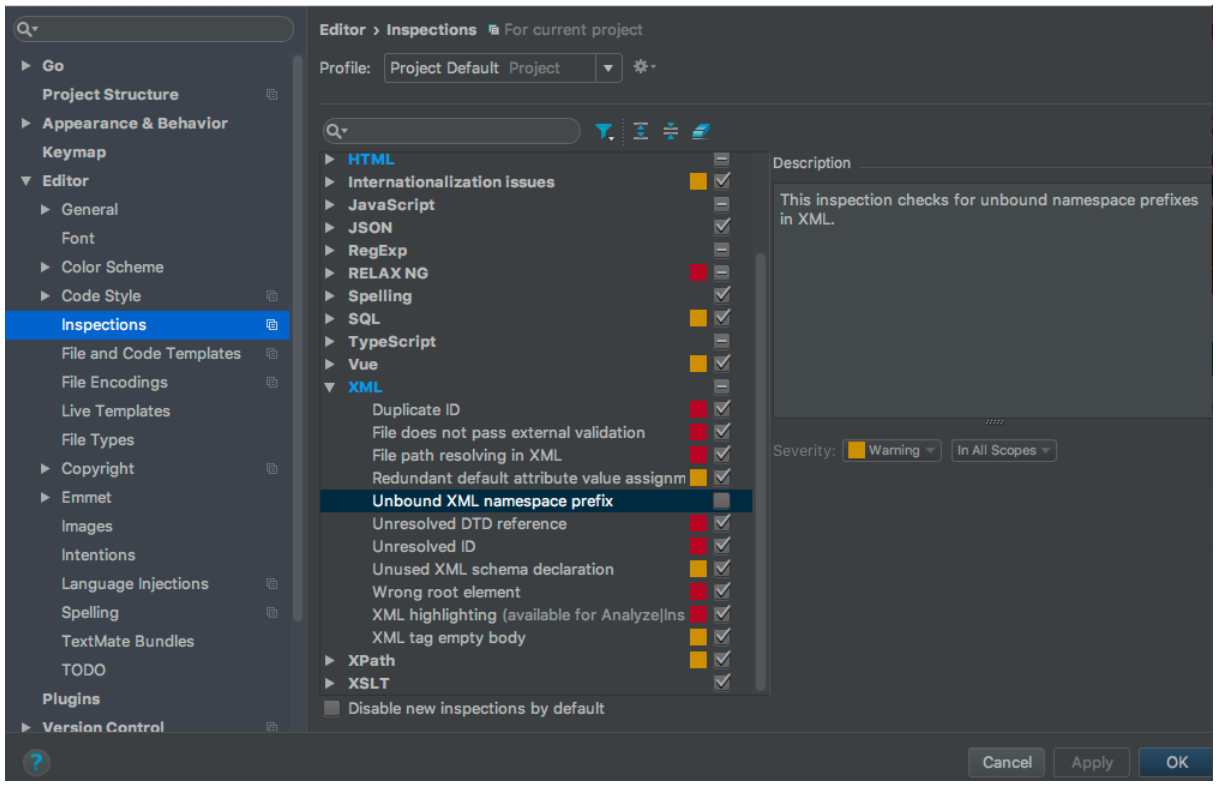


图 1-16　Gogland 的运行界面

## 1.3.3　Visual Studio Code

Visual Studio Code（简称 VS Code）是一款由微软公司开发的，能运行在 Windows、Linux 及 Mac OS 上的跨平台开源代码编辑器。

VS Code 使用 JSON 格式的配置文件进行所有功能和特性的配置。VS Code 可以通过扩展程序为编辑器实现编程语言高亮、参数提示、编译、调试、文档生成等各种功能。

**1. 切换语言**

中文版的 VS Code 下载后的命令语言都是中文的，因此搜索英文指令变得十分困难，这里可以将 VS Code 的语言切换为英文。

选择 VS Code 的菜单“查看”命令，在打开的面板中输入“配置语言”，弹出如图 1-17 所示的页面。

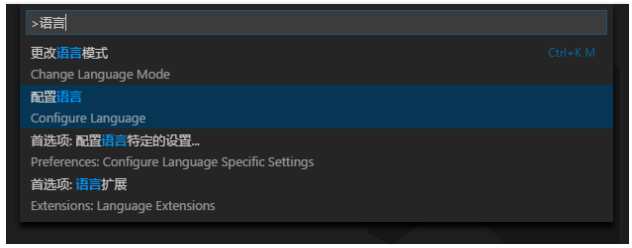


图 1-17　VS Code 中打开配置语言面板

输入信息后按 Enter 键打开 local.json 文件，如图 1-18 所示。将“zh-CN”修改为“en-US”，关闭 VS Code 后再重新打开，语言修改就生效了。

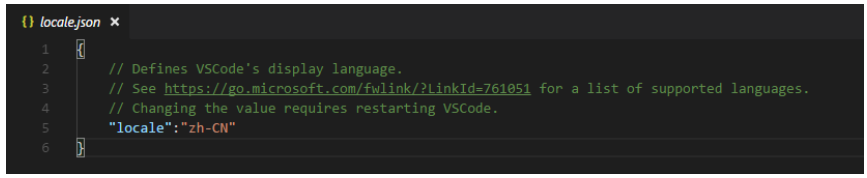


图 1-18　VS Code 中修改显示语言

### 2. 安装 Go 语言扩展

选择 View|Extensions 命令，打开扩展面板，如图 1-19 所示。

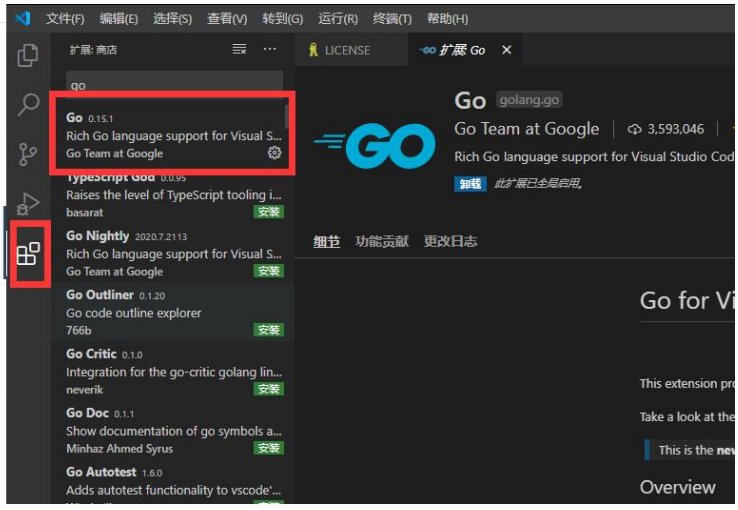


图 1-19　Code 安装 Go 语言插件

在搜索框中输入 Go，找到 Rich Go language support for Visual Studio Code 字样的扩展，单击 Install 按钮，即可安装 Go 语言扩展，然后打开项目源文件，此时 Code 检测到打开的是 Go 语言源代码，便会检查 Go 语言开发工具链是否完整。如果是新手，一般都会不完整，此时 Code 会提醒安装其他 Go 语言插件，这些插件实际上是通过 go get 命令安装的，Code 会把这些工具下载到 SGOPATH 的第一个值的路径中。

