1

Краткое введение в Go

Представьте, что вы разработчик, которому нужно создать утилиту командной строки. Или что у вас есть REST API и вы хотите создать сервер RESTful, который реализует этот REST API. Первый вопрос, который придет вам в голову, скорее всего, будет в том, какой язык программирования выбрать.

Рекомендуется использовать тот язык, который вы знаете лучше всего. Однако эта книга предназначена для того, чтобы дать вам возможность использовать Go для решения всех этих и многих других задач и проектов. Главу мы начнем с объяснения того, что такое Go, далее расскажем его историю и покажем, как запускать код Go. Мы объясним некоторые основные характеристики Go, например, как определять переменные, управлять потоком ваших программ и получать пользовательский ввод, после чего применим некоторые из этих концепций, создав приложение телефонной книги для командной строки.

В этой главе:

- введение в Go;
- Hello World;
- запуск Go-кода;
- важные особенности Go;
- разработка утилиты `which(1)` в Go;
- вывод информации в лог;
- обзор Go-дженериков;
- разработка базового приложения телефонной книги.

Введение в Go

Go — это язык системного программирования с открытым исходным кодом, первоначально разработанный как внутренний проект Google и ставший общедоступным еще в 2009 году. Духовными отцами Go являются Роберт Гризмер, Кен Томсон и Роб Пайк.

Официальное название языка — Go, однако его иногда называют Golang. Официальная причина в том, что домен go.org был недоступен для регистрации и вместо него был выбран golang.org. Практическая же причина в том, что, когда вы запрашиваете в поисковой системе информацию, связанную с Go, слово Go обычно интерпретируется как глагол. Кроме того, официальный хештег Go в Twitter — #golang.

Go является языком программирования общего назначения, но в основном используется для написания системных инструментов, утилит командной строки, веб-сервисов и программного обеспечения, которое работает в сетях. С помощью Go также можно обучаться программированию, плюс он является хорошим кандидатом на первый язык программирования благодаря своей немногословности, четким идеям и принципам. Go может помочь в разработке следующих видов приложений:

- профессиональные веб-сервисы;
- сетевые инструменты и серверы, такие как Kubernetes и Istio;
- серверные системы;
- системные утилиты;
- эффективные утилиты командной строки, такие как `docker` и `hugo`;
- приложения, которые обмениваются данными в формате JSON;
- приложения, обрабатывающие данные из реляционных баз данных, баз данных NoSQL или других популярных систем хранения;
- компиляторы и интерпретаторы для разрабатываемых вами языков программирования;
- системы баз данных, такие как CockroachDB, и хранилища ключей/значений, такие как etcd.

Есть много вещей, которые Go делает лучше, чем другие языки программирования, например:

- компилятор Go по умолчанию может обнаруживать большой набор глупых ошибок, которые могут привести к ошибкам в программном коде;

- Go использует меньше круглых скобок, чем C, C++ или Java, и не использует точки с запятой, что делает внешний вид исходного кода Go более удобочитаемым и менее подверженным ошибкам;
- Go поставляется с богатой и надежной стандартной библиотекой;
- Go поддерживает параллелизм «из коробки» через горутины и каналы;
- горутины действительно легковесные. Вы можете с легкостью запустить тысячи горутин на любой современной машине, обойдясь без каких-либо проблем с производительностью;
- в отличие от C, Go поддерживает функциональное программирование;
- Go-код имеет обратную совместимость, то есть более новые версии компилятора Go принимают программы, созданные с использованием предыдущей версии языка, без каких-либо изменений. Эта гарантия совместимости распространяется только на основные версии Go. Например, нет никакой гарантии, что программа на Go 1.x будет скомпилирована в Go 2.x.

Теперь, когда вы знаете, что может Go и чем он хорош, вспомним его историю.

