Содержание

Предисловие

Когда я услышал, что Мэтт Фарина и Мэтт Батчер начали работу над новой книгой о языке Go, это меня очень взволновало. Они оба много лет были важнейшими действующими лицами в экосистеме Go, обладают большим опытом работы и способны добавить в аромат содержания этой книги запахи специй прошлых учебных пособий. Эта книга должна стать преемницей книги «Go in Action», развивающей заложенные там основы и переводящей их в практическое русло.

Книга разбита на четыре простые в освоении части, каждая из которых имеет собственную направленность. Часть 1 освежает в памяти основные идеи языка Go. Если вы спешите и уже обладаете навыками, позволяющими уверенно писать код на языке Go, можете смело пропустить этот раздел, хотя я не рекомендую этого делать. Знакомясь с окончательным вариантом рукописи, я обнаружил в ней самородки такого размера, что, полагаю, главы этой части будут полезны всем читателям.

Часть 2 погружает читателя в недра механики управления Go-приложениями. Глава об ошибках является одним из лучших описаний Go-ошибок из всех, прочитанных мной прежде, а глава, посвященная отладке и тестированию, содержит массу полезной информации об этом важном этапе разработки, помогающем поднять приложение с уровня, требующего доказательства идеи, до уровня надежной производственной системы.

В третьей части вы узнаете о способах создания пользовательских интерфейсов. Глава по шаблонам является отличным руководством по самой сложной, как многие полагают, части экосистемы Go. Она знакомит с практическими приемами многократного использования шаблонов и создания выразительных веб-интерфейсов. Приведенные примеры соответствуют уровню книги, поскольку трудно найти примеры использования шаблонов, легко переносимые в реальные приложения. Затем вы увидите, как создавать и использовать REST API, и познакомитесь с хитростями управления версиями этого API.

Заключительная часть книги посвящена теме функциональной совместимости, необходимой практически любому современному приложению. Она позволяет глубоко погрузиться в облачную инфраструктуру и увидеть, как язык Go вписывается в модель облачных

вычислений. Заканчивается эта часть широким обзором микрослужб и методов взаимодействий между службами.

Кем бы вы ни были, новичком, только что познакомившимся с языком Go, или профессионалом с многолетним опытом, эта книга даст вам жизненно необходимые знания, которые помогут вам поднять ваши приложения на новый уровень. Авторы проделали большую работу по представлению сложной информации в согласованной манере, позволяющей ее легко усвоить. Я искренне рад публикации этой книги и тому вкладу, которое она привнесет в Go-сообщество. Я надеюсь, что вы получите то же удовольствие от ее прочтения, что и я.

— *Брайан Кетелсен* (*Brian Ketelsen*),
один их авторов книги «Go in action»,
один из основателей «Gopher academy»

Введение

При первом знакомстве с языком Go мы сразу оценили его большой потенциал. У нас появилось желание писать на нем приложения. Но это был новый язык, а во многих компаниях опасаются использовать новые языки программирования.

Это особенно касается тех компаний, где потенциал языка Go нашел бы широкое применение. Новым языкам приходится добиваться доверия, признания и принятия. Сотни тысяч разработчиков заняты в бизнесе, лидеры которого долго колеблются, перед тем как попробовать новый язык, а разработчикам, чтобы понять его выгоды и применить в разработке приложений, необходимо достаточно хорошо изучить язык.

Продвижению нового языка способствуют проекты с открытым исходным кодом, конференции, курсы и книги. Цель этой книги – помочь в изучении языка Go всем желающими, внести наш вклад в развитие Go-сообщества и разъяснить перспективы применения языка Go руководству компаний, занятых в том числе разработкой программного обеспечения.

Начиная работу над книгой, мы представляли ее целиком посвященной применению Go в разработке облачных приложений. Мы уже несколько лет работаем в сфере облачных вычислений, а Go – это язык, специально созданный для нее. Начав сотрудничать с издательством Manning, мы сочли возможным выйти за пределы облачных технологий и дополнительно охватить некоторые полезные и практичные шаблоны программирования. В результате центр внимания книги сместился из сферы облачных вычислений в сферу практического применения Go. Тем не менее корнями она по-прежнему уходит в облачные технологии.

Книга «Go на практике» – это наша попытка помочь разработчикам познакомиться с языком для продуктивного его использования, а также помочь развитию сообщества и способствовать улучшению разрабатываемого программного обеспечения.

О книге

Книга «Go на практике» описывает практические приемы программирования на языке Go. Разработчики, знакомые с основами Go, найдут в ней шаблоны и методики создания Go-приложений. Каждая глава затрагивает определенную тему (как, например, глава 10 «Взаимодействие облачных служб») и исследует различные технологии, относящиеся к ней.

Как организована книга

Одиннадцать глав разделены на четыре части.

Часть I «Основные понятия и принципы» закладывает фундамент для будущих приложений. Глава 1 описывает основы языка Go и будет полезна всем, кто еще не знаком с ним или хотел бы узнать о нем больше. В главе 2 рассказывается о создании консольных приложений и серверов, а в главе 3 – об организации параллельных вычислений в Go.

Часть II «Надежные приложения» включает главы 4 и 5, охватывающие темы ошибок, аварий, отладки и тестирования. Цель этого раздела – рассказать о создании надежных приложений, способных автоматически справляться с возникающими проблемами.