## 1.4　Go 语言的目录结构

Go 语言的目录结构包括开发包目录（GOROOT）和工作区目录（GOPATH）。

### 1.4.1　GOROOT 结构

GOROOT 是 Go 语言环境的根目录，打开 Go 语言安装包的安装路径即可看到 GOROOT 目录结构中的内容，如图 1-20 所示。

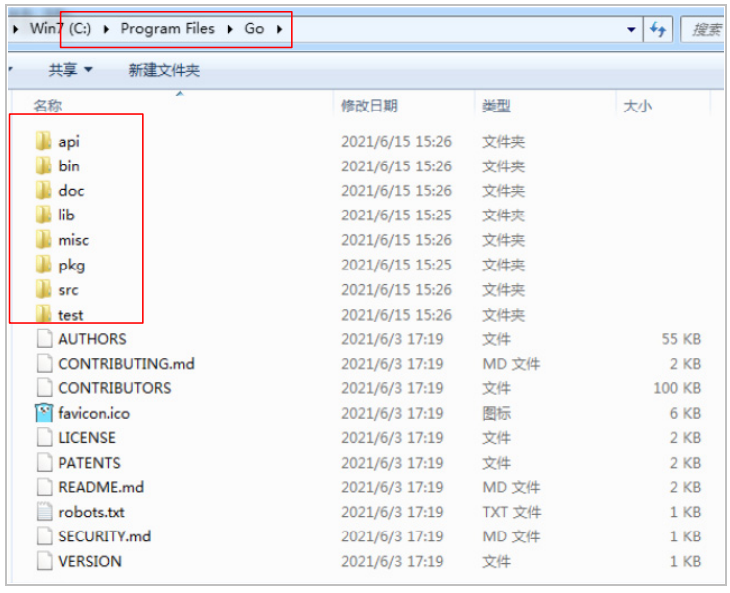


图 1-20　GOROOT 目录结构

1. api 文件夹

api 文件夹中存放了 Go API 检查器的辅助文件，包括公开的变量、常量及函数等。其中，go1.1.txt、go1.2.txt、go1.3.txt 和 go1.txt 文件分别存放了不同版本的 Go 语言的全部 API 特征；except.txt 文件中存放了一些（在不破坏兼容性的前提下）可能会消失的 API 特性；next.txt 文件则存放了可能在下一个版本中添加的新 API 特性，如图 1-21 所示。

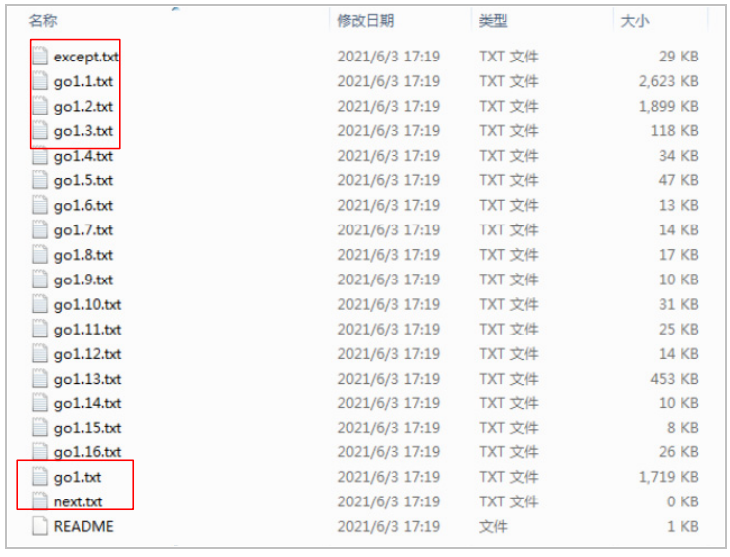


图 1-21　api 文件夹中存放的内容

### 2. bin 文件夹

bin 文件夹中存放了所有由官方提供的 Go 语言相关工具的可执行文件。默认情况下，该目录会包含 go 和 gofmt 这两个工具，如图 1-22 所示。


图 1-22　bin 文件夹中存放的内容

### 3. doc 文件夹

doc 文件夹中存放了 Go 语言几乎全部 HTML 格式的官方文档和说明，方便开发者在离线时查看，如图 1-23 所示。


图 1-23　doc 文件夹中存放的内容

### 4. lib 文件夹

lib 文件夹中存放引用的库文件，可以为程序的运行提供帮助，如图 1-24 所示。


图 1-24　lib 文件夹中存放的内容

### 5. misc 文件夹

misc 文件夹中存放各类编辑器或 IDE（集成开发环境）软件的插件，辅助开发者查看和编写 Go 语言代码，如图 1-25 所示。

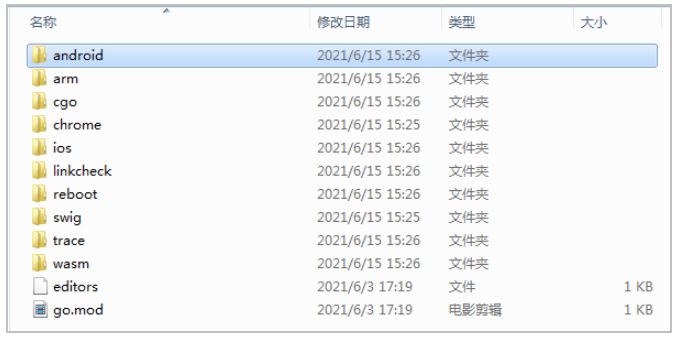


图 1-25　misc 文件夹中存放的内容

### 6. pkg 文件夹

pkg 文件夹用于在构建安装后，保存 Go 语言标准库的所有归档文件。pkg 文件夹包含一个与 Go 语言安装平台相关的子目录，被称为“平台相关目录”。例如，Windows 64bit 操作系统的安装包中，平台相关目录的名字则为 windows_amd64。Go 源码文件对应于以“.a”为结尾的归档文件，存储在 pkg 文件夹下的

平台相关目录中。

pkg 文件夹下还有一个名为 tool 的子文件夹，该子文件夹下也有一个平台相关目录，其中存放了很多可执行文件，如图 1-26 所示。

### 7. src 文件夹

src 文件夹中存放了所有的标准库、Go 语言工具及相关底层库（C 语言实现）的源码。通过查看 src 文件夹，可以了解 Go 语言的方方面面，如图 1-27 所示。


