Программирование на языке Go

Марк Саммерфильд

ДМК издательство

УДК 004.438Go
ББК 32.973.26-018.1
С17

Марк Саммерфильд

С17 Программирование на Go. Разработка приложений XXI века: пер. с англ.: Киселёв А. Н. – М.: ДМК Пресс, 2016. – 580 с.: ил.

ISBN 978-5-97060-338-3

На сегодняшний день Go – самый впечатляющий из новых языков программирования. Изначально он создавался для того, чтобы помочь задействовать всю мощь современных многоядерных процессоров. В этом руководстве Марк Саммерфильд, один из основоположников программирования на языке Go, показывает, как писать программы, в полной мере использующие его революционные возможности и идиомы.

Данная книга представляет собой одновременно и учебник, и справочник, сводя воедино все знания, необходимые для того, чтобы продолжать освоение Go, думать на Go и писать на нем высокопроизводительные программы. Автор приводит множество сравнений идиом программирования, демонстрируя преимущества Go перед более старыми языками и уделяя особое внимание ключевым инновациям. Попутно, начиная с самых основ, Марк Саммерфильд разъясняет все аспекты параллельного программирования на языке Go с применением каналов и без использования блокировок, а также показывает гибкость и необычность подхода к объектно-ориентированному программированию с применением механизма динамической типизации.

Издание предназначено для программистов разной квалификации, желающих освоить и применять в своей практике язык Go.

ISBN 978-0-321-77463-7 (англ.)
ISBN 978-5-97060-338-3 (рус.)

Содержание

Эта книга посвящается

Жасмин Бланшетт (Jasmin Blanchette) и Трентону Шульцу (Trenton Schulz)

1. Обзор в пяти примерах

В этой главе приводится серия из пяти примеров с подробными их описаниями. Несмотря на небольшой размер примеров, каждый из них (кроме «Hello Who?») имеет некоторую практическую ценность, а все вместе они представляют краткий обзор ключевых возможностей языка Go и некоторых его основных пакетов. (То, что в других языках называется модулями или библиотеками, в языке Go называется *пакетами*, а все пакеты, распространяемые вместе с Go, образуют *стандартную библиотеку* языка Go.) Цель этой главы – сформировать представление о языке и вселить в читателя уверенность в необходимости освоения языка Go. Не стоит волноваться, если какие-то синтаксические конструкции или идиомы останутся за рамками понимания, – все, что будет показано в этой главе, обязательно будет рассматриваться в последующих главах.

Обучение программированию на языке Go с применением специфических приемов требует определенного времени и усилий. Желающие заняться переносом программ с языков C, C++, Java, Python и др. на язык Go должны найти время на изучение Go, особенно его объектно-ориентированных возможностей и средств организации параллельных вычислений, в долгосрочной перспективе это позволит сэкономить время и силы. А желающие писать на языке Go собственные приложения добьются большего успеха, если максимально будут использовать все его возможности, поэтому они также должны потратить силы и время на изучение – позднее все затраты окупятся с лихвой.

1.1. Начало

Go – это компилирующий язык, а не интерпретирующий, поэтому программы, написанные на этом языке, имеют максимальную производительность. Компиляция выполняется очень быстро – намного быстрее, чем в некоторых других языках, особенно в сравнении с языками C и C++.

Документация по языку Go

По адресу golang.org находится официальный веб-сайт языка Go, где можно найти массу свежей документации по языку Go. По ссылке **Packages** (Пакеты) можно перейти к разделу с документацией ко всем пакетам из стандартной библиотеки Go и их исходными текстами, которые могут очень пригодиться, когда в документации обнаруживаются пробелы. Ссылка **References** => **Command Documentation** (Справочники => Документация к командам) ведет в раздел с документацией к программам, распространяемым вместе с Go (компиляторам, инструментам сборки и др.). По ссылке **References** => **Language Specification** (Справочники => Спецификация языка) можно перейти к разделу с неофициальной и весьма полной спецификацией языка Go. А по ссылке **Documents** => **Effective Go** (Документы => Эффективный Go) находится документ, описывающий многие приемы программирования на Go.

