ГЛАВА 1

Настройка среды разработки для языка Go

Любой язык программирования нуждается в среде разработки, и Go не исключение. Если вы уже успели создать на Go одну-две программы, то у вас уже, вероятно, имеется рабочая среда, но при этом вы могли упустить из виду некоторые новые методы и инструменты. Если же вы в первый раз настраиваете свой компьютер для разработки на Go, не беспокойтесь: установка этого языка и его вспомогательных инструментов не составляет большого труда. После того как вы настроите среду и убедитесь, что все сделано правильно, мы напишем простую программу и опробуем несколько способов компиляции и запуска Go-кода, после чего рассмотрим несколько инструментов и методов, упрощающих процесс разработки на Go.

Установка средств разработки для языка Go

Чтобы приступить к написанию кода на Go, нужно установить средства разработки. Последнюю версию этих инструментов можно скачать с официального сайта языка Go (https://golang.org/dl). Скачайте и установите версию, подходящую для вашей платформы. Установочные пакеты с расширением `.pkg` для Mac и расширением `.msi` для Windows автоматически установят Go в нужном каталоге, удалят ранее установленные файлы и разместят исполняемый файл языка Go в соответствии с путем по умолчанию для исполняемых файлов.

Для платформы Mac средства разработки для языка Go можно установить с помощью менеджера пакетов Homebrew (https://brew.sh), выполнив команду `brew install go`. Для Windows это можно сделать с помощью менеджера пакетов Chocolatey (https://chocolatey.org), выполнив команду `choco install golang`.

Версии установочных пакетов для Linux и BSD представляют собой сжатые TAR-файлы, которые распаковываются в каталог с именем `go`. Скопируйте этот каталог в `/usr/local` и добавьте путь `/usr/local/go/bin` в переменную среды `$PATH`, чтобы сделать доступной команду `go`:

```
$ tar -C /usr/local -xzf go1.20.5.linux-amd64.tar.gz
$ echo 'export PATH=$PATH:/usr/local/go/bin' >> $HOME/.bash_profile
$ source $HOME/.bash_profile
```

Для записи в `/usr/local` могут потребоваться полномочия `root`. Если команда `tar` не выполняется, выполните ее повторно с помощью `sudo tar -C /usr/local -xzf go1.20.5.linux-amd64.tar.gz`.

Go-программы компилируются в один нативный двоичный файл и не требуют установки дополнительного программного обеспечения для запуска. Устанавливать средства разработки для языка Go нужно только на тех компьютерах, на которых вы будете компилировать Go-программы. В этом состоит отличие от таких языков, как Java, Python и JavaScript, которые требуют установки виртуальной машины для запуска программы. Использование одного собственного двоичного файла значительно упрощает распространение программ, написанных на языке Go. В этой книге не рассматриваются контейнеры, но разработчики, применяющие Docker или Kubernetes, часто могут упаковать приложение Go в образ scratch или distroless. Подробнее об том можете почитать в блоге Герта Бека (Geert Baeke), в статье *Distroless or scratch for Go apps?* (https://oreil.ly/W0VUB).

Чтобы убедиться, что ваша среда правильно настроена, откройте окно терминала или командной строки и введите команду:

```
$ go version
```

Если все в порядке, то вы увидите что-то наподобие следующего:

```
go version go1.20.5 darwin/amd64
```

Это сообщение говорит о наличии компилятора Go версии 1.20.5 для macOS. (Darwin — это операционная система, лежащая в основе macOS, а arm64 — название 64-разрядных чипов, основанных на разработках ARM.) В x64 Linux вы увидите:

```
go version go1.20.5 linux/amd64
```

Устранение неполадок при установке Go

Если вместо сообщения с описанием версии вы увидите сообщение об ошибке, значит, у вас нет файла `go` в каталоге, заданном в качестве пути для исполняемых файлов, или в этом файле находится другая программа с таким же именем.

В macOS или других Unix-подобных системах можно выяснить, какой файл `go` запускается и запускается ли вообще, с помощью команды `which go`. Если ничего не возвращается, скорректируйте у себя путь для исполняемых файлов.