图 1-26　pkg 文件夹中存放的内容

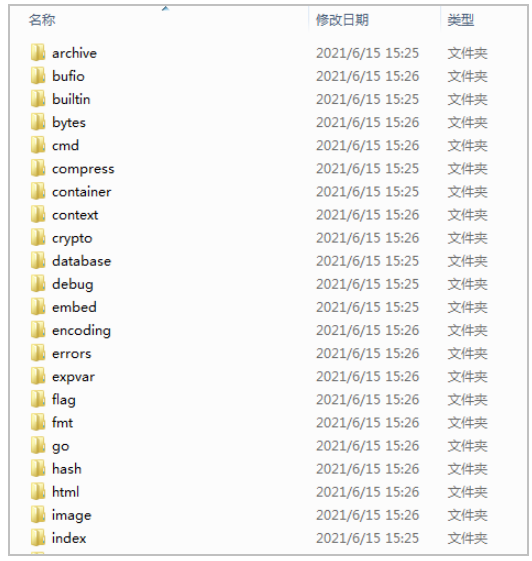


图 1-27　src 文件夹中存放的内容

### 8. test 文件夹

test 文件夹中存放了测试 Go 语言自身代码的文件。通过阅读这些测试文件，可以了解 Go 语言的特性和使用方法，如图 1-28 所示。

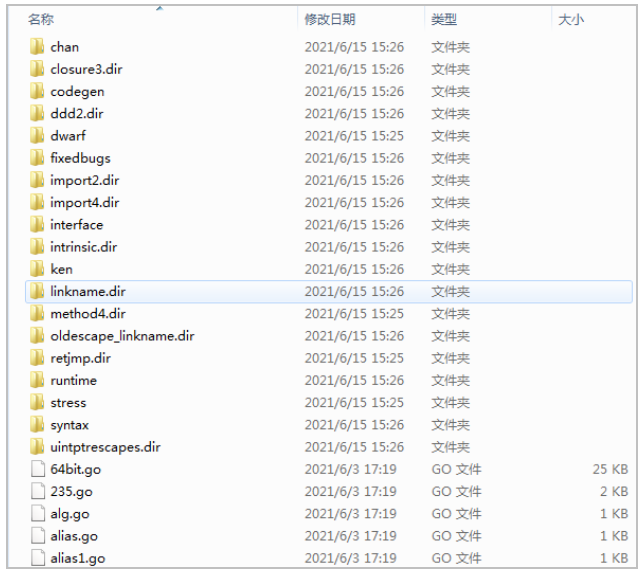


图 1-28　test 文件夹中存放的内容

### 1.4.2 GOPATH 结构

GOPATH 工作区有 3 个子目录，分别是 src 目录、pkg 目录和 bin 目录。

#### 1. src 目录

src 目录主要用于以代码包的形式组织并保存 Go 源码文件。代码包与 src 的子目录相对应。例如，若一个源码文件被声明为属于代码包 logging，那么它就应当被保存在 src 目录下名为 logging 的子目录中。当然，也可以把 Go 源码文件直接放在 src 目录下，但这样的 Go 源码文件就只能被声明为属于 main 的代码包。一般建议把 Go 源码文件放入特定的代码包中。

**注意：**Go 语言的源码文件分为 3 类，即 Go 库源码文件、Go 命令源码文件和 Go 测试源码文件。Go 语言的命令源码文件和库源码文件的区别如下：所谓命令源码文件，就是声明为属于 main 代码包，并且包含无参数声明和结果声明的 main 函数的源码文件。这类源码文件可以独立运行（使用 go run 命令），也可被 go build 或 go install 命令转换为可执行文件。库源码文件则是指存在于某个代码包中的普通源码文件。

#### 2. pkg 目录

pkg 目录主要用于存放经由 go install 命令构建安装后的代码包（包含 Go 库源码文件）的“.a”归档文件。该目录与 GOROOT 目录下的 pkg 功能类似。区别在于，GOPATH 结构中的 pkg 目录专门用来存放程序开发者代码的归档文件。构建和安装用户源码的过程一般会以代码包为单位进行，例如，logging 包被编译安装后，将生成一个名为 logging.a 的归档文件，并存放在当前工作区的 pkg 目录下的平台相关目录中。

#### 3. bin 目录

bin 目录与 pkg 目录类似，在通过 go install 命令完成安装后，保存由 Go 命令源码文件生成的可执行文件。在 Linux 操作系统下，这个可执行文件一般是一个与源码文件同名的文件。而在 Windows 操作系统下，这个可执行文件的名称是源码文件名称加.exe。

## 1.5 第一个 Go 语言程序

#### 1. 新建项目

双击打开安装完成的 LiteIDE，在“文件”菜单中选择“新建”命令，弹出“新项目或文件”对话框，选择系统默认的 GOPATH 路径，“模板”选择 Go1 Command Project，最后输入项目名称，并选择合适的目录存储，确认无误后单击 OK 按钮，新项目创建完成，如图 1-29 所示。

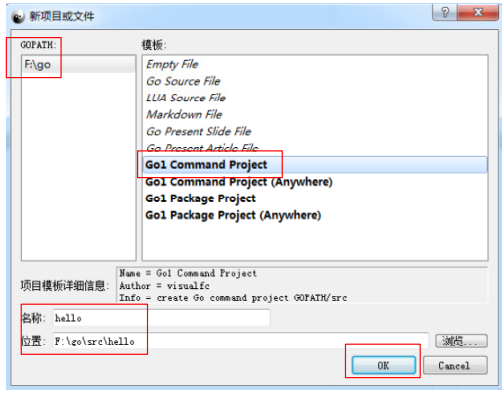


图 1-29 新建项目

### 2. 了解项目的结构

新建完成的项目如图 1-30 所示，编辑器自动创建了两个文件，并在 main.go 中生成了简单的代码。


图 1-30　新建完成的项目

图 1-30 所示的程序分为如下三个部分：

第一部分是包归属，即 package main。package 是 Go 语言的一个关键字，用于定义当前代码所属的包，与 Java 中的 package 类似，作为模块化的标识。main 包是一个特殊的包名，它表示当前是一个可执行程序，而不是一个库。所有的 Go 源程序文件头部必须有一个包声明语句。

第二部分是 import，称为包的导入。import 也是 Go 语言的一个关键字，表示要导入的代码包，与 Java 中的 import 关键字类似，导入后就可以使用。Go 语言要求只有用到的包才能导入，如果导入一个代码包又不使用，那么编译时会报错。fmt 是这个程序导入的一个包的包名，fimt 是标准库中的一个包，也是标准的输入/输出包，导入之后，就可以使用它的函数了。

第三部分是程序的主体。func 是一个函数定义关键字；main 是程序主函数，表示程序执行的入口；fmt.Println 是 fmt 包中的函数，其调用方法与其他语言类似，这个程序用于输出一段文字。Go 语言默认不需要分号结束每行代码，如果两行代码写在一行时才需要使用分号分隔，但是不建议这样书写代码。

### 3. 项目的运行

在运行项目时，需要先编译源码再运行程序，可以单击工具栏上蓝色的编译执行按钮 BR 运行代码。也可以选择菜单栏中的 Build 选项先编译源码，再选择菜单栏中的 Run 命令运行程序。如果运行了该项目，可以在底部的编译输出（Build Output）窗口中看到项目运行的结果，如图 1-31 所示。

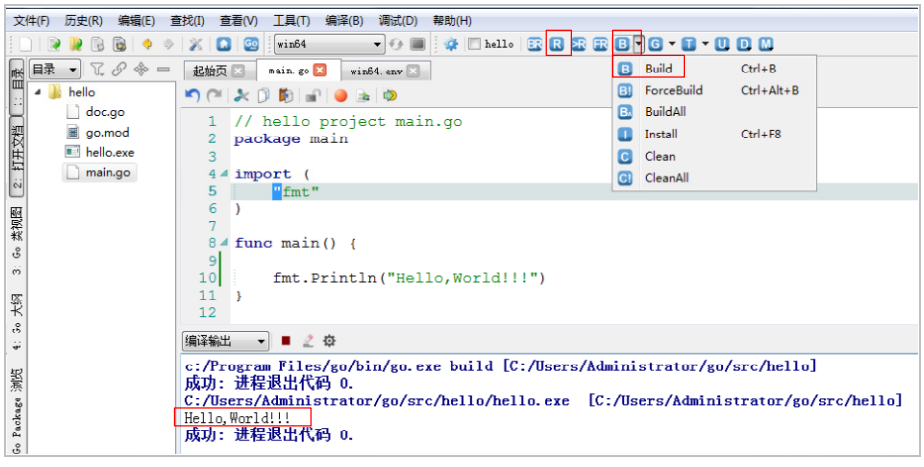


图 1-31　运行结果

LiteIDE 中运行程序的方式有如下两种：

①BR（BuildAndRun）是编译并运行整个项目，可以使用 Command + R 快捷键。

②FR（FileRun）是编译并运行单个文件，可以使用 Shift + Alt + R 快捷键。

编译运行单个文件和编译运行整个项目的区别如下：

①编译运行整个项目时，只允许一个源文件中有 main 函数。

②编译运行单个文件时，允许多个源文件包含 main 函数，运行时并不会报错。

#### 4. Go 语言程序规则

①源程序以.go 为扩展名。

②源程序默认为 UTF-8 编码。

③标识符区分大小写。

④语句结尾的分号可以省略。

⑤函数以 func 开头，函数体开头的“{”必须在函数头所在的行尾部，不能单独起一行。

⑥字符串字面量使用“""”（双引号）括起来。

⑦调用包中的方法通过“.”访问符，如 fmt.Printf。

⑧main 函数所在的包名必须是 main。

## 1.6 就业面试技巧与解析

本章首先通过 Go 语言的诞生、Go 语言的特性及 Go 语言的使用等基础知识带领读者认识 Go 语言。接着通过在 Windows、Linux 和 Mac OS 上部署 Go 语言的开发环境及安装 Go 语言的开发工具，帮助读者学会搭建 Go 语言开发环境的方法。最后，通过简单的一个 Go 语言程序，让读者了解 Go 语言的程序结构及目录结构。