Часть III «Интерфейсы приложений» содержит три главы и охватывает диапазон тем, от генерации разметки HTML и обслуживания ресурсов до реализации различных API и работы с ними. Многие Go-приложения поддерживают взаимодействие через веб-интерфейс и REST API. Эти главы охватывают приемы, помогающие в их конструировании.

Часть IV «Облачные приложения» содержит остальные главы, посвященные облачным вычислениям и генерации кода. Язык Go разрабатывался с учетом нужд облачных технологий. В этой части демонстрируются приемы, позволяющие работать с облачными службами и приложениями и даже с микрослужбами в них. Кроме того, она охватывает приемы генерации кода и метапрограммирование.

В книге описывается 70 приемов, и в каждом случае сначала ставится задача, затем дается решение и в заключение обсуждаются причины выбора тех или иных подходов.

Соглашения и загрузка примеров кода

Весь исходный код в книге оформлен моноширинным шрифтом, `как этот текст`, чтобы выделить его в окружающем тексте. Большинство листингов сопровождаются указателями на ключевые понятия или пронумерованными маркерами для ссылки на них в тексте с пояснениями к коду.

Исходный код примеров можно загрузить на сайте издательства, по адресу: www.manning.com/books/go-in-practice, или из репозитория GitHub: github.com/Masterminds/go-in-practice.

Авторский интернет-форум

Приобретая книгу «Go на практике», вы получаете свободный доступ к закрытому веб-форуму издательства Manning Publications, где можно оставить отзыв о книге, задать технические вопросы и получить помощь авторов или других пользователей. Чтобы получить доступ к форуму и зарегистрироваться на нем, перейдите на страницу www.manning.com/books/go-in-practice. Здесь вы найдете информацию о том, как пользоваться форумом после регистрации, какого рода помощь можно получить и правила поведения на форуме.

Издательство Manning, идя навстречу пожеланиям своих читателей, предоставляет площадку для конструктивного диалога между читателями, а также между читателями и авторами. Это не обязывает авторов к участию в нем – любое их участие является добровольным (и никак не оплачивается). Задавайте авторам сложные вопросы, чтобы заинтересовать их!

Авторский интернет-форум и архивы предыдущих дискуссий будут доступны на веб-сайте издателя, пока книга продолжает печататься.

Об авторах

Мэтт Батчер – архитектор программного обеспечения в компании Deis, где занимается проектами с открытым исходным кодом. Написал несколько книг и множество статей. Имеет докторскую степень и преподает на факультете информационных технологий в университете Лойола (Чикаго, США). Мэтт увлечен созданием сильных команд и разработкой элегантных решений сложных проблем.

Мэтт Фарина работает в должности ведущего инженера группы передовых технологий в компании Hewlett Packard Enterprise. Технический писатель, лектор и регулярно вносит свой вклад в проекты с открытым исходным кодом. Занимается разработкой программного обеспечения уже более четверти века. Любит решать проблемы, применяя не только новейшие, но и самые обычные технологии, о которых часто забывают.

Часть I

ОСНОВНЫЕ ПОНЯТИЯ И ПРИНЦИПЫ

Эта часть книги содержит базовые сведения о языке Go и описание фундаментальных принципов разработки приложений на нем. Глава 1 начинается с обзора языка Go для тех, кто еще с ним не знаком.

Главы 2 и 3 охватывают основные компоненты приложений. Глава 2 описывает основы разработки приложений, включая консольные и серверные, и приемы их настройки. Глава 3 рассматривает использование сопрограмм Go. Сопрограммы являются одним из наиболее мощных и полезных элементов языка Go. Они широко используются в Go-приложениях, и вы часто будете сталкиваться с ними на протяжении всей книги.

Глава 1

Введение в язык Go

В этой главе рассматриваются следующие темы:

- *введение в язык Go;*
- *место языка Go в ландшафте языков программирования;*
- *подготовка к работе с языком Go.*

С течением времени меняется подход к разработке и запуску программного обеспечения. Инновации трансформируют понятие о вычислительной среде, где выполняются программы. Чтобы полностью использовать преимущества нововведений, необходимы языки и инструменты, изначально их поддерживающие.

Во времена, когда создавалось большинство популярных ныне языков программирования и поддерживающих их инструментов, существовали только одноядерные процессоры, соответственно, они проектировались с прицелом на работу в однопроцессорной среде. В настоящее время настольные компьютеры, серверы и даже телефоны оснащены многоядерными процессорами. На любом из них можно выполнять программы с параллельными операциями.

Меняются инструменты разработки приложений. Увеличение сложности программного обеспечения требует окружений, способных быстро собирать код и эффективно его выполнять. Тестирование больших и сложных приложений должно выполняться быстро, чтобы не тормозить процесс разработки. Многие приложения используют библиотеки. Благодаря решению проблемы нехватки дискового пространства появилась возможность поддерживать несколько версий библиотек.