Если работаете в Linux или BSD, ошибка может состоять в том, что вы установили 64-разрядную версию средств разработки для языка Go в 32-разрядной системе или версию для другой процессорной архитектуры.

Инструментарий Go

Доступ ко всем инструментам разработки на Go осуществляется с помощью команды `go`. Помимо `go version`, есть компилятор `go build`, средство форматирования кода `go fmt`, менеджер зависимостей `go mod`, средство запуска тестов `go test`, инструмент для поиска распространенных ошибок при кодировании `go vet` и многое другое. Подробно о них рассказывается в главах 10, 11 и 15. А пока давайте вкратце рассмотрим наиболее часто используемые инструменты на примере создания всеми любимого первого приложения «Hello, World!».

С момента появления Go в 2009 году его разработчики ввели несколько изменений в способ организации кода и его зависимостей. Из-за этого вы можете найти множество противоречивых рекомендаций, большая часть которых уже устарела (например, смело игнорируйте обсуждения `GOROOT` и `GOPATH`).

В контексте современных разработок на Go действует простое правило: можете организовывать свои проекты любым удобным вам способом и хранить их в любом удобном месте.

Ваша первая программа на Go

Рассмотрим основы написания программы на Go. По ходу дела вы увидите, из каких частей состоит простая программа на Go. Возможно, вы еще не все поняли, но это нормально — для этого и предназначена остальная часть книги!

Создание модуля Go

Первое, что вам нужно сделать, — создать каталог для хранения своей программы. Назовем его `ch1`. В командной строке введите новый каталог. Если в терминале вашего компьютера используется командная оболочка `bash` или `zsh`, это будет выглядеть следующим образом:

```
$ mkdir ch1
$ cd ch1
```

Внутри каталога выполните команду `go mod init`, чтобы отметить его как модуль Go:

```
$ go mod init hello_world
go: creating new go.mod: module hello_world
```

Подробнее о том, что такое модуль, вы узнаете в главе 10, а пока достаточно знать, что проект Go называется *модулем*. Модуль — это не просто исходный код, это еще и точная спецификация зависимостей кода внутри модуля. В корневом каталоге каждого модуля есть файл `go.mod`. Выполнение команды `go mod init` создает этот файл. Содержимое базового файла `go.mod` выглядит следующим образом:

```
module hello_world

go 1.20
```

В файле `go.mod` указываются имя модуля, минимальная поддерживаемая версия Go для него, а также любые другие модули, от которых зависит ваш модуль. Можно считать, что он похож на файл `requirements.txt`, используемый в Python, или Gemfile, применяемый в Ruby.

Вам не следует редактировать файл `go.mod` напрямую. Вместо этого используйте команды `go get` и `go mod tidy` для управления изменениями в файле. Повторюсь: все, что связано с модулями, рассматривается в главе 10.

Команда go build

Теперь давайте напишем код! Откройте текстовый редактор, введите следующий текст:

```
package main

import "fmt"

func main() {
fmt.Println("Hello, world!")
}
```

и сохраните его в каталоге `ch1` в файле с именем `hello.go`. (Да, отступы в примере выглядят неаккуратно: я хотел сделать именно так! Сейчас вы поймете почему.)

Давайте быстро пройдемся по частям созданного вами файла Go. Первая строка — это объявление пакета. Внутри модуля Go код объединен в один или несколько пакетов. Главный пакет в модуле Go содержит код, который запускает программу Go.

Далее следует объявление об импорте. В инструкции `import` перечисляются пакеты, на которые ссылается данный файл. Вы используете функцию из пакета `fmt` (обычно произносится как «фампт») из стандартной библиотеки, поэтому указываете этот пакет здесь. В отличие от других языков, Go импортирует только целые пакеты. Вы не можете ограничить импорт определенными типами, функциями, константами или переменными внутри пакета.

Все программы на Go запускаются из функции `main` в пакете `main`. Вы объявляете эту функцию, введя `func main()` и открывающуюся фигурную скобку. Как и в Java, JavaScript и C, в Go для обозначения начала и конца блоков кода используются скобки.