История Go

Как упоминалось ранее, Go начинался как внутренний проект Google, который стал общедоступным еще в 2009 году. Гризмер, Томсон и Пайк разработали Go как язык для профессиональных программистов, желающих создавать надежное, стабильное и эффективное программное обеспечение, которым легко управлять. Они разрабатывали Go с прицелом на его простоту, даже если она означала, что Go не суждено стать языком программирования для всех.

На рис. 1.1 показано, какие языки программирования прямо или косвенно повлияли на Go. К примеру, синтаксис Go выглядит как синтаксис C, в то время как концепция пакетов была создана под влиянием Modula-2.

Результатами стали язык программирования, инструменты и стандартная библиотека. Помимо синтаксиса и инструментов Go, вы получаете довольно богатую стандартную библиотеку и систему типов, которая призвана избавить вас от простых ошибок, таких как неявные преобразования типов, неиспользуемые переменные и пакеты. Go-компилятор отлавливает большинство этих простых ошибок и останавливает компиляцию, пока вы как-то их не исправите. Кроме того, Go-компилятор с трудом обнаруживает сложные ошибки, такие как состояние гонки.

Если вы устанавливаете Go впервые, то можете начать со страницы https://golang.org/dl/. Однако есть большая вероятность, что в вашем варианте UNIX уже

имеется готовый к установке пакет для языка программирования Go, так что, возможно, у вас получится запустить Go с помощью вашего любимого менеджера пакетов.

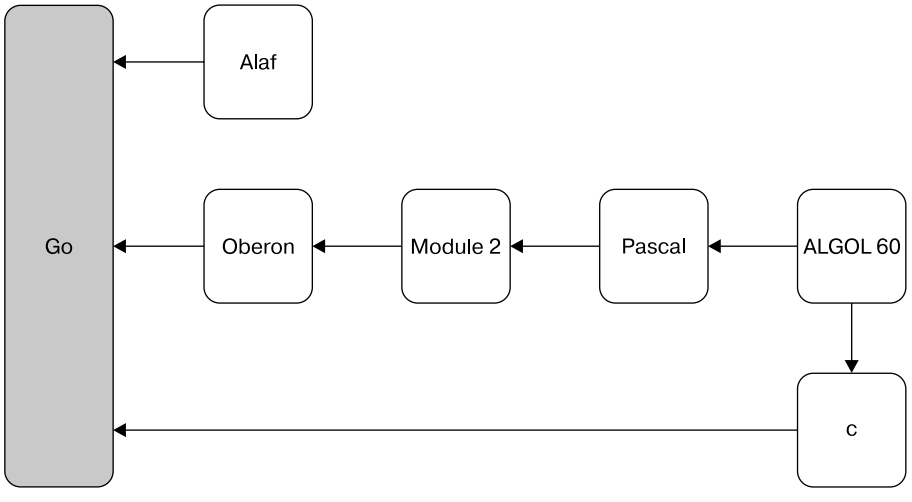

Рис. 1.1. Языки программирования, повлиявшие на Go

Почему UNIX, а не Windows

Вы можете задаться вопросом, почему мы все время говорим о UNIX и даже не обсуждаем Microsoft Windows. Для этого есть две основные причины. Первая состоит в том, что большинство Go-программ будут работать на компьютерах с Windows без каких-либо изменений, поскольку *Go переносим по сути*, и это означает, что вам не следует беспокоиться об используемой операционной системе.

Однако вам может потребоваться внести небольшие изменения в код некоторых системных утилит, чтобы они заработали в Windows. Кроме того, по-прежнему будут существовать библиотеки, работающие только на компьютерах с Windows или только на компьютерах, отличных от Windows. Вторая причина заключается в том, что многие сервисы, написанные на Go, выполняются в среде Docker — *образы Docker используют операционную систему Linux*, а это означает, что вы должны программировать свои утилиты с учетом операционной системы Linux.

Что же касается пользовательского опыта, то UNIX и Linux очень похожи. Основное отличие состоит в том, что Linux — программное обеспечение с открытым исходным кодом, тогда как UNIX — проприетарное ПО.