Меняется подход к предоставлению инфраструктуры и программного обеспечения. Использование групп серверов, размещаемых в одном месте, и простое выделение виртуальных частных серверов стали нормой. Прежде масштабирование служб, как правило, озна-

чало необходимость инвестиций в собственное оборудование, включая средства балансировки нагрузки, серверы и хранилища. Закупка всего перечисленного, установка и ввод в эксплуатацию занимали от нескольких недель до нескольких месяцев. Теперь все это можно получить в облаке за несколько секунд или минут.

Эта глава служит введением в язык программирования Go для тех, кто не сталкивался с ним раньше. Она содержит сведения о языке, средствах поддержки, его месте в ряду других языков, установке и начале работы с ним.

1.1. Что представляет собой язык Go?

Язык Go, который часто называют *golang*, чтобы упростить поиск сведений о нем в Интернете, – это статически типизированный и компилируемый язык программирования с открытым исходным кодом, разработку которого начинала компания Google. Роберт Грисемер (Robert Griesemer), Роб Пайк (Rob Pike) и Кен Томпсон (Ken Thompson) сделали попытку создать язык для современных программных систем, способных решать проблемы, с которыми они столкнулись при масштабировании больших систем.

Вместо попытки достичь теоретического совершенства создатели языка Go отталкивались от ситуаций, часто возникающих на практике. Их вдохновлял опыт ведущих языков, созданных прежде, таких как C, Pascal, Smalltalk, Newsqueak, C#, JavaScript, Python, Java и других.

Go – не обычный статически типизированный и компилируемый язык. Его статическая типизация имеет черты, делающие ее похожей на динамическую, а скомпилированные двоичные файлы включают среду выполнения со встроенным сборщиком мусора. На структуру языка наложили отпечаток проекты, для которых он должен был использоваться в Google: большие проекты, поддерживающие масштабирование и разрабатываемые многочисленными группами разработчиков.

По сути, Go – это язык программирования, определяемый спецификациями, которые можно реализовать в любом компиляторе. Их реализация по умолчанию распространяется в виде инструмента `go`. Но Go – это не просто язык программирования. На рис. 1.1 изображена его многослойная структура.

Для разработки приложений нужен не только язык программирования, а также средства тестирования, документирования и форматирования исходного кода. Инструмент `go`, обычно используемый для

компиляции приложений, поддерживает также все вышеперечисленное. Это целый комплект инструментов для разработки приложений. Одним из наиболее важных аспектов этого комплекта является поддержка управления пакетами. Встроенная система управления пакетами, вместе с общими инструментами разработки, позволила сформировать вокруг языка программирования целую экосистему.

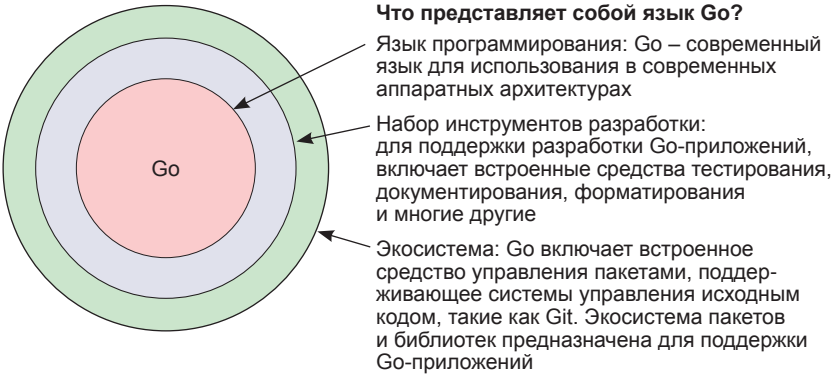

Рис. 1.1 ❖ *Слои языка Go*

Одной из определяющих особенностей Go является его простота. Когда Гисемер, Пайк и Томпсон начинали проектирование языка, новые функциональные возможности не включались в язык, пока все трое не приходили к согласию об их необходимости. Такой стиль принятия решений, а также их многолетний опыт привел к созданию простого и мощного языка. Он прост в изучении, но достаточно мощный для широкого спектра программного обеспечения.

Философию языка можно проиллюстрировать примером синтаксиса объявления переменной:

```
var i int = 2
```

Здесь создается целочисленная переменная, и ей присваивается значение 2. Поскольку имеется присваивание начального значения, определение можно сократить до:

```
var i = 2
```

При наличии начального значения компилятор автоматически определяет по нему тип. В данном случае компилятор обнаружит значение 2 и поймет, что переменная должна иметь целочисленный тип.

Но язык Go не останавливается на этом. А нужно ли само ключевое слово `var`? Нет, потому что Go поддерживает *краткое объявление переменных*:

```
i := 2
```

Это краткий эквивалент первой инструкции определения переменной. Он более чем вдвое короче, легко читается, и все это за счет автоматического определения компилятором недостающих частей.

Простота Go означает, что он поддерживает не все возможности, имеющиеся в других языках. Например, в языке Go отсутствуют тернарный оператор (обычно это `?:`) и обобщенные типы. Не содержит он и некоторых других функциональных возможностей, присутствующих в современных языках, что служит поводом для его критики, но это не должно служить поводом для отказа от использования языка Go. В мире программирования одна и та же задача часто может быть решена множеством способов. Даже если в Go отсутствуют какие-то возможности, имеющиеся в других языках, вместо них он предоставляет иные пути решения проблем, ничуть не хуже.