Тело функции состоит из одной строки. В ней говорится, что вы вызываете функцию `Println` из пакета `fmt` с аргументом `"Hello, world!"`. Как опытный разработчик, вы, скорее всего, можете догадаться, что делает вызов этой функции.

После сохранения файла вернитесь в терминал или командную строку и введите:

```
$ go build
```

Это создаст исполняемый файл с именем `hello_world` (или `hello_world.exe` в Windows) в текущем каталоге. Запустите его, и, как ни удивительно, вы увидите на экране надпись `Hello, world!`:

```
$ ./hello_world
Hello, world!
```

Имя этого двоичного файла совпадает с именем файла или пакета в объявлении модуля. Если вам нужно сохранить приложение под другим именем или в другом каталоге, используйте флаг `-o`. Например, чтобы скомпилировать код в двоичный файл с именем `hello`, нужно ввести следующую команду:

```
$ go build -o hello
```

В разделе «Тестирование небольших программ с помощью go run» главы 11 я расскажу о другом способе выполнения программы на Go.

Команда go fmt

Создатели языка Go прежде всего хотели создать язык, который позволял бы писать код эффективно. Это означало, что они должны были использовать простой синтаксис и быстрый компилятор. Кроме того, это вынудило создателей языка Go пересмотреть подход к форматированию кода. В то время как другие языки предоставляют вам большую свободу в выборе способа форматирования

кода, Go не делает этого. Он обязывает использовать стандартный формат, что существенно облегчает написание инструментов для работы над исходным кодом. Это упрощает компилятор и делает возможным создание продвинутых генераторов кода.

У этого подхода есть и еще одно преимущество. Раньше разработчики тратили очень много времени на «войну форматов». Благодаря тому что Go определяет стандартный способ форматирования кода, Go-разработчикам не приходится спорить относительно того, какой стиль размещения фигурных скобок лучше использовать или как лучше задавать отступы (https://oreil.ly/dAsbS): с помощью символов табуляции или пробелов (https://oreil.ly/dSkol). Так, например, для задания отступов в Go-коде применяются символы табуляции, и если открывающая фигурная скобка не находится в той же строке, что и начинающие этот блок команда или объявление, это считается ошибкой синтаксиса.

Среди Go-разработчиков бытует мнение, что создатели языка Go решили использовать стандартный формат для того, чтобы исключить споры о формате, и уже после этого обнаружили преимущества данного подхода в плане создания инструментов. Однако Расс Кокс (Russ Cox), ведущий разработчик Go, публично заявил о том, что его исходным мотивом было упрощение процесса создания инструментов (https://oreil.ly/rZEUv).

В набор средств разработки языка Go входит команда `go fmt`, которая автоматически исправляет пробелы в вашем коде в соответствии со стандартным форматом. Однако она не может исправить фигурные скобки в неправильной строке. Запустите эту команду следующим образом:

```
$ go fmt ./...
hello.go
```

Символы `./...` указывают инструменту Go применить команду ко всем файлам в текущем каталоге и во всех подкаталогах. Вы увидите это еще не раз, когда будете знакомиться с другими инструментами Go.

Если вы откроете файл `hello.go`, то увидите, что строка с `fmt.Println` теперь имеет правильный отступ с одной табуляцией.

Не забудьте запустить `go fmt` перед компиляцией кода и как минимум перед внесением изменений исходного кода в свой архив данных! А если вдруг забыли, сделайте отдельную операцию фиксации изменений, которая выполняет только `go fmt ./...`, чтобы не скрывать логические изменения в лавине изменений форматирования.

ПРАВИЛО ВСТАВКИ ТОЧКИ С ЗАПЯТОЙ

Команда `go fmt` не исправляет ошибочное размещение фигурной скобки не в той строке, что объясняется наличием правила вставки точки с запятой. Подобно языкам C и Java, Go требует, чтобы каждый оператор заканчивался символом точки с запятой. Однако Go-разработчики не должны расставлять символы точки с запятой вручную. Компилятор языка Go делает это автоматически, следуя очень простому правилу, суть которого изложена в кратком руководстве «Эффективный Go» (https://oreil.ly/hTOHU).