学习完本章内容，我们对 Go 语言的了解有多少呢？下面一起来检验一下吧！

### 1.6.1 面试技巧与解析（一）

**面试官：**什么是 Go 语言？Go 语言的特性及使用方法有哪些？

**应聘者：**

Go 语言是一门新型的静态类型的编译型语言，它的出现受到程序开发人员的广泛喜爱。使用 Go 语言不仅访问底层操作系统，还提供了强大的网络编程和并发编程支持。Go 语言的用途众多，可以进行网络编程、系统编程、并发编程、分布式编程等。

Go 语言的特性有以下几点：

①语法简单。

②工程结构简单。

③有垃圾回收机制。

④可以进行快速编译。

⑤拥有功能完善、质量可靠的标准库。

Go 使用编译器来编译代码。编译器将源代码编译成二进制（或字节码）格式；在编译代码时，编译器检查错误、优化性能并输出可在不同平台上运行的二进制文件。要创建并运行 Go 程序，程序员必须执行如下步骤：

①使用文本编辑器创建 Go 程序。
②保存文件。
③编译程序。
④运行编译得到的可执行文件。

## 1.6.2　面试技巧与解析（二）

**面试官：**通过学习 Go 程序，你对程序中的 main 函数了解多少？

**应聘者：**
①main 是程序主函数，表示程序执行的入口。
②main 函数不能带参数。
③main 函数不能定义返回值。
④main 函数所在的包必须为 main 包。
⑤main 函数中可以使用 flag 包来获取和解析命令行参数。
⑥所有的 Go 源程序文件头部必须有一个包声明语句。

# 第 2 章

# Go 语言程序元素的构成

## 本章概述

在学习 Go 语言的核心语法之前，先了解一下 Go 语言程序元素的构成。本章主要学习 Go 语言的词法单元、常量及变量的概念和使用方法，通过对 Go 语言程序元素的学习，为读者学习核心的 Go 语言知识打下坚实的基础。

## 知识导读

本章要点（已掌握的在方框中打钩）：

- ☐ 词法单元。
- ☐ 常量。
- ☐ 变量。

## 2.1 词法单元

Go 语言的词法单元包括标识符、关键字、字面量、分隔符、运算符及注释，它们是组成 Go 语言代码和程序的最基本的单位。学习 Go 语言的词法单元能够帮助程序员更好地掌握 Go 语言的语法结构，下面将依次进行介绍。

### 2.1.1 标识符

标识符是一种字符序列，在 Go 语言中可以使用字符序列，对各种变量、常量、类型、方法、函数等进行命名。标识符由若干个字母、下画线和数字组成，且第一个字符必须是字母。简单来说，凡可以自己定义的名称都可以称为标识符。

标识符有以下两种类型：

**1. 自定义标识符**

自定义标识符是指程序员在编程过程中自行定义的变量名、常量名、函数名等一切符合语言规范的标识符。

**注意：**用户自定义的标识符不应该使用语言设计者的预定义标识符，这样会导致代码语句有歧义，严重影响代码的可读性。

**2. 预定义标识符**

预定义标识符是指由程序设计者在 Go 语言源代码中事先声明的标识符。预定义标识符包括语言的预声明标识符，以及用于后续语言扩展的保留字等。预定义标识符如表 2-1 所示。

表 2-1　预定义标识符

| 分　类 | 名　称 |
| --- | --- |
| 数据类型 | bool（true 和 false）、byte、uint16、float32、float64、int、int8、int16、uint32、int32、int64、uint64、string、uint、uint8、uintptr |
| 内建函数 | append、cap、close、complex、copy、delete、imag、len、make、new、panic、print、println、real、recover |
| 其他标识符 | iota、nil、_ |

表 2-1 中有一个特殊的标识符，即下画线。下画线也被称为空标识符，它可以用于变量的声明或赋值（任何类型都可以赋值给它），但任何赋给这个标识符的值都将被抛弃，因此这些值不能在后续的代码中使用，也不可以使用下画线作为变量对其他变量进行赋值或运算。

标识符的命名规则需要遵循以下几点：

（1）标识符由 26 个英文字母、0～9 数字及下画线组成。

（2）标识符开头的一个字符必须是字母或下画线，后面跟任意多个字符、数字或下画线。标识符不能以数字开头，例如，int 5abc 是错误的。

（3）标识符在 Go 语言中严格区分大小写，例如，Test 和 test 在 Go 语言程序中表示两个不同的标识符。

（4）标识符不能包含空格。

（5）在 Go 语言中不允许标识符使用标点符号，如@、$、%等一系列符号。

（6）不能以系统保留关键字作为标识符，如 break、if 等。

```
5d      //这不是一个合法标识符,不是以字母或下画线开头的
$ab     //这不是一个合法标识符,不是以字母或下画线开头的
abc     //这是一个合法标识符
_aa     //这是一个合法标识符
abc5    //这是一个合法标识符
```

标识符的命名还需要注意以下几点：

（1）标识符的命名要尽量简短且有意义。

（2）不能和标准库中的包名重复。

（3）为变量、函数、常量命名时采用驼峰命名法，如 stuName、getVal。

**注意：**在使用标识符之前必须进行声明，声明一个标识符就是将这个标识符与常量、类型、变量、函数或者代码包绑定在一起。在同一个代码块内标识符的名称不能重复。

## 2.1.2　关键字

关键字是指语言设计者保留的具有特定语法含义的标识符，也可以称为保留字。关键字主要用来控制程序的结构，每个关键字都代表着不同语义的语法。Go 语言中的关键字一共有 25 个，如表 2-2 所示。

**注意：**关键字不能用作常量、变量或任何其他标识符名称，也不能再声明和关键字相同的标识符。

表 2-2　关键字

| 关　键　字 | 说　　明 |
| --- | --- |
| package | 用于定义包名 |
| import | 用于导入包名 |
| const | 用于常量声明 |
| var | 用于变量声明 |
| func | 用于函数定义 |
| defer | 用于延迟执行 |
| go | 用于并发语法 |
| return | 用于函数返回 |
| struct | 用于定义结构类型 |
| interface | 用于定义接口类型 |
| map | 用于声明或创建 map 类型 |
| chan | 用于声明或创建通道类型 |
| if else | 用于 if else 语句 |
| for change break continue | 用于 for 循环 |
| switch select type case default fallthrough | 用于 switch 和 select 语句 |
| goto | 用于 goto 跳转语句 |

## 2.1.3　字面量

字面量代表着一种标记法，常用来表示一些固定值。但在 Go 语言中，字面量的含义有多种，既可以表示基础数据类型值的各种字面量，又可以表示程序员构造的自定义符合数据类型的类型字面量，还可以表示符合数据类型的值的符合字面量。