На веб-сайте также имеется «песочница» (с ограниченным набором возможностей), где можно вводить, компилировать и опробовать небольшие программы на языке Go. Данная возможность будет полезна начинающим для проверки непонятных синтаксических конструкций и изучения сложного пакета `fmt`, содержащего инструменты для работы с форматированным текстом, или пакета `regexp` – механизма регулярных выражений. Строка поиска на веб-сайте языка Go может использоваться только для поиска документации – если потребуется отыскать другие ресурсы, посвященные языку Go, посетите страницу go-lang.cat-v.org/go-search.

Документацию по языку Go можно также просматривать локально, например в веб-браузере. Для этого выполните команду `godoc`, передав ей аргумент, сообщающий, что она должна действовать как веб-сервер. Ниже показано, как выполнить эту команду в консоли Unix (`xterm`, `gnome-terminal`, `konsole`, `Terminal.app` и др.):

```
$ godoc -http=:8000
```

Или в консоли Windows (например, в окне **Command Prompt** (Командная строка) или **MS-DOS Prompt** (Сеанс MS-DOS)):

```
C:\>godoc -http=:8000
```

Номер порта здесь выбран произвольно. Если он уже занят – просто выберите другой. Здесь предполагается, что выполняемый файл `godoc` находится в каталоге, указанном в переменной `PATH`.

Для просмотра локальной документации откройте веб-браузер и введите адрес http://localhost:8000. На экране появится страница, внешним видом напоминающая главную страницу веб-сайта golang.org. Ссылка **Packages** (Пакеты) ведет в раздел документации к стандартной библиотеке Go, где также имеется документация к сторонним

пакетам, установленным в каталог GOROOT. Если в системе определена переменная окружения GOPATH (например, для локальных программ и пакетов), рядом со ссылкой **Packages** (Пакеты) появится ссылка к разделу с соответствующей документацией. (Переменные GOROOT и GOPATH обсуждаются ниже в этой главе, а также в главе 9.) С помощью команды godoc можно еще просматривать документацию для всего пакета в целом или для отдельного его элемента непосредственно в консоли. Например, команда godoc image NewRGBA выведет описание функции image.NewRGBA(), а команда godoc image/png – описание пакета image/png в целом.

Стандартный компилятор языка Go называется gc, а в состав его инструментов входят программы: 5g, 6g и 8g – для компиляции, 5l, 6l и 8l – для компоновки и godoc – для просмотра документации. (В Windows эти программы называются 5g.exe, 6l.exe и т. д.) Такие странные имена были даны в соответствии с соглашениями об именовании компиляторов, принятыми в операционной системе Plan 9, где цифра определяет аппаратную архитектуру (например, «5» – ARM, «6» – AMD-64, включая 64-битные процессоры Intel, и «8» – Intel 386.) К счастью, нет необходимости напрямую использовать эти инструменты благодаря наличию высокоуровневого инструмента сборки программ на языке Go – go, который автоматически выбирает нужный компилятор и компоновщик.

Все примеры из этой книги, доступные для загрузки на странице www.qtrac.eu/gobook.html, были проверены в Linux и Mac OS X с помощью gc, и в Windows с помощью компилятора версии Go 1. Разработчики Go предполагают обеспечить обратную совместимость с версией Go 1 во всех последующих версиях Go 1.x, поэтому описание в книге и примеры должны быть верными для всей серии 1.x. (Со временем, при обнаружении каких-либо несовместимостей, загружаемые примеры для книги будут обновляться в соответствии с последней версией Go, поэтому они могут отличаться от программного кода в книге.)