Несмотря на простоту ядра языка Go, встроенная система управления пакетами позволяет добавлять в него дополнительные возможности. Многие из недостающих элементов можно подключить как сторонние пакеты и включить в приложение.

Минимальный размер и сложность дают свои преимущества. Язык можно быстро освоить, и он хорошо запоминается. Это важное преимущество, когда требуется быстро исследовать чужой код.

1.2. Примечательные особенности языка Go

Поскольку язык Go разрабатывался, исходя из практических нужд, он обладает рядом особенностей, достойных особого упоминания. Все вместе эти полезные характеристики образуют строительные блоки, из которых конструируются Go-приложения.

1.2.1. Возврат нескольких значений

Одна из первых особенностей, которую вы узнаете, начав изучать Go, – функции и методы могут возвращать несколько значений. Большинство языков программирования поддерживает возврат из

функции только одного значения. Если требуется вернуть несколько значений, они встраиваются в кортеж, хэш или любое другое значение составного типа, возвращаемое функцией. Go – один из немногих языков, естественным образом поддерживающих возврат нескольких значений. Эта возможность используется в нем повсеместно, в чем легко убедиться, заглянув в исходный код библиотек и приложений, написанных на языке Go. Для примера рассмотрим следующую функцию, возвращающую две строки с именами.

Листинг 1.1 ❖ Возврат нескольких значений: `returns.go`

```
package main

import (
    "fmt"
)

func Names() (string, string) {  ← ❶ Определена как возвращающая две строки
    return "Foo", "Bar"          ← ❷ Возвращаются две строки
}

func main() {
    n1, n2 := Names()            ❸ Значения присваиваются двум переменным
    fmt.Println(n1, n2)            и выводятся

    n3, _ := Names()         ← ❹ Первое возвращаемое значение сохраняется,
    fmt.Println(n3)              второе – отбрасывается
}
```

> **СОВЕТ** Импортируемые пакеты, что используются в этой главе, такие как `fmt`, `bufio`, `net` и другие, входят в состав стандартной библиотеки. Более подробную информацию о них, включая описание программных интерфейсов и особенностей работы, можно найти на странице: https://golang.org/pkg.

Как показано в этом примере, типы возвращаемых значений объявляются в определении функции, после списка параметров ❶. В данном случае функция возвращает два строковых значения. Оператор `return` возвращает две строки ❷, в соответствии с определением. Вызывая функцию `Names`, необходимо предоставить переменные для всех возвращаемых значений ❸. Но, если требуется сохранить только одно из возвращаемых значений, для отбрасываемого значения укажите специальную переменную `_` ❹. (Можете особенно не беспокоиться о деталях реализации в этом примере. Мы еще вернемся к понятиям, библиотекам и инструментам, использованным здесь, в следующих главах.)

Опираясь на идею возврата нескольких значений, возвращаемым значениям можно дать имена и работать с ними, как с переменными. Для иллюстрации переделаем предыдущий пример, использовав именованные возвращаемые значения.

Листинг 1.2 ❖ Именованные возвращаемые значения: `returns2.go`

```
package main

import (
    "fmt"
)

func Names() (first string, second string) {
    first = "Foo"
    second = "Bar"
    return
}

func main() {
    n1, n2 := Names()
    fmt.Println(n1, n2)
}
```

❶ Именованные возвращаемые значения

❷ Присваивание значений именованным возвращаемым переменным

❸ Оператор return вызывается без значений

❹ Значения присваиваются двум переменным

Функция `Names` возвращает именованные переменные ❶, содержащие присвоенные им значения ❷. Когда оператор `return` вызывается без перечисления возвращаемых значений ❸, он возвращает текущие значения возвращаемых именованных переменных. Код, вызывающий функцию, получает возвращаемые значения ❹ и использует их так же, как в примере, где возвращаемые значения не имели имен.

1.2.2. Современная стандартная библиотека

Современные приложения решают определенные типовые задачи, такие как сетевые операции или шифрование. И чтобы не обременять разработчиков поиском библиотек для решения этих задач, стандартная библиотека Go изначально включает такие полезные функции. Остановимся на некоторых элементах стандартной библиотеки, чтобы получить общее представление о ее содержимом.

> **ПРИМЕЧАНИЕ** Полное описание стандартной библиотеки с примерами можно найти на странице: http://golang.org/pkg/.

Сетевые операции и HTTP

Под сетевыми приложениями подразумеваются и клиентские приложения, способные подключаться к другим сетевым устройствам,

и серверные, позволяющие подключаться к себе другим приложениям (листинг 1.3). Стандартная библиотека Go легко со всем этим справляется, будь то работа по протоколу HTTP, TCP (Transmission Control Protocol), UDP (User Datagram Protocol) или с применением других типовых возможностей.

Листинг 1.3 ❖ Чтение состояния по протоколу TCP: `read_status.go`

```
package main

import (
    "bufio"
    "fmt"
    "net"
)

func main() {
    conn, _ := net.Dial("tcp", "golang.org:80") ← ❶ Соединение по TCP
    fmt.Fprintf(conn, "GET / HTTP/1.0\r\n\r\n") ← ❷ Отправка строки через
    status, _ :=                                    соединение
     ➥bufio.NewReader(conn).ReadString('\n')   ❸ Вывод первой строки ответа
    fmt.Println(status)
}
```

Непосредственное подключение к порту обеспечивает пакет `net`, в котором язык Go предоставляет типовые настройки для различных типов соединений. Функция `Dial` ❶ при подключении использует указанный тип и конечную точку. В данном случае создается TCP-соединение с портом 80 по адресу `golang.org`. После подключения посылается запрос `GET` ❷ и выводится первая строка ответа ❸.