使用字面量时，一般使用裸字符序列来表示不同类型的值。字面量还可以被编译器直接转换为某个类型的值。

字面量可以分为以下几类：

### 1. 整型字面量

整型字面量就是使用特定的字符序列来表示具体的整型数值，它常被用于整型变量或常量的初始化，例如：

```
55
0550
0xHello
123456789
```

### 2. 浮点型字面量

浮点型字面量就是使用特定的字符序列来表示一个浮点数值。浮点型字面量的表示方法有两种：数学记录法和科学计数法，例如：

```
25.12
0.15
.25
.12345F+4
1.e+0
```

**3. 复数类型字面量**

复数类型字面量就是使用特定的字符序列来表示复数类型的常量值，例如：

```
0i
5.e+0i
.25i
.12345F+4i
1.e+0i
```

**4. 字符型字面量**

字符型字面量就是使用特定的字符序列来表示字符型的数值。Go 语言程序的源码通常采用 UTF-8 的编码方式，UTF-8 的字符占用 1～4B。Rune 字符常量也有多种表现形式，但通常使用“' '”（单引号）将字符常量括起来，例如：

```
'b'
'\t'
'\0000'
'\abc'
'\u45e3'
```

**5. 字符串字面量**

字符串字面量通常使用“" "”（双引号）将字符序列括起来，双引号中的内容可以是 UTF-8 的字符字面量，也可以是其编码值，例如：

```
"\n"
"hello world!\n"
"Go 语言"
```

## 2.1.4 分隔符

分隔符主要用来分隔其他元素，例如：

```
fmt
.
Println
(
"hello,world!"
)
```

在以上程序中使用了小数点、括号、冒号及逗号等分隔符。

括号分隔符包括括号、中括号和大括号。

标点分隔符包括小数点、逗号、分号、冒号和省略号。

无论哪种编程语言，程序代码都是通过语句来实现结构化的，但是 Go 语言不需要以分号进行结尾，结尾的工作都是由 Go 语言的编译器自动来完成的。当多个语句写在同一行时，它们就必须使用“;”进行分隔，但在实际开发中不提倡这种写法。

## 2.1.5 运算符

运算符主要用于在程序运行时执行运算或逻辑操作。Go 语言中的运算符如表 2-3 所示。

表 2-3 Go 语言运算符及说明

| 运 算 符 | 说 明 |
| --- | --- |
| \|\| | 逻辑或，二元逻辑运算符 |

续表

| 运 算 符 | 说 明 |
| --- | --- |
| && | 逻辑与，二元逻辑运算符 |
| ! | 逻辑非，一元逻辑运算符 |
| == | 相等判断，二元逻辑运算符 |
| != | 不等判断，二元逻辑运算符 |
| < | 小于判断，二元逻辑运算符 |
| <= | 小于或等于判断，二元逻辑运算符 |
| > | 大于判断，二元逻辑运算符 |
| >= | 大于或等于判断，二元逻辑运算符 |
| + | 求和，二元算术运算符 |
| - | 求差，二元算术运算符 |
| * | 求积，二元算术运算符 |
| / | 求商，二元算术运算符 |
| \| | 按位或，二元算术运算符 |
| ^ | 按位异或，二元算术运算符 |
| % | 求余，二元算术运算符 |
| << | 按位左移，二元算术运算符 |
| >> | 按位右移，二元算术运算符 |
| & | 按位与，二元算术运算符 |
| &^ | 按位清除，二元算术运算符 |

在 Go 语言中，一个表达式可以包含多个运算符。当表达式中存在多个运算符时，就会遇到优先级的问题，此时应该先处理哪个运算符由 Go 语言运算符的优先级来决定。

所谓优先级，就是当多个运算符出现在同一个表达式中时，先执行哪个运算符。二元运算符的优先级如表 2-4 所示。

**表 2-4 二元运算符的优先级**

| 运 算 符 | 优 先 级 |
| --- | --- |
| *、/、%、<<、>>、&、&^ | 最高 |
| +、-、^、\| | 较高 |
| ==、!=、<、<=、>、>= | 中 |
| && | 较低 |
| \|\| | 最低 |

当表达式中出现相同优先级的运算符时，可以根据从左到右的顺序依次执行操作。当遇到括号时，括号的优先级是最高的，括号中的表达式会优先执行。

**注意：**在 Go 语言中，++和—是语句，不是表达式，没有运算符优先级之说。

### 2.1.6 注释

注释是源代码中最重要的组成部分之一。当 Go 语言的编译器遇到标有注释的代码时，编译器会自动识别并跳过标有注释的代码。

注释不会被编译，每一个包都应该有相关注释。注释分为单行注释和多行注释。

#### 1. 单行注释

单行注释是最常见的注释形式，程序员可以在代码中的任何地方使用以“//”开头的单行注释，例如：

```
fmt.Println("hello,world!")    //单行注释
```

#### 2. 多行注释

多行注释又称块注释，多行注释通常以“/*”进行开头，并以“*/”进行结尾，例如：

```
func main() {
    /*多行注释
    var stockcode = 123
    var enddate = "2021-07-02"
    var url = "Code=%d&endDate=%s"
    var target_url = fmt.Sprintf(url, stockcode, enddate)
    */
    fmt.Println(target_url)
}
```

## 2.2 常量

Go 语言中的常量使用关键字 const 定义，用于存储不会改变的数据，常量是在编译时被创建的，即使定义在函数内部也是如此，并且只能是布尔型、数值型（整型、浮点型和复数类型）和字符串型。由于编译时的限制，定义常量的表达式必须为能被编译器求值的常量表达式。

### 2.2.1 常量的定义

常量是在程序运行时不会被修改的量。常量的定义格式如下：

```
const identifier [type] = value
```

例如：

```
const Pi = 3.1415926           //可以指定常量的类型
const Pi float32=3.1415926     //也可以是布尔值、字符串等
const hello="Go 语言"          //还可以使用中文
```

在 Go 语言中，由于编译器可以根据变量的值来推断其类型，因此可以省略类型说明符 [type]。

Go 语言中常量的定义包括显式定义和隐式定义两种。

（1）显式定义如下：

```
const a string = "apple"
```

（2）隐式定义如下：

```
const a = "apple"
```

Go 语言的常量还可以是十进制、八进制或十六进制的常数。

①十进制无前缀。

②前缀为 0 的是八进制。

③前缀为 0x 或 0X 的是十六进制。

④另外，整数还可以有一个后缀，其后缀可以是大小写字母（顺序任意）。后缀通常是 U 和 L 的组合，通常用于表示 unsigned 和 long，例如：

```
3.1415926     //十进制,合法
0215          //八进制,合法
0x25          //十六进制,合法
30u           //无符号整型,合法
25l           //long,合法
25ul          //无符号 long,合法
```

## 2.2.2 常量的声明

常量的声明和变量的声明类似，只是把 var 换成了 const。多个变量可以一起声明，同样，多个常量也可以一起声明，例如：

```
const (
    e = 2.7182
    pi = 3.1415
)
```

在 Go 语言中，所有常量的运算都可以在编译期来完成，这样不仅可以减少运行时的工作，也方便其他代码的编译优化。当操作数是常量时，运行时经常出现的错误也可以在编译时被发现，例如，整数除零、字符串索引越界、任何导致无效浮点数的操作等。

常量间的所有算术运算、逻辑运算和比较运算的结果也是常量，对常量的类型转换操作或函数调用（len、cap、real、imag、complex 和 unsafe.Sizeof）都是返回常量结果。