Чтобы загрузить и установить Go, откройте страницу golang.org/doc/install.html, где приводятся ссылки для загрузки и инструкции по установке. На момент написания этих строк версия Go 1 была доступна в виде двоичных и исходных файлов для FreeBSD 7+, Linux 2.6+, Mac OS X (Snow Leopard и Lion) и Windows 2000+ и во всех случаях для аппаратных архитектур Intel 386 and AMD-64. Для Linux имеется также поддержка архитектуры ARM. Предварительно собранные пакеты Go имеются для дистрибутива Ubuntu Linux, и

к моменту, когда вы будете читать эти строки, они могут появиться для других дистрибутивов Linux. Для начинающих изучать программирование на языке Go проще установить двоичную версию, чем собирать инструменты Go из исходных текстов.

Для программ, собираемых компилятором gc, действуют определенные соглашения об именовании. То есть программы, скомпилированные с помощью gc, могут быть скомпонованы только с внешними библиотеками, следующими тем же соглашениям, в противном случае необходимо использовать подходящий инструмент, устраняющий разногласия. В комплект Go входит инструмент cgo (golang.org/cmd/cgo), обеспечивающий возможность использования внешнего программного кода на языке C в программах на языке Go, кроме того, в Linux и BSD-системах имеется возможность использовать код на C и C++ с помощью инструмента SWIG (www.swig.org).

Помимо gc, имеется также компилятор gccgo. Это интерфейс к компилятору gcc (GNU Compiler Collection) для языка Go, который может быть задействован с компиляторами gcc, начиная с версии 4.6. Подобно gc, компилятор gccgo может быть доступен в некоторых дистрибутивах Linux в виде готовых пакетов. Инструкции по сборке и установке компилятора gccgo можно найти на странице golang.org/doc/gccgo_install.html.

1.2. Правка, компиляция и запуск

Программы на языке Go записываются в виде простого текста Юникода с использованием кодировки UTF-8[1]. Большинство современных текстовых редакторов обеспечивают эту поддержку автоматически, а некоторые наиболее популярные из них поддерживают даже подсветку синтаксиса для языка Go и автоматическое оформление отступов. Если ваш текстовый редактор не поддерживает Go, попробуйте ввести имя редактора в строке поиска на сайте Go, чтобы узнать, имеются ли для него расширения, обеспечивающие требуемую поддержку. Для удобства правки все ключевые слова и операторы языка Go записываются символами ASCII, однако идентификаторы в языке Go могут начинаться с любых алфавитных символов

[1] Некоторые текстовые редакторы для Windows (такие как Notepad (Блокнот)) не следуют рекомендациям стандарта Юникода и вставляют байты 0xEF, 0xBB, 0xBF в начало файлов с текстом в кодировке UTF-8. В этой книге предполагается, что файлы в кодировке UTF-8 не содержат этих байтов.

Юникода и содержать любые алфавитные символы Юникода или цифры. Благодаря этому программисты на Go свободно могут определять идентификаторы на своем родном языке.

Сценарии на языке Go

Одним из побочных эффектов высокой скорости компиляции программ на языке Go является возможность создания сценариев в Unix-подобных системах, начинающихся со строки `#!`. Для этого достаточно лишь установить подходящий инструмент, выполняющий компиляцию и запуск программы. На момент написания этих строк имелись два таких инструмента: `gonow` (github.com/kless/gonow) и `gorun` (wiki.ubuntu.com/gorun).

После установки `gonow` или `gorun` любую программу на языке Go можно оформить в виде сценария. Достигается это выполнением двух простых действий. Первое – добавить строку `#!/usr/bin/env gonow` или `#!/usr/bin/env gorun` в самое начало файла с расширением `.go`, содержащим функцию `main()` (в пакете `main`). Второе – дать файлу права на выполнение (например, командой `chmod +x`). Такие файлы могут компилироваться только инструментами `gonow` и `gorun`, потому что строка `#!` не является синтаксически допустимой строкой на языке Go.