Прием входящих запросов на соединение реализуется так же просто. Только вместо функции `Dial` используется функция `Listen` из пакета `net`, которая переводит приложение в режим приема входящих соединений.

Протокол HTTP, службы REST (Representational State Transfer) и веб-серверы широко используются для взаимодействий в сети. Для их поддержки в язык Go был включен пакет `http`, содержащий реализации клиента и сервера (пример его использования приводится в листинге 1.4). Клиент достаточно прост в использовании, поскольку отвечает обычным повседневным потребностям и допускает расширение, охватывающее сложные случаи.

Листинг 1.4 ❖ HTTP-запрос GET: `http_get.go`

```
package main

import (
    "fmt"
    "io/ioutil"
    "net/http"
)

func main() {
    resp, _ :=
    ➥http.Get("http://example.com/") ← Создание HTTP-запроса GET
    body, _ :=
    ➥ioutil.ReadAll(resp.Body) ← Чтение тела ответа
    fmt.Println(string(body)) ← Вывод тела в виде строки
    resp.Body.Close() ← Закрытие соединения
}
```

Этот пример демонстрирует вывод тела простого HTTP-запроса `GET`. HTTP-клиент может намного больше, например работать с прокси, обрабатывать шифрование TLS, устанавливать заголовки, обрабатывать cookie, создавать клиентские объекты и даже подменять весь транспортный уровень.

Создание HTTP-сервера на языке Go – весьма распространенная задача. Стандартная библиотека Go обладает мощными возможностями, поддерживающими масштабирование, простыми в освоении и достаточно гибкими для применения в сложных приложениях. Созданию HTTP-сервера и работе с ним будет посвящена глава 3.

HTML

Работая над созданием веб-сервера, невозможно обойти стороной разметку HTML. Пакеты `html` и `html/template` отлично справляются с формированием веб-страниц. Пакет `html` имеет дело с экранированной и неэкранированной HTML-разметкой, а пакет `html/template` предназначен для создания многократно используемых HTML-шаблонов. Модель безопасности обработки данных прекрасно документирована, и имеется ряд вспомогательных функций для работы с HTML, JavaScript и многим другим. Система шаблонов поддерживает возможность расширения, что делает ее идеальной основой для реализации более сложных действий.

Криптография

В настоящее время криптография стала обычным компонентом приложений, используемым для работы с хэшами или шифрования конфиденциальной информации. Язык Go предоставляет часто используемые функциональные возможности, включающие поддержку MD5, нескольких версий Secure Hash Algorithm (SHA), Transport Layer Security (TLS), Data Encryption Standard (DES), Triple Data Encryption Algorithm (TDEA), Advanced Encryption Standard (AES, прежний Rijndael), Keyed-Hash Message Authentication Code (HMAC) и многих других. Кроме того, в нем имеется криптографически безопасный генератор случайных чисел.

Кодирование данных

При совместном использовании данных несколькими системами встают типичные для современных сетей задачи кодирования, такие как: обработка данных в формате base64, преобразование данных в формате JSON (JavaScript Object Notation) или XML (Extensible Markup Language) в локальные объекты.

Язык Go разрабатывался с учетом необходимости решать задачи кодирования. Внутренне Go использует только кодировку UTF-8. Это неудивительно, поскольку создатели Go прежде занимались созданием UTF-8. Но далеко не всегда при обмене между системами используется кодировка UTF-8, поэтому приходится иметь дело с самыми разными форматами данных. Для их обработки в языке Go имеются соответствующие пакеты и интерфейсы. Пакеты поддерживают такие возможности, как преобразование строк JSON в экземпляры объектов, а интерфейсы обеспечивают переключение между кодировками и добавление новых способов работы с кодировками посредством подключения внешних пакетов.

1.2.3. Параллельная обработка с помощью сопрограмм и каналов

Многоядерные процессоры стали обычным явлением. Они применяются во многих устройствах, начиная с серверов и заканчивая мобильными телефонами. Однако большинство языков программирования разрабатывалось под одноядерные процессоры, поскольку на тот момент только они и существовали.

Кроме того, среда выполнения в некоторых языках имеет глобальную блокировку, что затрудняет параллельное выполнение процедур.

Язык Go изначально ориентировался на параллельную и конкурентную обработку.

В языке Go имеются так называемые *сопрограммы*, или *go-подпрограммы*, – процедуры, способные выполняться параллельно с основной программой и другими сопрограммами. Иногда называемые *легковесными потоками*, сопрограммы управляются средой выполнения Go, которая помещает их в соответствующие потоки операционной системы и утилизирует их с помощью сборщика мусора, когда они становятся ненужными. При наличии в системе процессора с несколькими ядрами go-подпрограммы способны выполняться параллельно, поскольку различные потоки могут выполняться одновременно на разных ядрах. Для разработчика создание сопрограмм выглядит не сложнее написания обычных функций. Рисунок 1.2 иллюстрирует работу go-подпрограмм.

