ГЛАВА 2

Простые типы и объявление переменных

Теперь, когда мы уже настроили среду разработки, пришло время перейти к изучению возможностей языка Go и наилучших способов их использования. Под наилучшими здесь имеется в виду то, что код будет удовлетворять главному принципу: программа должна быть написана так, чтобы сразу было понятно, для чего она предназначена. По мере изучения вы познакомитесь с различными подходами и поймете, почему тот или иной подход позволяет получить более ясный код.

Начнем с рассмотрения простых типов и переменных. Хотя эти понятия и знакомы каждому программисту, в Go некоторые вещи делаются по-другому, что отличает его от других языков.

Встроенные типы

В языке Go есть много тех же встроенных типов, что и в других языках: логические значения, целые числа, числа с плавающей точкой и строки. Идиоматическое использование этих типов иногда вызывает затруднения у разработчиков, ранее писавших на других языках. Вы познакомитесь с этими типами и узнаете, как они применяются в Go. Но перед этим остановимся на ряде концепций, которые применимы ко всем типам.

Нулевое значение

Go, как и большинство других современных языков, по умолчанию присваивает *нулевое значение* любой переменной, которая объявлена, но не имеет значения. Наличие четко заданного нулевого значения делает код более ясным и устраняет

источник программных ошибок, часто встречающихся в программах на C и C++. При этом у каждого типа есть свое нулевое значение.

Литералы

Под *литералом* в Go понимается запись числа, символа или строки. Существует четыре вида литералов (но есть и очень редкий пятый литерал, о котором мы поговорим при обсуждении комплексных чисел).

Целочисленные литералы — это последовательности цифр. Обычно это числа с основанием 10, но с помощью специального префикса можно задать и другое основание: префикс 0b означает двоичную систему счисления (с основанием 2), префикс 0o — восьмеричную (с основанием 8) и префикс 0x — шестнадцатеричную (с основанием 16). Префикс может быть записан как в верхнем, так и в нижнем регистре. Для обозначения восьмеричного литерала также можно использовать цифру 0 без какой-либо буквы после нее, но лучше никогда этого не делать, чтобы не было путаницы.

Go позволяет разбивать литерал на несколько частей с помощью символов подчеркивания, чтобы упростить чтение длинных целочисленных литералов. Это позволяет, например, отделять разряд тысяч в числах с основанием 10 (1_234). Эти символы подчеркивания не влияют на значение числа. Единственное ограничение состоит в том, что их нельзя размещать в начале или в конце числа и рядом друг с другом. Вы можете отделить друг от друга все разряды числа (1_2_3_4), но лучше не поступайте таким образом. Используйте символы подчеркивания для улучшения восприятия, например отделяя разряд тысяч в десятичных числах или разбивая двоичные, восьмеричные и шестнадцатеричные числа на группы из одного, двух или четырех байт.

Литералы чисел с плавающей запятой содержат десятичную запятую, отделяющую дробную часть значения. Они также могут содержать показатель степени, обозначаемый буквой e, за которой следует положительное или отрицательное число (например, 6,03e23). Их можно записывать в шестнадцатеричном виде, используя префикс 0x и букву `p` для обозначения показателя степени. Как и целочисленные литералы, литералы чисел с плавающей точкой можно форматировать с помощью символов подчеркивания.

Литералы рун представляют собой символы и заключаются в одинарные кавычки. В отличие от многих других языков в Go одинарные и двойные кавычки не являются взаимозаменяемыми. Литералы рун можно записывать в виде одиночных символов стандарта Unicode (`'a'`), восьмиразрядных восьмеричных чисел (`'\141'`), восьмиразрядных шестнадцатеричных чисел (`'\x61'`), 16-разрядных шестнадцатеричных чисел (`'\u0061'`) и 32-разрядных кодов стандарта Unicode

(`'\U00000061'`). Существует также ряд рунных литералов, экранированных символом обратной косой черты, из которых чаще всего используются символ новой строки (`'\n'`), символ табуляции (`'\t'`), одинарная кавычка (`'\''`), двойная кавычка (`'\"'`) и обратная косая черта (`'\\'`).

На практике рекомендуется использовать десятичную систему счисления для представления числовых литералов и, если это не объясняется контекстом, по возможности не применять шестнадцатеричные экранированные рунные литералы. В редких случаях может использоваться восьмеричное представление, главным образом для представления значений флагов разрешения стандарта POSIX (например, восьмеричное значение 0o777 для флагов rwxrwxrwx). Шестнадцатеричный и двоичный форматы представления иногда используются в битовых фильтрах или сетевых и инфраструктурных приложениях.