由于常量的值是在编译期就确定的，因此常量可以用于数组的声明。

```
const size =5
var arr [size] int
```

如果是批量声明的常量，除了第一个常量外，其他的常量右边的初始化表达式都可以省略。如果省略初始化表达式，则表示使用前面常量的初始化表达式，对应的常量类型也是一样的，例如：

```
const (
    a = 3
    b
    c = 5
    d
)
fmt.Println(a, b, c, d) //"3 3 5 5"
```

在 Go 语言中有一个特殊的常量——iota。iota 是一个可以被编译器修改的常量。

iota 在 const 关键字出现时将被重置为 0，const 中每新增一行常量声明将使 iota 计数一次。

iota 可以被用作枚举值，例如：

```
const (
    a = iota
    b = iota
    c = iota
)
```

第一个 iota 等于 0，每当 iota 在新的一行被使用时，它的值都会自动加 1；所以 a=0, b=1, c=2 可以简写为如下形式：

```
const (
    a = iota
    b
    c
)
```

## 2.2.3 转义字符

与其他语言一样，Go 语言也使用反斜杠表示转义字符。常用的转义字符及其含义如表 2-5 所示。

表 2-5 常见的转义字符及其含义

| 转 义 字 符 | 含 义 |
| --- | --- |
| \\ | \字符 |
| \' | '字符 |
| \" | "字符 |
| \? | ?字符 |
| \b | 退格 |
| \f | 换页 |
| \n | 新行 |
| \r | 回车 |
| \t | 水平制表符 |
| \v | 垂直制表符 |

转义字符的使用，例如：

```
package main
import "fmt"
func main() {
    fmt.Println("hello\tworld!")
    fmt.Println("hello\rworld!")
}
```

在 liteIDE 开发工具中编译并执行以上代码，运行结果如图 2-1 所示。

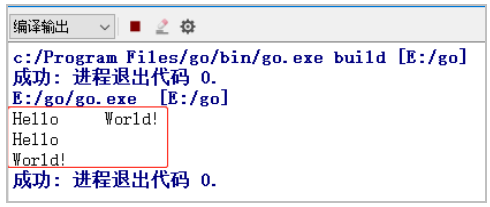


图 2-1 转义字符的运行结果

## 2.2.4 赋值

Go 语言支持直接赋值运算符、相加和赋值运算符、相减和赋值运算符、相乘和赋值运算符、相除和赋值运算符、左移和赋值运算符、右移和赋值运算符、按位与和赋值运算符、按位异或和赋值运算符、按位或和赋值运算符等。

```
package main
import (
    "fmt"
)
func main() {
    a := 10
    c := 20
    c = a
    fmt.Println("赋值操作,把 a 赋值给 c,所以 c 的值为: ", c)
    c += a
    fmt.Println("相加和赋值运算符,实际为 c=c+a,所以 c 的值为: ", c)
    c -= a
    fmt.Println("相减和赋值运算符,实际为 c=c-a,所以 c 的值为: ", c)
    c *= a
    fmt.Println("相乘和赋值运算符,实际为 c=c*a,所以 c 的值为: ", c)
    c /= a
    fmt.Println("相除和赋值运算符,实际为 c=c/a,所以 c 的值为: ", c)
    c <<= 2
    fmt.Println("左移和赋值运算符,所以 c 的值为: ", c)
    c >>= 2
    fmt.Println("右移和赋值运算符,所以 c 的值为: ", c)
    c &= 2
    fmt.Println("按位与和赋值运算符,所以 c 的值为: ", c)
    c ^= 2
    fmt.Println("按位异或和赋值运算符,所以 c 的值为: ", c)
    c |= 2
    fmt.Println("按位或和赋值运算符,所以 c 的值为: ", c)
}
```

编译运行结果如图 2-2 所示。

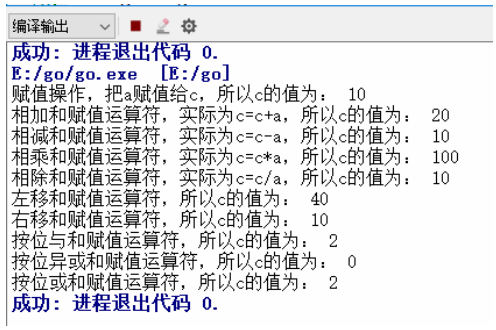


图 2-2　赋值运算符的运行结果

## 2.2.5　枚举

常量还可以用作枚举。

```
const (
    Connected = 0
    Disconnected =1
    Unknown =2
)
```

在以上代码中，数字 0 表示连接成功；数字 1 表示连接断开；数字 2 表示未知状态。

枚举类型的实现需要使用 iota 关键字。

```
package main
import (
```

```
    "fmt"
)
const (
    a = iota                          //a==0
    b                                 //b==1,隐式使用 iota 关键字,等同于 b=iota
    c                                 //c==2,等同于 c=iota
    d, e, f = iota, iota, iota        //d=3,e=3,f=3,同一行值相同
    g = iota                          //g==4
    h = "h"                           //h=="h"单独赋值,iota 递增为 5
    i                                 //i=="h",默认使用上面的赋值,iota 递增为 6
    j = iota                          //j==7
)
const z = iota                        //每个单独定义的 const 常量中,iota 都会重置,此时 z==0
func main() {
    fmt.Println(a, b, c, d, e, f, g, h, i, j, z)
}
```

运行结果如图 2-3 所示。

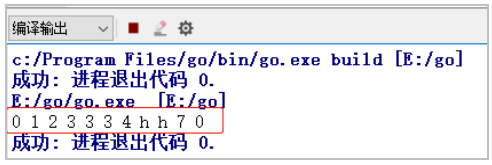


图 2-3 枚举运行结果

每个 const 定义的第一个常量被默认设置为 0，显式设置为其他值除外。后续的常量默认设置为它上面那个常量的值，如果前面那个常量的值是 iota，则它也被设置为 iota。由于 iota 可以实现递增，因此它也可以实现枚举操作。另外，iota 还可以在表达式中使用。

**注意：**当遇到 d, e, f = iota, iota, iota 这种情况时，虽然同一行值相同，但不能省略其他 iota，书写完整才是正确的。

# 2.3 变量

变量主要用来存储数据信息，变量的值可以通过变量名进行访问。Go 语言的变量名的命名规则与其他语言一样，都是由字母、数字和下画线组成，其中变量名的首字符不能为数字。常见的变量的数据类型包括整型、浮点型、布尔型及结构体等。

## 2.3.1 变量的声明

变量的声明通常使用 var 关键字，变量的声明格式如下：

```
var identifier type
```

其中，var 是声明变量的关键字，identifier 是变量名，type 是变量的类型，行尾无须添加分号，例如：

```
var a int              //声明整型类型的变量,保存整数数值
var b string           //声明字符串类型的变量
var c []float32        //声明 32 位浮点切片类型的变量,浮点切片表示由多个浮点类型组成的数据结构
var d func() bool      //声明返回值为布尔类型的函数变量
var e struct {         //声明结构体类型的变量,该结构体拥有整型的 x 字段
    x int
}
```

**注意：**Go 语言和其他编程语言不同之处在于，它在声明变量时将变量的类型放在变量的名称之后。这样做的好处就是可以避免出现含糊不清的声明形式。例如，如果想要两个变量都是指针，不需要将它们分开书写，写成 var a, b *int 即可。

同样，可以一次声明多个变量。在声明多个变量时可以写成常量的那种形式，例如：

```
var (
    a int
    b string
    c []float32
    d func() bool
    e struct {
        x int
    }
)
```