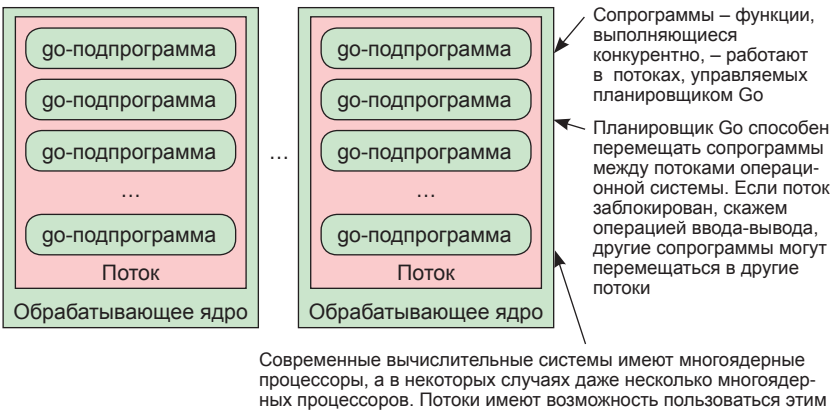

Рис. 1.2 ❖ *Go-подпрограммы выполняются в потоках, распространяемых на все доступные ядра*

Для иллюстрации рассмотрим сопрограмму, ведущую счет от 0 до 4 одновременно с тем, как основная программа печатает `Hello World`, как показано в листинге 1.5.

Листинг 1.5 ❖ Результат конкурентного вывода

```
0
1
Hello World 2
3
4
```

Такой вывод получается в результате одновременного выполнения двух функций. Соответствующий код, следующий ниже, напоминает обычную процедурную программу, с небольшим исключением.

Листинг 1.6 ❖ Конкурентный вывод

```
package main

import (
    "fmt"
    "time"
)

func count() {
    for i := 0; i < 5; i++ {
        fmt.Println(i)
        time.Sleep(time.Millisecond * 1)
   }
}

func main() {
    go count()
    time.Sleep(time.Millisecond * 2)
    fmt.Println("Hello World")
    time.Sleep(time.Millisecond * 5)
}
```

❶ Функция, выполняемая как go-подпрограмма

← ❷ Вызов go-подпрограммы

Функция `count` ❶ – это самая обычная функция, которая выводит числа от 0 до 4. Чтобы запустить ее параллельно с основной программой, используется ключевое слово `go` ❷. Благодаря этому функция `main` продолжает работать, как обычно, после вызова функции `count`. В результате функции `count` и `main` выполняться одновременно.

Передача данных между сопрограммами может осуществляться через каналы. По умолчанию они блокируют выполнение, позволяя сопрограммам синхронизироваться друг с другом. Рисунок 1.3 иллюстрирует это на простом примере.

В этом примере переменная передается от одной сопрограммы к другой через канал. Эта операция работает, даже когда сопрограммы выполняются параллельно, на различных ядрах процессора. В примере показана однонаправленная передача информации, но вообще каналы могут быть не только однонаправленными, но и двунаправленными.

В следующем листинге приводится пример использования канала.

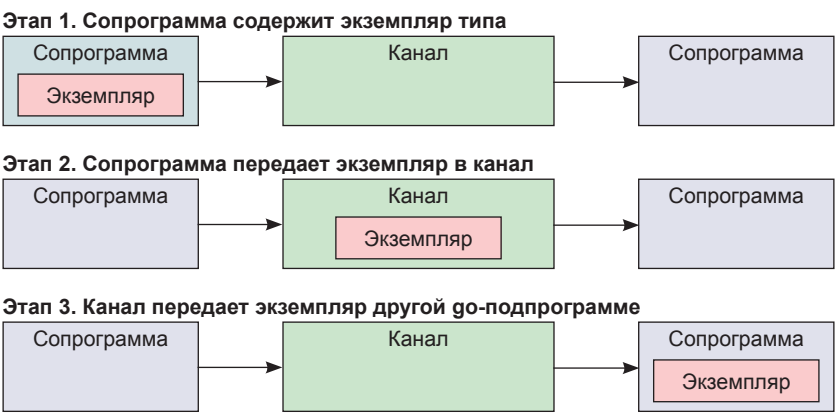

Рис. 1.3 ❖ *Передача переменных между go-подпрограммами*

Листинг 1.7 ❖ Использование каналов: `channel.go`

```
package main import (

    "fmt"
    "time"
)

func printCount(c chan int) {  ← ❶ Канал для передачи целого значения
    num := 0
    for num >= 0 {
        num = <-c  ← ❷ Ожидание целого значения
        fmt.Print(num, " ")
    }
}

func main() {
    c := make(chan int) ← ❸ Создание канала
    a := []int{8, 6, 7, 5, 3, 0, 9, -1}
    go printCount(c) ← ❹ Вызов сопрограммы
    for _, v := range a {  ← ❺ Запись целого значения в канал
        c <- v
    }
    time.Sleep(time.Millisecond * 1) ← ❻ Функция main приостанавливается
    fmt.Println("End of main")          перед завершением
}
```