Существует два способа обозначения *строковых литералов*. В большинстве случаев следует использовать *интерпретируемый строковый литерал*, который создается с помощью двойных кавычек (например, `"Greetings and Salutations"`). Такой литерал может содержать несколько рунных литералов в любом из допустимых форматов или не содержать их вовсе. Единственные символы, которые не могут здесь появиться, — символы обратной косой черты, новой строки и двойных кавычек. Если, допустим, нам нужно, чтобы второе слово фразы выводилось на новой строке и было заключено в кавычки, можно написать так: `"Greetings and\n\"Salutations\""`.

Если же необходимо включить в строку символы обратной косой черты, двойных кавычек или новой строки, используйте *необработанный строковый литерал*. Такие литералы заключаются в символы обратной одинарной кавычки (`` ` ``) и могут содержать любые символы, за исключением символа обратной кавычки. Используя необработанный строковый литерал, мы можем записать нашу фразу в двух строках следующим образом:

```
`Greetings and
"Salutations"`
```

Как вы увидите в подразделе «Явное преобразование типов» на с. 46, Go даже не позволяет складывать значения двух целочисленных переменных, если они объявлены как целые числа разной размерности. В то же время можно использовать целочисленный литерал в выражениях с плавающей запятой и даже присваивать целочисленный литерал самой переменной с плавающей запятой. Это объясняется тем, что литералы в Go представляют собой нетипизированные значения и потому могут взаимодействовать с любой переменной совместимого с литералами типа. В главе 7 вы увидите, что литералы можно использовать даже совместно с типами, определяемыми пользователем на основе простых типов. Однако это все, что позволяет сделать нетипизированный характер литералов:

невозможно присвоить строковый литерал переменной числового типа и, наоборот, числовой литерал — строковой переменной либо присвоить литерал числа с плавающей точкой целочисленной переменной. Все подобные действия компилятор посчитает ошибкой.

Литералы являются нетипизированными по той причине, что Go ориентирован на практику. Нет смысла относить литерал к какому-либо типу, если разработчик еще не указал этот тип. При этом существуют ограничения в плане размера. Вы, конечно, можете записать числовой литерал, размер которого превышает размерность любого целочисленного типа, но получите ошибку времени компиляции, если попытаетесь присвоить переменной чрезмерно большое значение литерала, как, например, в случае присвоения литерала 1000 переменной типа `byte`.

Как вы увидите в разделе, посвященном присвоению значений переменным, иногда в Go-коде не производится явное указание типа. В таких случаях Go использует для литерала *тип по умолчанию*: если в выражении ничто не указывает на то, к какому типу следует отнести литерал, то этот литерал относится к типу по умолчанию. Мы еще коснемся отнесения литералов к типу по умолчанию при обсуждении различных встроенных типов.

Логические значения

Для представления логических значений используется тип `bool`. Переменные типа `bool` могут иметь одно из двух значений: `true` или `false`. Нулевым значением для типа `bool` является значение `false`:

```
var flag bool // переменной не присвоено значение, поэтому она равна false
var isAwesome = true
```

Надо сказать, что говорить о типах переменных достаточно сложно, не обсудив сначала способы их объявления, и наоборот. Мы сначала рассмотрим несколько примеров объявления переменных и обсудим их подробнее в разделе «var или :=» на с. 48.

Числовые типы

В Go много числовых типов (12, а также несколько специальных имен для этих типов), которые разделены на три категории. Если раньше вы использовали такой язык, как JavaScript, в котором лишь один числовой тип, то вас может немного удивить такое количество. При этом, на самом деле, одни типы используются довольно часто, а другие — крайне редко. Начнем с целочисленных типов, после чего рассмотрим типы чисел с плавающей запятой и редко используемые типы комплексных чисел.

Целочисленные типы

Язык Go предлагает типы для знаковых и беззнаковых целых чисел с размерностью от одного до восьми байт. Эти типы перечислены в табл. 2.1.

Таблица 2.1. Целочисленные типы языка Go

Имя типа	Диапазон значений
int8	От –128 до 127
int16	От –32 768 до 32 767
int32	От –2 147 483 648 до 2 147 483 647
int64	От –9 223 372 036 854 775 808 до 9 223 372 036 854 775 807
uint8	От 0 до 255
uint16	От 0 до 65 535
uint32	От 0 до 4 294 967 295
uint64	От 0 до 18 446 744 073 709 551 615

Нулевым значением для всех целочисленных типов, очевидно, является значение `0`.