При первом запуске команда `gonow` или `gorun` скомпилирует файл с расширением `.go` (очень быстро, разумеется) и запустит его. При последующих попытках перекомпиляция будет выполняться, только если исходный файл `.go` изменился с момента предыдущей компиляции. Это делает возможным написание на языке Go различных небольших вспомогательных программ, например для решения задач системного администрирования.

Чтобы получить представление, как писать, компилировать и выполнять программы на языке Go, начнем с классического примера «Hello World». Несмотря на небольшой размер, программа будет выглядеть чуть сложнее, чем обычно. Но для начала обсудим компиляцию и запуск программы, а затем, в следующем разделе, перейдем к детальному изучению исходного программного кода в файле `hello/hello.go`, использующего некоторые базовые идеи и особенности языка Go.

Все примеры для этой книги доступны на странице www.qtrac.eu/gobook.html в виде архива каталога `goeg`. Поэтому полный путь к файлу `hello.go` (предполагается, что архив с примерами распакован непосредственно в домашний каталог, хотя его можно распаковать в любой другой каталог) будет иметь вид `$HOME/goeg/src/hello/hello.go`. При ссылке на имена файлов в этой книге всегда

будет предполагаться наличие первых трех компонентов пути, то есть в данном случае путь к файлу выглядит как `hello/hello.go`. (Разумеется, пользователи Windows должны читать символ «/» как «\» и использовать имя каталога, куда были распакованы примеры, например: `C:\goeg` или `%HOMEPATH%\goeg`.)

Если Go был установлен из двоичного дистрибутива или собран из исходных текстов и установлен с привилегиями пользователя root или Administrator, необходимо создать хотя бы одну переменную окружения, `GOROOT`, содержащую путь к каталогу установки Go, а в переменную `PATH` включить путь `$GOROOT/bin` или `%GOROOT%\bin`. Чтобы убедиться, что установка Go была выполнена правильно, можно выполнить следующую команду консоли Unix (`xterm`, `gnome-terminal`, `konsole`, `Terminal.app` и др.):

```
$ go version
```

Или в консоли Windows (например, в окне **Command Prompt** (Командная строка) или **MS-DOS Prompt** (Сеанс MS-DOS)):

```
C:\>go version
```

Если в консоли появится сообщение «command not found» (команда не найдена) или «'go' is not recognized...» (команда go не опознана), это означает, что путь к каталогу установки Go не был включен в переменную `PATH`. Простейший способ решить эту проблему в Unix-подобных системах (включая Mac OS X) – установить значения переменных окружения в файле `.bashrc` (или в эквивалентном ему, если используется другая командная оболочка). Например, файл `.bashrc` у автора содержит следующие строки:

```
export GOROOT=$HOME/opt/go
export PATH=$PATH:$GOROOT/bin
```

Естественно, конкретные значения следует установить в соответствии со своей системой. (И, разумеется, делать это необходимо только в случае неудачной попытки выполнить команду `go version`.)

Для Windows одно из решений заключается в том, чтобы создать пакетный файл, настраивающий окружение Go, и выполнять его при каждом запуске консоли для программирования на языке Go. Однако намного удобнее один раз настроить переменные окружения в панели управления. Для этого щелкните на кнопке **Start**

(Пуск) (с логотипом Windows), выберите пункт меню **Control Panel** (Панель управления), затем пункт **System and Security** (Система и безопасность), потом **System** (Система), далее **Advanced system settings** (Дополнительные параметры системы) и в диалоге **System Properties** (Свойства системы) щелкните на кнопке **Environment Variables** (Переменные окружения), затем на кнопке **New...** (Создать) и добавьте переменную с именем `GOROOT` и соответствующим значением, таким как `C:\Go`. В том же диалоге отредактируйте значение переменной окружения `PATH`, добавив в конец текст `;C:\Go\bin` – начальная точка с запятой имеют важное значение! В обоих случаях замените компонент пути `C:\Go` на фактический путь к каталогу установки Go, если он отличается от `C:\Go`. (Опять же, делать это необходимо только в случае неудачной попытки выполнить команду `go version`.)