Преимущества Go

Go обладает рядом важных преимуществ для разработчиков, начиная с того факта, что он был разработан и поддерживается настоящими программистами. Помимо этого, Go прост в освоении, особенно если вы уже знакомы с такими языками программирования, как C, Python или Java. Вдобавок ко всему код на Go выглядит красиво (по крайней мере, на мой взгляд), и это замечательно, особенно когда вы зарабатываете на жизнь программированием и вам приходится работать с кодом ежедневно. Go-код также легко читается и имеет поддержку Unicode «из коробки», а это означает, что вы можете легко вносить изменения в существующий код Go. Наконец, Go резервирует лишь 25 ключевых слов, что делает его очень простым для запоминания. Возможно ли такое в случае C++?

Go также поставляется с поддержкой простой модели параллелизма, которая реализована с использованием *горутин* и *каналов*. Go не только управляет потоками операционной системы за вас, но и имеет эффективную среду выполнения, позволяющую создавать облегченные рабочие модули *(горутины)*, которые взаимодействуют друг с другом с помощью *каналов*. Хотя Go поставляется с богатой стандартной библиотекой, существуют действительно удобные пакеты Go, такие как `cobra` и `viper`. Они позволяют разрабатывать на Go сложные утилиты командной строки, такие как `docker` и `hugo`. Это в значительной степени подтверждается тем фактом, что в исполняемых двоичных файлах Go используется *статическая компоновка*. Иными словами, после создания они уже не зависят от каких-либо совместно используемых библиотек и содержат всю необходимую информацию.

Благодаря своей простоте Go-код предсказуем и не имеет странных побочных эффектов. Хотя Go и поддерживает *указатели*, он не поддерживает подобную C арифметику указателей, если, конечно, вы не используете пакет `unsafe`, который зачастую служит причиной множества ошибок и дыр в безопасности. Язык Go не является объектно-ориентированным, тем не менее Go-интерфейсы очень универсальны и позволяют имитировать некоторые возможности объектно-ориентированных языков, такие как *полиморфизм*, *инкапсуляция* и *композиция*.

Кроме того, последние версии Go предлагают поддержку *универсальных паттернов* (или дженериков), благодаря чему упрощается код при работе с несколькими типами данных. И последнее, но не менее важное: Go поставляется с поддержкой *сборки мусора*, а это означает, что ручное управление памятью не требуется.

Go — очень практичный и надежный язык программирования, однако он далеко не идеален.

- Хотя это скорее личное предпочтение, нежели фактический технический недостаток, но Go не имеет прямой поддержки объектно-ориентированного программирования, которое является распространенной парадигмой программирования.
- Горутины достаточно легковесны, однако не так эффективны, как потоки операционной системы. В зависимости от реализуемого приложения вполне допустимы некоторые редкие случаи, когда горутины не подойдут для решения задачи. Однако в большинстве случаев разработка вашего приложения с помощью горутин и каналов решит поставленные задачи.
- Сборка мусора выполняется достаточно быстро бóльшую часть времени и почти для всех видов приложений. Тем не менее бывают случаи, когда вам требуется работать над распределением памяти вручную, но Go так не может. На практике это означает, что Go не позволит осуществить какое-либо управление памятью вручную.

Однако существует множество сценариев, когда вы можете выбрать Go, например:

- создание сложных утилит командной строки со множеством команд, подкоманд и параметров командной строки;
- создание приложений с высокой степенью параллелизма;
- разработка серверов, работающих с API, и клиентов, которые взаимодействуют путем обмена данными во множестве форматов, включая JSON, XML и CSV;
- разработка серверов и клиентов WebSocket;
- разработка серверов и клиентов gRCP;
- разработка надежных системных инструментов для UNIX и Windows;
- изучение программирования.