В начале функции `main` создается канал `c` для передачи целого числа ❸ между сопрограммами. Функция `printCount`, вызываемая как сопрограмма, передает значение в канал ❹. Согласно определению

параметра функции printCount, канал идентифицируется как канал для передачи целых чисел ❶. В цикле for функция printCount ожидает появления в канале c целого числа ❷ и присваивает его переменной num. Тем временем функция main выполняет обход списка целых чисел и поочередно передает их в канал c ❺. После передачи в канал очередного целого числа из main ❺ оно присваивается переменной num в функции printCount ❷. Функция printCount продолжит выполнение цикла, пока в следующей итерации вновь не достигнет оператора чтения из канала ❷, и снова приостановится до появления в канале следующего целого значения. После очередной итерации в функции main и записи нового целого значения выполнение продолжится без задержек. По завершении функции main будет закончено выполнение всей программы, поэтому необходимо выполнить паузу в одну секунду ❻, чтобы позволить функции printCount завершить выполнение раньше, чем это сделает функция main. Если запустить этот код, он выведет следующее.

Листинг 1.8 ❖ Вывод с использованием канала

```
8 6 7 5 3 0 9 -1 End of main
```

Совместное использование каналов и сопрограмм обеспечивает возможности, напоминающие легковесные потоки выполнения или внутренние микрослужбы, обменивающиеся данными через специализированный прикладной программный интерфейс. Они могут объединяться в цепочки или комбинироваться различными способами.

Go-подпрограммы (сопрограммы) и каналы являются двумя самыми мощными особенностями языка Go, которые не раз будут упоминаться на протяжении всей книги. Вы увидите, как использовать их для реализации серверов, передачи сообщений и задержки выполнения задач. Также будут описаны шаблоны проектирования, основанные на сопрограммах и каналах.

1.2.4. Go – больше, чем язык

Разработка современных масштабируемых и простых в обслуживании приложений требует множества элементов. Компиляция – лишь один из этапов в этом процессе. Так было задумано изначально. Go – это больше, чем язык и компилятор. Утилита go – это целый комплекс инструментов для управления пакетами, тестирования, документирования и многого другого, помимо компиляции кода на языке Go в исполняемые файлы. Рассмотрим несколько компонентов в этом наборе.

Управление пакетами

Диспетчеры пакетов существуют для многих современных языков программирования, но у скольких из них функция управления пакетами является встроенной? В языке Go она встроенная, и тому есть две веские причины. Самая очевидная – продуктивность программиста. Вторая причина – ускорение компиляции. Управление пакетами разрабатывалось с учетом нужд компилятора. Это один из аспектов, определяющих быстроту компиляции.

Идею пакетов в Go проще всего изучать на примере стандартной библиотеки (ее демонстрирует следующий листинг), которая организована на основе системы управления пакетами.

Листинг 1.9 ❖ Простой импорт пакета

```
package main

import "fmt" ← Импорт пакета fmt

func main() {
    fmt.Println("Hello World!") ← Использование функции пакета fmt
}
```

Пакеты импортируются по именам. В данном случае `fmt` – это пакет с функциями форматирования. Все имеющееся в пакете доступно при использовании имени пакета в виде префикса. Как, например, вызов `fmt.Println` в предыдущем примере.

Импортируемые пакеты можно группировать, но в этом случае они должны указываться в алфавитном порядке. После импортирования пакетов, как показано ниже, на пакет `net/http` можно ссылаться, используя префикс `http`.

```
import (
    "fmt"
    "net/http"
)
```

Механизм импортирования работает также с пакетами, не входящими в стандартную библиотеку Go, и ссылки на такие пакеты используются точно так же:

```
import (
    "golang.org/x/net/html" ← Ссылка на внешний пакет по адресу URL
    "fmt"
    "net/http"
)
```

Имена пакетов являются уникальными строками и могут быть чем угодно. Здесь для ссылки на внешний пакет использован URL-адрес. Это позволяет языку Go опознать уникальный ресурс и получить его.

```
$ go get ./...
```

Команда `go get` принимает путь к отдельному пакету, например `golang.org/x/net/html`, для загрузки отдельного пакета, или `./...`. В последнем случае Go выполнит обход базы кода и загрузит все внешние пакеты. В данном случае (см. пример выше) Go обнаружит инструкцию `import`, выделит из нее внешнюю ссылку, загрузит пакет и сделает его доступным в текущей рабочей области.

Язык Go может загружать пакеты из систем управления версиями. Он поддерживает Git, Mercurial, SVN и Bazaar, если они установлены в локальном окружении. В этом случае Go загружает базу кода из Git и проверяет последнюю версию в ветви по умолчанию.

Система управления пакетами поддерживает далеко не все, чего можно было бы пожелать. Она реализует лишь базовые возможности, которые можно использовать непосредственно или как основу для системы с более широкими возможностями.

Тестирование

Тестирование – общепринятый элемент разработки программного обеспечения, и по мнению некоторых – весьма существенный. В Go имеется полноценная система тестирования, включающая пакет в стандартной библиотеке, средство выполнения тестов из командной строки, средство оценки охвата кода тестами и средство обнаружения состояний «гонки за ресурсами».

Процесс создания и выполнения тестов достаточно прост, как показано в следующем листинге.