С этого момента будет предполагаться, что язык Go установлен и путь к его каталогу `bin`, содержащему все инструменты Go, включен в переменную окружения `PATH`. (Чтобы новые настройки вступили в силу, может потребоваться открыть новое окно консоли.)

Сборка программ на языке Go выполняется в два этапа: компиляция и компоновка[1]. Оба этапа выполняются инструментом `go`, который не только собирает локальные программы и пакеты, но также способен загружать, собирать и устанавливать сторонние программы и пакеты.

Чтобы обеспечить сборку локальных программ и пакетов с помощью инструмента `go`, необходимо выполнить три обязательных условия. Первое: каталог `bin` с инструментами языка Go (`$GOROOT/bin` или `%GOROOT%\bin`) должен находиться в пути поиска `PATH`. Второе: в дереве каталогов должен существовать каталог `src` для хранения исходных текстов локальных программ и пакетов. Например, примеры для книги распаковываются в каталоги `goeg/src/hello`, `goeg/src/bigdigits` и т. д. Третье: путь к каталогу, *вмещающему* каталог `src`, должен быть включен в переменную окружения `GOPATH`. Так, чтобы собрать пример `hello` с помощью инструмента `go`, необходимо выполнить следующие операции:

[1] Поскольку эта книга предполагает, что компиляция выполняется с помощью компилятора `gc`, читатели, использующие `gccgo`, должны выполнять компиляцию и компоновку в соответствии с инструкциями на странице golang.org/doc/gccgo_install.html. Аналогично читатели, использующие другие компиляторы, должны выполнять компиляцию и компоновку в соответствии с инструкциями для их компилятора.

```
$ export GOPATH=$HOME/goeg
$ cd $GOPATH/src/hello
$ go build
```

В Windows эти операции выполняются практически так же:

```
C:\>set GOPATH=C:\goeg
C:\>cd %gopath%\src\hello
C:\goeg\src\hello>go build
```

В обоих случаях предполагается, что переменная PATH включает путь $GOROOT/bin или %GOROOT%\bin. После сборки программы инструментом go ее можно запустить. По умолчанию выполняемый файл получает имя каталога, в который он помещается (например, hello – в Unix-подобных системах и hello.exe – в Windows). Запуск программы выполняется как обычно.

```
$ ./hello
Hello World!
```

Или:

```
$ ./hello Go Programmers!
Hello Go Programmers!
```

В Windows запуск выполняется похожим способом:

```
C:\goeg\src\hello>hello Windows Go Programmers!
Hello Windows Go Programmers!
```

В примерах выше жирным шрифтом выделен текст, который должен вводиться вручную. Здесь также предполагается, что строка приглашения к вводу имеет вид $, но на самом деле это не имеет никакого значения (она может иметь вид, например, C:\>).

Обратите внимание на *отсутствие* необходимости компилировать или явно компоновать какие-либо другие пакеты (хотя в исходном файле hello.go, как будет показано ниже, используются три пакета из стандартной библиотеки). Это еще одна причина, объясняющая высокую скорость компиляции программ на языке Go.

Если бы потребовалось скомпилировать несколько программ, было бы удобнее, если бы все выполняемые файлы помещались в один

каталог, путь к которому включен в переменную PATH. К счастью, инструмент go поддерживает такую возможность:

```
$ export GOPATH=$HOME/goeg
$ cd $GOPATH/src/hello
$ go install
```

Или то же самое в Windows:

```
C:\>set GOPATH=C:\goeg
C:\>cd %gopath%\src\hello
C:\goeg\src\hello>go install
```

Команда go install делает то же самое, что и команда go build, но помещает выполняемые файлы в стандартный каталог ($GOPATH/bin или %GOPATH%\bin). То есть, если добавить путь ($GOPATH/bin или %GOPATH%\bin) в переменную PATH, *все* программы на языке Go будут устанавливаться в каталог, путь к которому включен в переменную PATH.