Далее мы рассмотрим ряд концепций и утилит, что послужит прочной основой, после чего создадим упрощенную версию утилиты `which(1)`. В конце главы мы разработаем простое приложение для телефонной книги, которое будет развиваться по мере того, как в последующих главах мы будем продолжать знакомиться с особенностями Go.

Но сначала мы представим команду `go doc`, которая позволяет вам искать информацию о стандартной библиотеке Go, ее пакетах и их функциях. Затем, на примере программы `Hello World!`, мы покажем, как выполнить Go-код.

Утилиты go doc и godoc

Дистрибутив Go поставляется со множеством инструментов, которые облегчают вашу жизнь как программиста. Двумя такими инструментами являются подкоманда go doc и утилита godoc, которые позволяют вам просматривать документацию имеющихся функций и пакетов Go, не подключаясь к Интернету. Однако если вы предпочитаете просматривать документацию Go онлайн, то можете посетить https://pkg.go.dev/. Поскольку godoc не установлена по умолчанию, вам может потребоваться установить ее. Для этого выполните `go get golang.org/x/tools/cmd/godoc`.

Команда go doc может быть выполнена как обычное приложение командной строки, которое отображает свои выходные данные на терминале, а godoc — как приложение командной строки, которое запускает веб-сервер. В последнем случае вам понадобится браузер, чтобы просматривать документацию Go. Первая утилита аналогична команде UNIX `man(1)`, но для функций и пакетов Go.

Число после названия программы или системного вызова UNIX указывает на раздел руководства, к которому относится данная страница. Большинство имен встречаются на страницах руководства только один раз (это значит, указывать номер раздела не требуется), однако существуют имена, которые можно найти в нескольких разделах, поскольку они имеют несколько значений, например crontab(1) и crontab(5). Так что если вы попытаетесь получить страницу руководства с многозначным названием, не указав номер раздела, то получите запись с наименьшим номером раздела.

Итак, чтобы найти информацию о функции `Printf()` пакета `fmt`, вам нужно выполнить следующую команду:

```
$ go doc fmt.Printf
```

Аналогичным образом вы можете найти информацию обо всем пакете `fmt`, выполнив следующую команду:

```
$ go doc fmt
```

Вторая утилита требует выполнения godoc с параметром `-http`:

```
$ godoc -http=:8001
```

Числовое значение в этой команде, которое в данном случае равно `8001`, является номером порта, который будет прослушиваться HTTP-сервером. Поскольку мы опустили IP-адрес, godoc будет прослушивать все сетевые интерфейсы.

Вы можете выбрать любой доступный номер порта при условии, что у вас есть соответствующие привилегии. Однако обратите внимание, что порты под номерами 0–1023 предназначены для служебного пользования и могут быть задействованы только пользователем root, поэтому лучше выбрать другой порт при условии, что он не используется другим процессом.

Вы можете опустить знак равенства в представленной команде и поставить вместо него пробел. Итак, следующая команда полностью эквивалентна предыдущей:

```
$ godoc -http :8001
```

После этого вы должны перейти в своем браузере по адресу http://localhost:8001/, что даст возможность получить список доступных Go-пакетов и просмотреть их документацию. Если вы используете Go впервые, то документация по Go позволит изучить параметры и возвращаемые значения используемых функций. По мере своего путешествия по Go вы будете применять документацию Go для детального изучения функций и переменных, которые хотите использовать.

Hello World!