Специальные целочисленные типы

В Go предусмотрено несколько специальных имен для целочисленных типов. Так, `byte` — это псевдоним для типа `uint8`; значения типов `byte` и `uint8` можно присваивать друг другу, сравнивать и использовать в одной математической операции. Однако вы редко увидите имя `uint8` в Go-коде: вместо него принято применять имя `byte`.

Вторым специальным именем является имя `int`. На машинах с 32-разрядным процессором `int` означает 32-разрядное знаковое целое число, как и `int32`. На машинах с 64-разрядным процессором `int` означает 64-разрядное знаковое целое число, как и `int64`. Поскольку размерность типа `int` зависит от используемой платформы, вы получите ошибку времени компиляции, если попытаетесь присвоить друг другу, сравнить или применить в одной математической операции числа типов `int` и `int32` (или типов `int` и `int64`) без преобразования типов (подробнее об этом будет рассказано в подразделе «Явное преобразование типов» на с. 46). Целочисленные литералы по умолчанию относятся к типу `int`.

В ряде нетипичных 64-разрядных процессорных архитектур в качестве типа int используются 32-разрядные знаковые целые числа. Go поддерживает три такие архитектуры: amd64p32, mips64p32 и mips64p32le.

Третьим специальным именем является имя `uint`. Для него действуют те же правила, что и в случае типа `int`, с тем лишь отличием, что это беззнаковый тип (то есть значение этого типа представляет собой либо 0, либо положительное число).

Существует еще два имени для целочисленных типов — `rune` и `uintptr`. Мы уже сталкивались с рунными литералами выше и подробно обсудим тип `rune` в подразделе «Пробуем использовать строки и руны» с. 46. Тип `uintptr` будет рассмотрен в главе 14.

Выбор подходящего целочисленного типа

Go предлагает больше целочисленных типов, чем многие другие языки. Такое богатство выбора может вызвать вопрос: когда лучше использовать каждый из этих типов? Здесь следует использовать три простых правила.

- Если вы работаете с файлами двоичного формата или с сетевым протоколом, использующим целые числа определенной размерности или знака, выбирайте соответствующий целочисленный тип.
- Если вы пишете библиотечную функцию, которая должна работать с любым целочисленным типом, напишите две функции, одна из которых будет использовать для параметров и переменных тип `int64`, а вторая — тип `uint64`. (О функциях и их параметрах мы подробно поговорим в главе 5.)

В данном случае идиоматический подход сводится к применению типов int64 и uint64 из-за того, что в языке Go нет (по крайней мере пока) универсальных функций и перегрузки. Без этих возможностей вам пришлось бы написать несколько функций для разных типов данных. Применение int64 и uint64 позволит написать код один раз, а затем пользоваться преобразованием типов для изменения данных по необходимости.

Этот подход применяется в стандартной библиотеке Go с функциями FormatInt/FormatUint и ParseInt/ParseUint из пакета strconv. Возможны и другие ситуации: например, в пакете math/bits размер целого числа имеет значение. В таких случаях необходимо написать отдельную функцию для каждого целочисленного типа.

- Во всех остальных случаях следует использовать тип `int`.

Если нет *необходимости* в явном указании размерности или знака целочисленных значений из соображений производительности или обеспечения интеграции, используйте тип int. Пока не доказано иное, использование любого другого типа следует считать преждевременной оптимизацией.

Целочисленные операторы

Целые числа в Go поддерживают обычный набор арифметических операторов: `+`, `–`, `*`, `/` и `%` (деление по модулю). Результатом целочисленного деления является целое число, и, чтобы получить в качестве результата число с плавающей запятой, необходимо использовать преобразование типов. Старайтесь не допускать деления целого числа на 0: это вызовет панику (подробнее о паниках будет рассказано в разделе «Функции panic и recover» на с. 216).

Округление при целочисленном делении в Go производится путем отбрасывания дробной части; подробности можно найти в разделе документации языка Go, посвященном арифметическим операторам (https://oreil.ly/zp3OJ).

Для изменения переменной можно сочетать любой из арифметических операторов с `=`: например, `+=`, `–=`, `*=`, `/=`, `%=`. Так, после выполнения следующего кода переменная `x` будет иметь значение `20`:

```
var x int = 10
x *= 2
```

Для сравнения целых чисел можно использовать операторы `==`, `!=`, `>`, `>=`, `<` и `<=`.