同一类型的多个变量可以声明在同一行，例如：

```
var a, b, c int
```

多个变量可以在同一行进行声明和赋值，例如：

```
var a, b, c int = 1, 2, 3
```

多个变量可以在同一行进行赋值，但注意只能在函数体内，例如：

```
a, b = 1, 2
```

如果想要交换两个变量的值，可以使用交换语句，例如：

```
a, b = b, a
```

## 2.3.2 初始化变量

Go 语言在声明变量时，自动对变量对应的内存区域进行初始化操作。每个变量会初始化其类型的默认值，例如：

①整型和浮点型变量的默认值为 0。

②字符串变量的默认值为空字符串。

③布尔型变量的默认值为 false。

④切片、函数、指针变量的默认值为 nil。

### 1. 变量初始化的标准格式

变量的初始化标准格式如下：

```
var 变量名 类型 = 表达式
```

例如：

```
var a int = 2
```

其中，a 为变量名，类型为 int，a 的初始值为 2。

### 2. 编译器推导类型的格式

2 和 int 同为 int 类型，因此可以进一步简化初始化的写法，即

```
var a = 2
```

等号右边的部分在编译原理里被称为右值（rvalue）。

```
var attack = 35
var defence = 15
var damageRate float32 = 0.28
var damage = float32(attack+defence) * damageRate
```

```
fmt.Println(damage)
```

第 1 行和第 2 行，右值为整型，attack 和 defence 变量的类型为 int。

第 3 行，表达式的右值中使用了 0.28。Go 语言和 C 语言一样，这里如果不指定 damageRate 变量的类型，Go 语言编译器会将 damageRate 类型推导为 float64，由于这里不需要 float64 的精度，所以需要强制指定类型为 float32。

第 4 行，将 attack 和 defence 相加后的数值结果依然为整型，使用 float32()将结果转换为 float32 类型，再与 float32 类型的 damageRate 相乘后，damage 类型也是 float32 类型。

第 5 行，输出 damage 的值。

运行结果如图 2-4 所示。

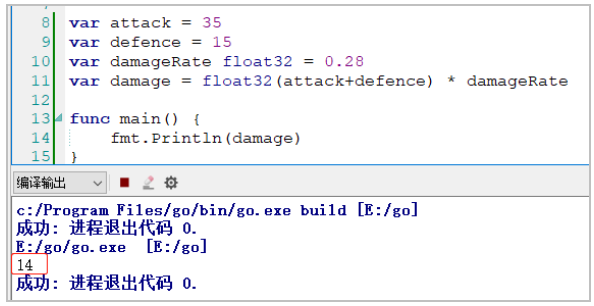


图 2-4 变量初始化运行结果

### 3. 短变量声明并初始化

var 的变量声明还有一种更为精简的写法，即

```
a := 2
```

其中，:=只能出现在函数内（包括在方法内），此时 Go 编译器会自动进行数据类型的推断。

**注意：由于使用了:=，而不是赋值的=，因此推导声明写法的左值变量必须是没有定义过的变量。若再次定义，将会出现编译错误。**

该写法同样支持多个类型变量同时声明并赋值，例如：

```
a, b := 1, 2
```

Go 语言中，除了可以在全局声明中初始化实体，也可以在 init 函数中初始化。init 函数是一个特殊的函数，它会在包完成初始化后自动执行，执行优先级比 main 函数高，并且不能手动调用 init 函数。每一个源文件有且只有一个 init 函数，初始化过程会根据包的依赖关系按顺序单线程执行。

可以在开始执行程序之前通过 init 函数来对数据进行检验与修复，保证程序执行时状态正常，例如：

```
package main
import (
    "fmt"
    "math"
)
//Pi 为圆周率
var Pi float64
func init() {
    Pi = 4 * math.Atan(1)  //在 init 函数中计算 Pi 的值
}
func main() {
    DPi := Pi * Pi
    fmt.Println(Pi, DPi)
}
```

运行结果如图 2-5 所示。

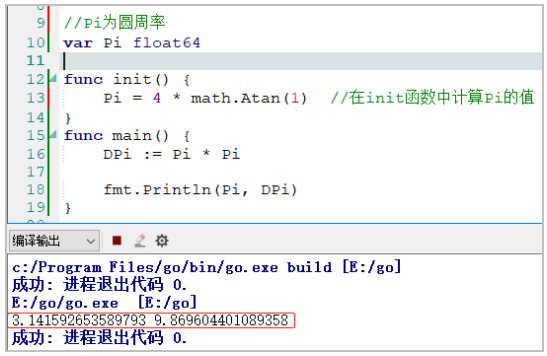


图 2-5　init 函数初始化运行结果

## 2.3.3　多个变量同时赋值

变量的交换是编程最简单的算法之一。在进行交换变量时，通常需要一个中间变量临时对变量进行保存。用传统方法编写变量交换代码如下：

```
var a int = 100
var b int = 200
var t int
t = a
a = b
b = t
fmt.Println(a, b)
```

传统方法的变量交换往往占用了较大的内存空间，因此，根据这一情况又发明了一些算法来避免使用中间变量，例如：

```
var a int = 100
var b int = 200
a = a ^ b
b = b ^ a
a = a ^ b
fmt.Println(a, b)
```

这种算法往往对数值范围和类型都有一定的要求。到了 Go 语言时，内存不再是紧缺资源，而且写法可以更加简单。使用 Go 语言的“多重赋值”特性，可以轻松完成变量交换的任务，例如：

```
var a int = 100
var b int = 200
b, a = a, b
fmt.Println(a, b)
```

在对多个变量同时赋值时，变量的左值和右值按从左到右的顺序依次赋值。多重赋值在 Go 语言的错误处理和函数返回值中会大量地使用。例如，使用 Go 语言进行排序时就需要使用变量的交换，代码如下：

```
type IntSlice []int
func (p IntSlice) Len() int          { return len(p) }
func (p IntSlice) Less(i, j int) bool { return p[i] < p[j] }
func (p IntSlice) Swap(i, j int)      { p[i], p[j] = p[j], p[i] }
```

在以上代码中：

第 1 行，将 IntSlice 声明为[]int 类型。

第 2 行，为 IntSlice 类型编写一个 Len 方法，提供切片的长度。

第 3 行，根据提供的 i、j 元素索引，获取元素后进行比较，返回比较结果。
第 4 行，根据提供的 i、j 元素索引，交换两个元素的值。

## 2.3.4　匿名变量

在编码过程中，可能会遇到没有名称的变量、类型或方法。虽然这不是必需的，但有时候这样做可以极大地增强代码的灵活性，这些变量被统称为匿名变量。

匿名变量可以用下画线（“_”）表示，而“_”本身就是一个特殊的标识符，因此被称为空白标识符。它可以像其他标识符那样用于变量的声明或赋值（任何类型都可以赋值给它），但任何赋给这个标识符的值都将被抛弃，因此这些值不能在后续的代码中使用，也不可以使用这个标识符作为变量对其他变量进行赋值或运算。使用匿名变量时，只需要在变量声明的地方使用下画线替换即可，例如：

```
func GetData() (int, int) {
   return 100, 50
}
func main(){
   a, _ := GetData()
   _, b := GetData()
   fmt.Println(a, b)
}
```

运行结果如图 2-6 所示。

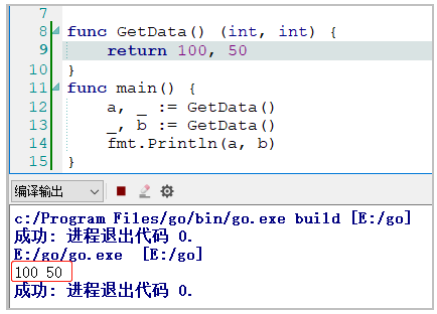


图 2-6　匿名变量赋值运算结果

GetData()是一个函数，拥有两个整型返回值。每次调用将会返回 100 和 50 两个数值。
在以上代码中：
第 5 行只需要获取第一个返回值，所以将第二个返回值的变量设为下画线（匿名变量）。
第 6 行将第一个返回值的变量设为匿名变量。