Ниже приведена версия программы Hello World на Go. Введите ее и сохраните как `hw.go`:

```
package main

import (
    "fmt"
)

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

Любой исходный код на Go начинается с объявления пакета `package`. В нашем случае название пакета — `main`, что имеет особое значение в Go. Ключевое слово `import` позволяет включать функционал из существующих пакетов. В нашем случае нам нужна только часть функций пакета `fmt`, который принадлежит стандартной библиотеке Go. Пакеты, которые не являются ее частью, импортируются с использованием их полного интернет-пути. Следующая важная вещь при создании исполняемого приложения — это функция `main()`. Go считает ее точкой входа в приложение и начинает выполнение приложения с кода, обнаруженного в функции `main()` пакета `main`.

`hw.go` — это Go-программа, которая работает сама по себе. Две характеристики делают ее автономным исходным файлом, который может генерировать

исполняемый двоичный файл: имя пакета, которое должно быть `main`, и наличие функции `main()`. Более подробно мы обсудим Go-функции в следующем подразделе, но еще больше о функциях и методах (функциях, привязанных к конкретным типам данных) вы узнаете в главе 5.

Введение в функции

Каждое определение Go-функции начинается с ключевого слова `func`, за которым следуют название, сигнатура и реализация. Как и в случае пакета `main`, вы можете называть свои функции как угодно — существует глобальное правило Go, которое также применяется к именам функций и переменных и действует для всех пакетов, кроме `main`: *все, что начинается со строчной буквы, считается закрытым и доступно только в текущем пакете*. Больше информации о данном правиле мы узнаем в главе 5. Единственным исключением из него являются имена пакетов, которые могут начинаться как со строчных, так и с прописных букв. Хотя я это и сказал, но не помню ни одного Go-пакета, который начинался бы с заглавной буквы!

Теперь вы можете спросить, как функции организуются и предоставляются. Ответ: с помощью пакетов и в следующем подразделе мы немного поговорим об этом.

Введение в пакеты

Go-программы организованы в пакеты, и даже самая маленькая Go-программа должна поставляться в виде пакета. Ключевое слово `package` помогает задать имя нового пакета, которое может быть любым, за одним исключением: если вы создаете исполняемое приложение, а не просто пакет, который будет совместно использоваться другими приложениями или пакетами, то должны назвать свой пакет `main`. Больше информации о разработке Go-пакетов вы получите в главе 5.

Пакеты могут использоваться другими пакетами. Фактически повторное использование существующих пакетов — хорошая практика, которая избавляет от необходимости писать много кода или реализовывать существующую функциональность с нуля.

Ключевое слово `import` используется для импорта других Go-пакетов в ваши Go-программы и дает возможность использовать некоторые или все их функциональные возможности. Go-пакет может либо быть частью богатой стандартной библиотеки Go, либо поступать извне. Пакеты стандартной библиотеки Go импортируются по имени (`os`) без необходимости указывать имя хоста и путь, в то время как внешние пакеты импортируются с использованием их полных интернет-путей, таких как `github.com/spf13/cobra`.

Запуск Go-кода

Теперь нам нужно понять, как же запустить `hw.go` или любое другое Go-приложение. В двух следующих подразделах мы выясним, что есть два способа выполнения Go-кода: в виде скомпилированного языка с использованием `go build` или в виде языка сценариев с использованием `go run`. Подробно поговорим об этих двух способах запуска Go-кода.

Компиляция Go-кода

Чтобы скомпилировать Go-код и создать двоичный исполняемый файл, вам необходимо использовать команду `go build`. Она создает исполняемый файл, который вы можете распространять и выполнять вручную. Это означает, что команда `go build` требует дополнительного шага для запуска вашего кода.

Сгенерированный исполняемый файл автоматически называется по имени файла исходного кода за вычетом расширения `.go`. Следовательно, из исходного файла `hw.go` получится исполняемый файл `hw`. Если это не то, что вам нужно, то `go build` поддерживает параметр `-o`, который позволяет изменить имя файла и путь к сгенерированному исполняемому файлу. Например, если вы хотите назвать исполняемый файл `HelloWorld`, то вам следует выполнить `go build -o HelloWorld hw.go`. Если исходные файлы не предоставлены, то `go build` ищет пакет `main` в текущем каталоге.

После этого вам необходимо самостоятельно запустить сгенерированный исполняемый двоичный файл. В нашем случае это означает выполнение либо `hw`, либо `HelloWorld`. Это показано в следующем выводе:

```
$ go build hw.go
$ ./hw
Hello World!
```

Теперь, когда мы выяснили, как компилировать Go-код, перейдем к использованию Go в качестве языка скриптов.

Использование Go в качестве языка скриптов

Команда `go run` создает именованный Go-пакет (который в нашем случае является пакетом `main`, реализованным в единственном файле), временный исполняемый файл, выполняет его и удаляет после завершения — для нас это похоже на использование языка скриптов. В нашем случае мы можем сделать следующее:

```
$ go run hw.go
Hello World!
```

Если вы хотите протестировать свой код, то лучше использовать команду `go run`. Однако если вы хотите создать и распространить исполняемый двоичный файл, то лучше использовать команду `go build`.

Важные правила форматирования и кодирования

Вы должны знать, что Go подразумевает строгие правила форматирования и кодирования, которые помогают разработчикам избежать ошибок начинающих. Изучив эти несколько правил и характерных особенностей Go и поняв, какое значение они имеют для вашего кода, вы сможете сосредоточиться на фактической функциональности вашего кода. Кроме того, компилятор Go помогает вам следовать этим правилам, выдавая выразительные сообщения об ошибках и предупреждения. Наконец, Go предлагает стандартный инструментарий (`gofmt`), который умеет форматировать ваш код за вас, так что вам не придется думать об этом.

Ниже приведен список важных правил Go, которые помогут вам при чтении этой главы.

- Go-код поставляется в пакетах, и вы можете свободно использовать функционал существующих пакетов. Однако если вы собираетесь импортировать пакет, то вам следует использовать некоторые из этих функций. Из этого правила есть несколько исключений, но в основном они связаны с инициализацией подключений и сейчас не важны.
- Вы либо используете переменную, либо вообще ее не объявляете. Это правило поможет избежать ошибок, таких как неправильное написание существующей переменной или имени функции.
- В Go есть только один способ форматирования фигурных скобок.
- Блоки кода в Go заключаются в фигурные скобки, даже если содержат лишь один оператор или вообще их не содержат.
- Go-функции могут возвращать несколько значений.
- Вы не можете автоматически конвертировать различные типы данных, даже если они из одного семейства. Например, вы не можете неявно преобразовать целое число в число с плавающей запятой.

В Go гораздо больше правил, но эти — самые важные. Вы увидите их в действии не только в текущей главе, но и в других. Пока мы рассмотрим единственный способ форматирования фигурных скобок в Go, так как это правило применяется повсеместно.

Взгляните на следующую Go-программу `curly.go`:

```
package main

import (
```

```
    "fmt"
)

func main()
{
    fmt.Println("Go has strict rules for curly braces!")
}
```

Код выглядит просто замечательно, но при запуске вас ждет разочарование, так как он не будет компилироваться. В итоге вы получите следующее сообщение о синтаксической ошибке:

```
$ go run curly.go
# command-line-arguments
./curly.go:7:6: missing function body
./curly.go:8:1: syntax error: unexpected semicolon or newline before {
```

Официальное объяснение этого сообщения об ошибке заключается в том, что Go требует использования точек с запятой в качестве ограничителей оператора во многих контекстах и компилятор автоматически вставляет требуемые точки с запятой, когда считает, что они необходимы. Следовательно, из-за открывающей фигурной скобки (`{`), помещенной в отдельную строку, компилятор Go будет вынужден вставить точку с запятой в конце предыдущей строки (`func main()`), что является основной причиной сообщения об ошибке. Правильный способ написания кода, который был приведен выше, выглядит так:

```
package main

import (
    "fmt"
)

func main() {
    fmt.Println("Go has strict rules for curly braces!")
}
```

Теперь, когда мы познакомились с этим глобальным правилом, перейдем к некоторым важным особенностям Go.

Важные особенности Go

В этом большом разделе мы обсудим важные и незаменимые функции Go, включая переменные, управление потоком программы, итерации, получение пользовательского ввода и параллелизм в Go. Мы начнем с обсуждения переменных, их объявления и использования.