В Go также есть операторы побитовых манипуляций для целых чисел. Вы можете выполнять побитовый сдвиг влево и вправо с помощью операторов `<<` и `>>` или применять битовые маски с помощью операторов `&` (логическое И), `|` (логическое ИЛИ), `^` (логическое исключающее ИЛИ) и `&^` (логическое И-НЕ). Как и арифметические операторы, для изменения переменной все логические операторы могут быть объединены с `=` : `&=`, `|=`, `^=`, `&^=`, `<<=`, `>>=`.

Типы чисел с плавающей запятой

В Go есть два типа чисел с плавающей запятой, которые представлены в табл. 2.2.

Таблица 2.2. Типы чисел с плавающей запятой, используемые в языке Go

Имя типа	Наибольшее абсолютное значение	Наименьшее (ненулевое) абсолютное значение
float32	3,40282346638528859811704183484516925440e+38	1,40129846432481707092372958328991613128 0e–45
float64	1,79769313486231570814527423731704356798 1e+308	4,94065645841246544176568792868221372365 1e–324

Как и для целочисленных типов, для типов чисел с плавающей точкой нулевым значением является `0`.

Работа с числами с плавающей запятой в Go происходит практически так же, как в других языках. В Go используется формат представления таких чисел, основанный на спецификации IEEE 754, которая обеспечивает широкий диапазон и ограниченную степень точности. Выбрать подходящий тип достаточно просто: за исключением тех случаев, когда нужно обеспечить совместимость с имеющимся форматом, используется тип `float64`. Литералы чисел с плавающей запятой по умолчанию относятся к типу `float64`, поэтому самым простым решением будет всегда использовать тип `float64`. Это также позволяет уменьшить проблему точности чисел с плавающей запятой, связанную с тем, что точность типа `float32` ограничивается только шестью или семью десятичными знаками после запятой. При этом не стоит волноваться о разнице в расходе памяти, за исключением того случая, когда профайлер явно укажет, что расход памяти является значительным источником проблем. (Тестирование и профайлинг мы подробно обсудим в главе 13.)

Более важный вопрос состоит в том, стоит ли вообще использовать число с плавающей запятой. В большинстве случаев ответ на него будет отрицательным. Как и в других языках, в Go числа с плавающей запятой охватывают огромный диапазон, но не могут поддерживать каждое значение в этом диапазоне, позволяя сохранить лишь ближайшее приближение. Поскольку числа с плавающей запятой не относятся к точным, их можно использовать только в ситуациях, когда допускаются приблизительные значения или известны правила их применения. Поэтому их практически не используют в компьютерной графике и научных расчетах.

Числа с плавающей запятой не могут обеспечить точное представление десятичных значений. Не используйте их для представления денежных сумм или других значений, которые требуют точного десятичного представления!

В случае чисел с плавающей запятой можно использовать все стандартные операторы сравнения и математических действий, за исключением оператора `%`. При этом стоит обратить внимание на пару интересных особенностей операции деления чисел с плавающей запятой. Деление на ноль ненулевого значения этого типа чисел возвращает в качестве результата `+Inf` или `-Inf` (плюс или минус бесконечность), в зависимости от знака числа. Деление на ноль нулевого значения с плавающей запятой возвращает в качестве результата значение `NaN` (Not a Number, «не число»).

Хотя Go и позволяет сравнивать числа с плавающей запятой с помощью операторов `==` и `!=` , лучше не делайте этого. Из-за неточности два, казалось бы, равных числа с плавающей запятой могут оказаться не равны друг другу при их сравнении. Вместо того чтобы сравнивать два числа с плавающей запятой, следует определить максимально допустимое отклонение и посмотреть, не превышает ли его разница между этими числами. Величина этого отклонения (которое иногда называют *«эпсилон»*) зависит от требуемой точности. Здесь я могу дать лишь одну простую рекомендацию: если вы не знаете, каким оно должно быть, обратитесь за советом к ближайшему знакомому математику.

IEEE 754

Как уже упоминалось выше, в Go (и в большинстве других языков программирования) используется формат представления чисел с плавающей запятой, основанный на спецификации IEEE 754.

Знакомство с этими правилами выходит за рамки этой книги, и надо сказать, что они не отличаются простотой. Например, при сохранении числа –3,1415 в виде значения типа `float64` его 64-разрядное представление в памяти будет выглядеть так:

```
1100000000001001001000011100101011000000100000110001001001101111
```

что, если быть точным, равно –3,14150000000000018118839761883.

Большинство программистов рано или поздно начинают понимать принцип представления целых чисел в двоичном виде (крайний правый разряд — это 1, следующий разряд — 2, следующий — 4 и т. д.). Принцип представления чисел с плавающей запятой выглядит совершенно иначе. Из 64 разрядов представленного выше числа один разряд используется для представления знака числа (положительного или отрицательного), 11 разрядов — для представления показателя степени, и 52 разряда — для представления нормализованной формы числа (или, как ее еще называют, *мантиссы*).