Если символу новой строки предшествует одна из следующих лексем:

- идентификатор, включая такие слова, как `int` и `float64`;
- один из базовых литералов, таких как число или строковая константа;
- лексемы `break`, `continue`, `fallthrough`, `return`, `++`, `--`, `)`, или `}`,

то лексический анализатор вставляет после нее символ точки с запятой.

Зная, что в Go действует это простое правило, легко понять, почему невозможно исправить размещение в неправильном месте фигурной скобки. Если, например, вы напишете следующий код:

```
func main()
{
    fmt.Println("Hello, world!")
}
```

то благодаря правилу вставки точки с запятой будет распознан символ `)` в конце строки `func main()`, а код приведен к следующему виду:

```
func main();
{
    fmt.Println("Hello, world!");
};
```

а это некорректно.

Правило вставки точки с запятой и связанное с ним ограничение на размещение скобок — одна из тех вещей, которые делают компилятор языка Go проще и быстрее, в то же время обеспечивая единый стиль программирования, что весьма и весьма разумно.

Команда go vet

В одном из классов ошибок код синтаксически корректен, но, скорее всего, неверен. Инструмент `go` включает команду `go vet` для обнаружения ошибок такого рода. Добавьте одну из них в программу и посмотрите, как она будет обнаружена. Измените строку `fmt.Println` в `hello.go` на следующую:

```
fmt.Printf("Hello, %s!\n")
```

Команда `fmt.Printf` похожа на `printf` в C, Java, Ruby и многих других языках. Если вы раньше не встречались с `fmt.Printf`, то это функция с шаблоном в качестве первого параметра и значениями для заполнителей в шаблоне в остальных параметрах.

В этом примере у вас есть шаблон (`"Hello, %s!\n"`) с заполнителем `%s`, но для него не указано значение. Этот код скомпилируется и запустится, но он не будет корректным. Одна из вещей, которую определяет `go vet`, — это наличие значения для каждого заполнителя в шаблоне форматирования. Запустите `go vet` на измененном коде, и он обнаружит ошибку:

```
$ go vet ./...
# hello_world
./hello.go:6:2: fmt.Printf format %s reads arg #1, but call has 0 args
```

Теперь, когда `go vet` нашел ошибку, вы можете легко ее исправить. Измените строку 6 в `hello.go` на следующую:

```
fmt.Printf("Hello, %s!\n", "world").
```

Несмотря на то что `go vet` выявляет несколько распространенных ошибок программирования, есть вещи, которые он не может обнаружить. К счастью, сторонние инструменты для проверки качества кода Go могут восполнить этот пробел. Некоторые из наиболее популярных инструментов для проверки качества кода рассматриваются в разделе «Использование сканеров качества кода» главы 11.

Точно так же, как вы должны выполнить `go fmt`, чтобы убедиться, что код отформатирован правильно, выполните `go vet`, чтобы проверить наличие возможных ошибок в корректном коде. Эти команды — лишь первый шаг в обеспечении высокого качества вашего кода. В дополнение к советам из этой книги всем разработчикам Go следует ознакомиться с руководством «Эффективный Go» (https://oreil.ly/GBRut) и страницей комментариев к обзору кода на вики-странице Go (https://oreil.ly/FHi_h), чтобы понять, как выглядит идиоматический код Go.

Выбор инструментов

Хотя, как вы уже успели убедиться, написать небольшую программу на Go можно, используя только текстовый редактор и команду `go`, для работы над более крупными проектами вам, вероятно, потребуются продвинутые инструменты. Интегрированные среды разработки для Go предоставляют множество преимуществ по сравнению с текстовыми редакторами, включая автоматическое форматирование при сохранении, завершение кода, проверку типов, создание отчетов об ошибках и интегрированную отладку. К настоящему времени уже созданы отличные средства поддержки языка Go (https://oreil.ly/cav8N) для большинства существующих текстовых редакторов и интегрированных сред разработки. Если вы еще не определились с выбором среды разработки, то знайте, что двумя наиболее популярными вариантами в случае языка Go являются редактор кода Visual Studio Code и интегрированная среда разработки GoLand.