## 2.3.5　变量的作用域

一个变量（常量、类型或函数）在程序中都有一定的作用范围，该作用范围被称为作用域。
Go 语言中变量可以在以下 3 个地方进行声明：
（1）函数内定义的变量称为局部变量。
（2）函数外定义的变量称为全局变量。
（3）函数定义中的变量称为形式参数。

### 1. 局部变量

在函数体内声明的变量称为局部变量，它们的作用域只在函数体内，函数的参数和返回值变量都属于

局部变量。

局部变量不是一直存在的，它只在定义它的函数被调用后存在，函数调用结束后这个局部变量就会被销毁。

以下实例中 main()函数使用了局部变量 a、b、c：

```
package main
import (
    "fmt"
)
func main() {
    //声明局部变量 a 和 b 并赋值
    var a int = 5
    var b int = 3
    //声明局部变量 c 并计算 a 和 b 的和
    c := a + b
    fmt.Printf("a = %d, b = %d, c = %d\n", a, b, c)
}
```

运行结果如图 2-7 所示。

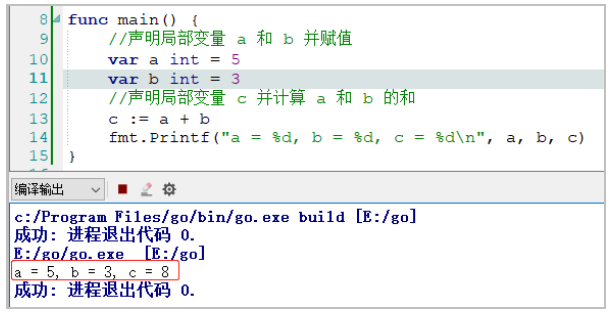


图 2-7　局部变量 a、b、c 运行结果

### 2. 全局变量

在函数体外声明的变量称为全局变量，全局变量只需要在一个源文件中定义，就可以在所有源文件中使用。当然，不包含这个全局变量的源文件需要使用 import 关键字引入全局变量所在的源文件之后才能使用这个全局变量。

全局变量声明必须以 var 关键字开头，如果想要在外部包中使用，全局变量的首字母必须大写。

以下实例中定义了全局变量 c：

```
package main
import "fmt"
//声明全局变量
var c int
func main() {
    //声明局部变量
    var a, b int
    //初始化参数
    a = 5
    b = 3
    c = a + b
    fmt.Printf("a = %d, b = %d, c = %d\n", a, b, c)
}
```

运行结果如图 2-8 所示。

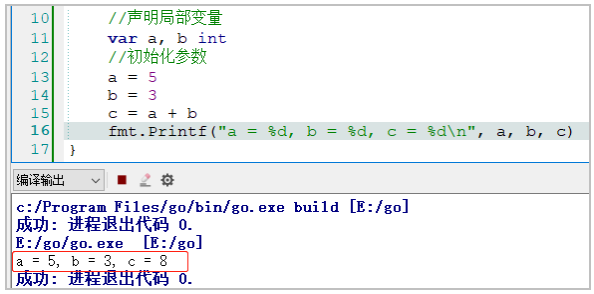


图 2-8 全局变量 c 运行结果

Go 语言程序中全局变量与局部变量名称可以相同，但是函数体内的局部变量会被优先考虑。

```
package main
import "fmt"
//声明全局变量
var a float32 = 3.14
func main() {
    //声明局部变量
    var a int = 5
    fmt.Printf("a = %d\n", a)
}
```

运行结果如图 2-9 所示。

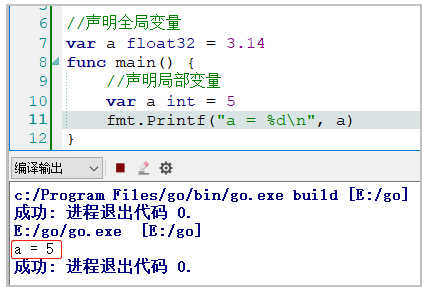


图 2-9 全局变量和局部变量同时使用

### 3. 形式参数

在定义函数时，函数名后面括号中的变量称为形式参数（简称形参）。形式参数只在函数调用时才会生效，函数调用结束后就会被销毁。在函数未被调用时，函数的形参并不占用实际的存储单元，也没有实际值。

形式参数会作为函数的局部变量来使用。

以下实例中定义了形式参数 a、b、c：

```
package main
import "fmt"
/*声明全局变量*/
var a int = 20;
func main() {

    /*main 函数中声明局部变量*/
    var a int = 10
    var b int = 20
    var c int = 0
    fmt.Printf("main()函数中 a = %d\n", a);
    c = sum( a, b);
    fmt.Printf("main()函数中 c = %d\n", c);

 }
 /*函数定义-两数相加*/
```

```
func sum(a, b int) int {
    fmt.Printf("sum() 函数中 a = %d\n", a);
    fmt.Printf("sum() 函数中 b = %d\n", b);
    return a + b;
}
```

运行结果如图 2-10 所示。

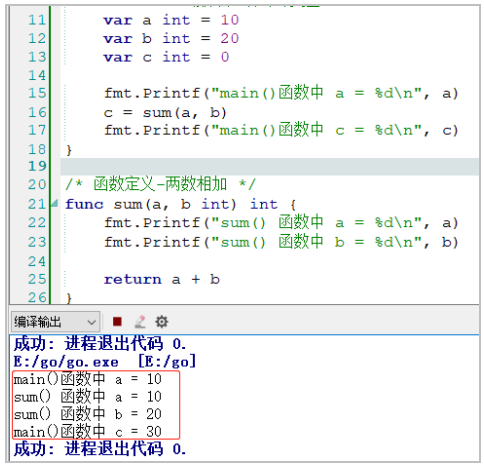


图 2-10　形式参数 a、b、c 运行结果

# 2.4　就业面试技巧与解析

本章主要讲解了 Go 语言程序元素的构成，通过本章内容的学习，读者不仅能够掌握标识符、关键字、字面量、分隔符、运算符及注释等词法单元的基础知识，而且还可以学习常量，以及变量的声明和赋值等内容。

## 2.4.1　面试技巧与解析（一）

**面试官：**Go 语言中如何表示枚举值（enums）？

**应聘者：**Go 语言中是没有枚举类型（enums）的，但可以使用其他函数代替，可以在一个独立的 const 区域中使用 iota 来生成递增的值。如果在 const 中，常量没有初始值，则会使用前面的初始化表达式代替。

在 Go 语言中没特别地为枚举指定创建方法，可以通过定义 func，然后在其中创建静态变量来定义枚举，例如：

```
func enums(){
    const(
        left = 0
        top = 1
        right = 2
        bottom = 3
    )
    fmt.Println(left,top,right,bottom)
}
```

在 Go 语言中还可以使用 iota 来创建枚举，iota 为自增值，例如：

```
const(
        left = iota
```

```
        top
        right
        bottom
    )
```

iota 代表了一个连续的整型常量。

iota 和 const 搭配使用时，将会被重置为 0。

iota 所定义的值类型为 int，它会在每次赋值给一个常量后自增。

## 2.4.2 面试技巧与解析（二）

**面试官：**=和:=有什么区别？

**应聘者：**

使用=前必须先用 var 声明，例如：

```
var a
a=100
//或
var b = 100
//或
var c int = 100
```

:=是声明并赋值，并且系统自动推断类型，不需要 var 关键字，例如：

```
d := 100
```