Более подробное описание стандарта IEEE 754 можно найти в Википедии (https://oreil.ly/Gc05u).

Типы комплексных чисел (возможно, они вам не потребуются)

Существует еще один редко используемый числовой тип. В языке Go превосходно реализована поддержка комплексных чисел. Если вы не знаете, что такое

комплексные числа, то, видимо, вам и не требуется эта функциональная возможность, поэтому можете спокойно пропустить данный раздел.

О поддержке комплексных чисел в Go можно сказать не так уж много. В Go определены два типа комплексных чисел: для представления действительной и мнимой части `complex64` использует значения типа `float32`, а `complex128` — значения типа `float64`. Оба этих типа объявляются с помощью встроенной функции `complex`. В Go существует несколько правил для определения типа вывода функции.

- Если оба параметра функции `complex` представляют собой нетипизированные константы или литералы, то функция возвратит нетипизированный литерал комплексного числа, по умолчанию обладающий типом.
- Если оба параметра функции `complex` представляют собой значения типа `float32`, то функция возвратит значение типа `complex64`.
- Если один параметр функции `complex` представляет собой значение типа `float32`, а второй параметр является нетипизированной константой или литералом, не выходящим за рамки диапазона типа `float32`, то функция возвратит значение типа `complex64`.
- Во всех остальных случаях функция возвратит значение типа `complex128`.

Для работы с комплексными числами можно использовать все стандартные арифметические операторы. Как и числа с плавающей запятой, комплексные числа можно сравнивать с помощью операторов `==` и `!=`, но из-за тех же проблем точности вместо этого лучше проверять величину разницы между числами. Для извлечения действительной и мнимой части комплексного числа можно использовать встроенные функции `real` и `imag` соответственно. В пакете `math/cmplx` также имеется ряд дополнительных функций для работы со значениями типа `complex128`.

Нулевым значением для обоих типов комплексных чисел является значение, и действительной, и мнимой части которого присвоено значение `0`.

В примере 2.1 показана простая программа, демонстрирующая основные приемы работы с комплексными числами. Вы можете запустить этот код в онлайн-песочнице (https://oreil.ly/fuyIu).

Пример 2.1. Комплексные числа

```
func main() {
    x := complex(2.5, 3.1)
    y := complex(10.2, 2)
    fmt.Println(x + y)
    fmt.Println(x - y)
```

```
    fmt.Println(x * y)
    fmt.Println(x / y)
    fmt.Println(real(x))
    fmt.Println(imag(x))
    fmt.Println(cmplx.Abs(x))
}
```

Запустив этот код, вы получите следующие результаты:

```
(12.7+5.1i)
(-7.699999999999999+1.1i)
(19.3+36.62i)
(0.2934098482043688+0.246390225842280 65i)
2.5
3.1
3.982461550347975
```

Здесь также можно увидеть неточность чисел с плавающей запятой.

Если вам интересно, что собой представляет пятая разновидность используемых в Go литералов простых типов, то сообщу, что это мнимые литералы, которые служат для представления мнимой части комплексного числа. Они выглядят так же, как литералы чисел с плавающей запятой, отличаясь от них лишь наличием суффикса `i`.

Несмотря на встроенную поддержку комплексных чисел, язык Go не приобрел популярности в качестве языка для числовых вычислений. Его ограниченное применение связано с тем, что другие возможности (например, поддержка матриц) не входят в состав языка, а библиотеки вынуждены использовать менее эффективные средства, такие как срезы. (О срезах и о том, как они реализованы в Go, мы подробно поговорим в главах 3 и 6 соответственно.) Но если вдруг вам потребуется вычислять множество Мандельброта в определенном месте большой программы или создать программу для решения квадратных уравнений, вы всегда сможете воспользоваться встроенной поддержкой комплексных чисел.

Если вы собираетесь писать на Go приложения для числовых вычислений, то для этой цели можно использовать сторонний пакет Gonum (https://www.gonum.org). Этот пакет в полной мере использует возможности комплексных чисел и предлагает удобные библиотеки для таких вещей, как линейная алгебра, матрицы, интегралы и статистика. Однако перед тем, как его применять, рассмотрите вариант использования других языков.

Возможно, у вас возник вопрос: зачем вообще в Go была включена поддержка комплексных чисел? Ответ прост: они показались интересными Кену Томпсону (Ken Thompson), одному из создателей языка Go (а также операционной системы