Visual Studio Code

Если вы хотите найти бесплатную среду разработки, то лучшим выбором будет редактор кода Visual Studio Code компании Microsoft (https://oreil.ly/zktT8). За время, прошедшее с момента его появления в 2015 году, VS Code успел приобрести чрезвычайно большую популярность среди разработчиков. Хотя поддержка языка Go не входит в его «комплект поставки», его можно превратить в среду разработки для языка Go, скачав расширение для поддержки этого языка из галереи расширений.

Поддержка языка Go в редакторе VS Code обеспечивается с помощью сторонних расширений, доступ к которым осуществляется через встроенный рынок программного обеспечения. К ним относятся инструменты для разработки языка Go, отладчик Delve (https://oreil.ly/sosLu) и gopls (https://oreil.ly/TLapT) — языковой сервер для языка Go, созданный командой разработчиков этого языка. При этом компилятор для Go вы должны установить самостоятельно, а Delve и gopls для вас установит Go-расширение редактора.

Что такое языковой сервер? Это стандартная спецификация API, позволяющего редакторам реализовать такие интеллектуальные функции редактирования, как автозавершение кода, проверки качества или поиск всех мест, где переменная или функция используется в вашем коде. Для получения более подробной информации о языковых серверах и их возможностях посетите веб-сайт Language Server Protocol (https://oreil.ly/2T2fw).

Установив и настроив инструменты, можете открыть свой проект и приступить к работе над ним. Окно вашего проекта должно выглядеть примерно так, как

показано на рис. 1.1. Основы работы с Go-расширением редактора VS Code демонстрируются во вводном видео «Приступая к работе с VS Code Go» (https://oreil.ly/XhoeB).