Помимо примеров из книги, многие пожелают писать на языке Go свои программы и пакеты и хранить их в отдельных каталогах. Обеспечить такую возможность можно простым включением в переменную окружения GOPATH двух (или более) путей к каталогам, разделенных двоеточием (точкой с запятой, в Windows). Например: export GOPATH=$HOME/app/go:$HOME/goeg или SET GOPATH=C:\app\go;C:\goeg[1]. В данном случае необходимо будет помещать все исходные тексты программ и пакетов в каталог $HOME/app/go/src или C:\app\go\src. То есть при создании программы с именем myapp ее исходный файл с расширением .go должен находиться в каталоге $HOME/app/go/src/myapp или C:\app\go\src\myapp. И если для сборки программы будет использоваться команда go install и при этом программа будет находиться в каталоге, путь к которому включен в переменную GOPATH, содержащую два или более каталога, выполняемый файл будет сохранен в соответствующем каталоге bin.

Естественно, слишком утомительно настраивать или экспортировать переменную GOPATH каждый раз, когда потребуется собрать программу, поэтому лучше определить эту переменную окружения на

[1] С этого момента практически всегда будут демонстрироваться команды только в стиле ОС Unix и предполагаться, что программисты, использующие ОС Windows, смогут мысленно преобразовать их в команды Windows.

постоянной основе. Сделать это можно, включив определение переменной GOPATH в файл .bashrc (или подобный ему) в Unix-подобных системах (см. файл gopath.sh в примерах к книге). В Windows то же самое можно сделать, либо написав пакетный файл (см. файл gopath.bat в примерах к книге), либо добавив определение переменной в системные переменные окружения: щелкните на кнопке **Start** (Пуск) (с логотипом Windows), выберите пункт меню **Control Panel** (Панель управления), затем выберите пункт **System and Security** (Система и безопасность), далее **System** (Система), потом **Advanced system settings** (Дополнительные параметры системы) и в диалоге **System Properties** (Свойства системы) щелкните на кнопке **Environment Variables** (Переменные окружения), затем на кнопке **New...** (Создать) и добавьте переменную с именем GOROOT и соответствующим значением, таким как C:\goeg или C:\app\go;C:\goeg.

Утилита go является стандартным инструментом сборки программ на языке Go. Тем не менее для тех же целей с успехом можно использовать утилиту make, другие подобные средства или альтернативные инструменты, предназначенные для сборки программ на языке Go, а также расширения для популярных интегрированных сред разработки (Integrated Development Environments, IDE), таких как Eclipse и Visual Studio.

1.3. Hello кто?

Теперь, когда стало понятно, как собрать программу hello, обратимся к исходным текстам. Не волнуйтесь, если что-то останется за рамками понимания, – все, что будет показано в этой главе (и многое другое!), будет подробно описываться в последующих главах. Ниже приводится полный листинг программы hello (в файле hello/hello.go):

```
// hello.go
package main
import ( ❶
    "fmt"
    "os"
    "strings"
)
func main() {
    who := "World!" ❷
    if len(os.Args) > 1 { /* os.Args[0] - имя команды «hello» или «hello.exe» */ ❸
```

```
        who = strings.Join(os.Args[1:], " ") ❹
    }
    fmt.Println("Hello", who) ❺
}
```

Комментарии в языке Go оформляются в стиле языка C++: однострочные комментарии, заканчивающиеся в конце строки, начинаются с символов //, а блочные комментарии, занимающие несколько строк, заключаются в символы /* ... */. Обычно в программах на языке Go используются однострочные комментарии, включая комментарии, используемые для исключения из программы фрагментов программного кода во время отладки[1].

Любой фрагмент программного кода на языке Go должен быть включен в пакет, а каждая программа должна иметь пакет `main` с функцией `main()`, которая является точкой входа в программу, то есть с функцией, выполняющейся в первую очередь. В действительности пакеты на языке Go могут также иметь функцию `init()`, которая выполняется перед функцией `main()`, как будет показано ниже (§1.7). Подробнее об этом будет рассказываться далее (§5.6.2). Обратите внимание, что здесь между именем пакета и именем функции нет никакого конфликта.

Язык Go оперирует в терминах *пакетов*, а не файлов. То есть пакет можно разбить на любое количество файлов, и если все они будут иметь одинаковое объявление пакета, с точки зрения языка Go все они будут являться частями одного и того же пакета, как если бы все их содержимое находилось в единственном файле. Естественно, точно так же всю функциональность приложения можно распределить по нескольким пакетам, чтобы обеспечить модульный принцип построения, как будет показано в главе 9.

Инструкция `import` (❶ в листинге выше) импортирует три пакета из стандартной библиотеки. Пакет `fmt` содержит функции форматирования текста и чтения форматированного текста (§3.5), пакет `os` содержит платформонезависимые системные переменные и функции, а пакет `strings` – функции для работы со строками (§3.6.1).

Фундаментальные типы данных в языке Go поддерживают привычные операторы (например, + – для сложения чисел и

[1] В листингах примеров будет использоваться прием подсветки синтаксиса и к отдельным строкам будут добавляться числа (❶, ❷, ...), чтобы проще было ссылаться на них в тексте. Ни один из этих элементов не является частью языка Go.

конкатенации строк), а стандартная библиотека Go добавляет дополнительные пакеты функций для работы с фундаментальными типами, такие как импортированный здесь пакет `strings`. Кроме того, имеется возможность определять пользовательские типы данных, опираясь на фундаментальные типы, и предусматривать собственные методы, то есть функции, для работы с ними. (Коротко об этом будет рассказываться в §1.5 ниже, а подробно эта тема будет обсуждаться в главе 6.)

Внимательный читатель, возможно, заметил, что в программе отсутствуют точки с запятой, импортируемые пакеты не отделяются друг от друга запятыми и условное выражение в инструкции `if` не требуется заключать в круглые скобки. В языке Go блоки программного кода, включая тела функций и управляющих конструкций (например, инструкций `if` и циклов `for`), заключаются в фигурные скобки. Отступы используются исключительно для удобства человека. Технически инструкции в языке Go должны отделяться друг от друга точками с запятой, но они автоматически добавляются компилятором, поэтому в них нет необходимости, если в одной строке не располагаются несколько инструкций. Отсутствие необходимости вставлять точки с запятой, небольшое количество ситуаций, когда требуется вставлять запятые и фигурные скобки, делают программы на языке Go более удобочитаемыми и уменьшают объем ввода с клавиатуры.

Функции и методы в языке Go определяются с помощью ключевого слова `func`. Функция `main()` в пакете `main` всегда имеет одну и ту же сигнатуру – она не имеет аргументов и ничего не возвращает. Когда функция `main.main()` завершается, одновременно с ней завершается выполнение программы, и она возвращает операционной системе значение `0`. Естественно, имеется возможность в любой момент завершить работу программы и вернуть любое значение, как будет показано далее (§1.4).

Первая инструкция в функции `main()` (❷ в листинге выше, где используется оператор :=) в терминологии языка Go называется *сокращенным объявлением переменной*. Такие инструкции одновременно объявляют и инициализируют переменные. Кроме того, в подобных инструкциях нет необходимости объявлять тип переменной, потому что компилятор Go автоматически определит его по присваиваевому значению. Таким образом, в данном случае объявляется переменная с именем `who` типа `string`, и, как следствие строгой типизации в языке Go, далее переменной `who` могут присваиваться только строковые значения.