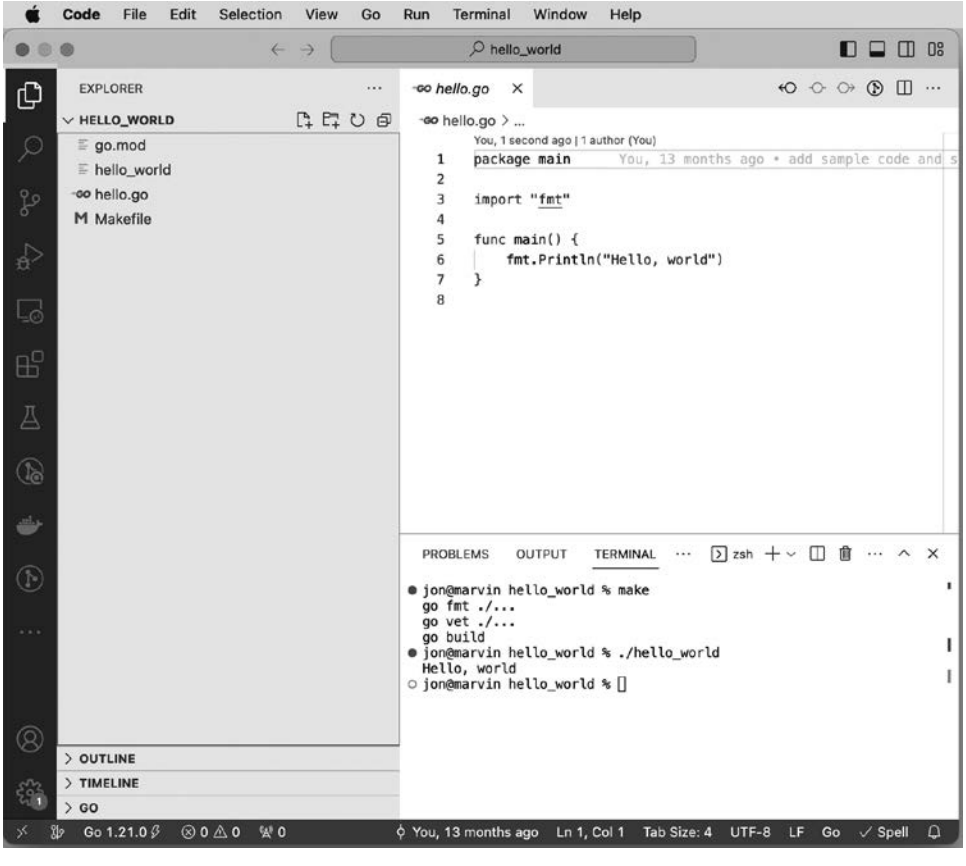

Рис. 1.1. Visual Studio Code

GoLand

GoLand (https://oreil.ly/6cXjL) — это ориентированная на язык Go интегрированная среда разработки (IDE) компании JetBrains. Хотя компания JetBrains славится в первую очередь своими инструментами для разработки на Java, это не мешает GoLand быть прекрасной средой разработки для языка Go. Как можно убедиться, взглянув на рис. 1.2, пользовательский интерфейс среды разработки GoLand выглядит практически так же, как интерфейс сред разработки IntelliJ, PyCharm,

RubyMine, WebStorm, Android Studio и любой другой IDE от компании JetBrains. Поддержка Go в GoLand включает в себя такие возможности, как рефакторинг, выделение синтаксиса, автодополнение и навигация по коду, всплывающие подсказки с описанием типов и функций, отладчик, отслеживание покрытия кода и многое другое. Помимо поддержки языка Go, среда разработки GoLand предлагает также инструменты для работы с JavaScript/HTML/CSS и базами данных SQL. В отличие от редактора кода VS Code, для использования среды разработки GoLand вам не потребуется установка подключаемого модуля.

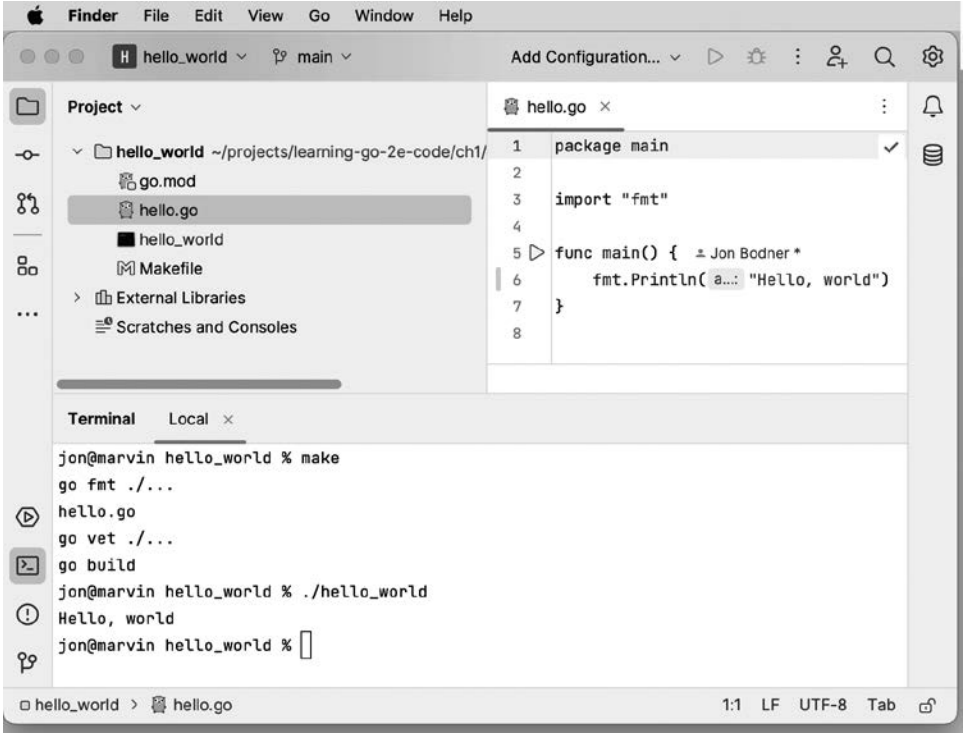

Рис. 1.2. GoLand

Если у вас уже есть подписка на IntelliJ IDEA Ultimate, можете добавить в нее поддержку Go, установив соответствующий плагин. Хотя GoLand — это коммерческое программное обеспечение, у JetBrains есть бесплатная лицензионная программа для студентов и разработчиков основного открытого исходного кода. Если вы не подходите под условия бесплатной лицензии (https://oreil.ly/48gEF), можете поработать с 30-дневной бесплатной пробной версией. После этого за использование GoLand придется платить.