Листинг 1.10 ❖ Пример Hello World: `hello.go`

```
package main

import "fmt"

func getName() string {
    return "World!"
}

func main() {
    name := getName()
    fmt.Println("Hello ", name)
}
```

Начнем с варианта приложения «Hello World», содержащего функцию `getName`, которая и будет тестироваться. В соответствии с соглашением в языке Go имена файлов с тестами заканчиваются на `_test.go`. Данный суффикс сообщает среде Go, что этот файл предназначен для тестирования и должен игнорироваться при сборке приложения.

Листинг 1.11 ❖ Тест для примера Hello World: `hello_test.go`

```
package main

import "testing"

func TestName(t *testing.T) {
    name := getName()
    if name != "World!" {
        t.Error("Respone from getName is unexpected value")
    }
}
```

❶ Инструмент запуска тестов вызывает функции, начинающиеся с Test

❷ Отчет об ошибке, если тест не пройден

Если запустить команду `go test`, она выполнит функцию с именем, начинающимся с префикса `Test` ❶, – в данном случае функцию `TestName` – и передаст ей вспомогательную структуру `t`. Структура содержит несколько полезных функций, например для вывода сообщений об ошибках. За дополнительной информацией обращайтесь к документации описанием пакета `testing`. Если значение переменной `name` окажется неправильным, тест выведет сообщение об ошибке ❷.

В следующем листинге показан вывод команды `go test`, сообщающий имя пакета и результаты его тестирования. Для проверки текущего пакета и всех пакетов во вложенных подкаталогах можно использовать команду `go ./...`.

Листинг 1.12 ❖ Выполнение команды `go test`

```
$ go test
PASS
ok  go-in-practice/chapter1/hello     0.012s
```

Если метод `getName` вернет строку, отличную от `"World!"`, результат будет иным. В следующем примере система тестирования сообщает о месте, где тест выявил ошибку, имя теста, имя файла с тестом и номер строки с ошибкой. Для следующего примера мы изменили метод `getName`, чтобы он возвращал строку, отличную от `"World!"`.

Листинг 1.13 ❖ Ошибка, выявленная при тестировании командой `go test`

```
$ go test
--- FAIL: TestName (0.00 seconds)
```

```
      hello_test.go:9: Response from getName is unexpected value
FAIL
exit status 1
FAIL    go-in-practice/chapter1/hello              0.010s
```

В состав Go входят все необходимые инструменты для тестирования. Более того, Go сам использует их. Для тех, кому потребуется что-то еще, например средства поддержки разработки, основанной на поведении (Behavior-Driven Development, BDD), или нечто другое, что можно найти в других языках, существуют внешние пакеты, расширяющие встроенные функциональные возможности. В Go-тестах можно использовать весь спектр возможностей языка, включая пакеты.

Охват кода

Помимо выполнения тестов, система тестирования поддерживает создание отчетов с полной детализацией охвата кода тестами, вплоть до уровня инструкций, как показано на рис. 1.14.

> **ПРИМЕЧАНИЕ** В версии Go 1.5 команды получения оценки охвата тестами стали частью ядра Go. До версии 1.5 команда `cover` относилась к дополнительным инструментам.

Чтобы получить оценку охвата тестами, выполните следующую команду:

```
$ go test -cover
```

Добавление флага `-cover` в команду `go test` заставит ее вывести информацию об охвате тестами вместе с прочими сведениями о выполнении тестов.

Листинг 1.14 ❖ Тестирование с получением сведений об охвате тестами

```
$ go test -cover
PASS
Coverage: 33.3% of statements
ok   go-in-practice/chapter1/hello        0.011s
```

Но предоставлением этой информации система оценки охвата кода не ограничивается. Сведения об охвате можно экспортировать в файлы для использования другими инструментами. Также эти отчеты можно отображать с помощью встроенных средств. На рис. 1.4 показано, как выглядит отчет в веб-браузере, содержащий данные о тестируемых инструкциях.

Рис. 1.4 ❖ *Отчет об охвате кода тестами в веб-браузере*

Обычно такие отчеты содержат данные с детализацией, вплоть до строки. Простым примером служат инструкции `if` и `else`. Среда Go покажет, какие инструкции были выполнены, а какие остались не охваченными тестированием.

> **СОВЕТ** Более полную информацию об инструменте `cover` можно найти в блоге языка Go: http://blog.golang.org/cover.

Тестированию в языке Go придается большое значение, и мы еще уделим время этой стороне программирования в главе 4.

Форматирование

Как предпочтительнее оформлять отступы в исходном коде, символами табуляции или пробелами? Вопросы форматирования и стиля затрагиваются и обсуждаются всякий раз, когда речь заходит о соглашениях оформления кода. Сколько можно было бы сэкономить времени, отказавшись от этих дебатов? Пользователям языка Go нет смысла тратить время на обсуждение форматирования или других языковых идиом.

На странице http://golang.org/doc/effective_go.html можно ознакомиться с руководством «*Effective Go*»[1], посвященным идиоматическим особенностям языка Go. В нем описаны стили и соглашения, используемые всеми членами Go-сообщества. Эти соглашения облегчают чтение и изменение написанных на языке Go программ.

Среда Go включает встроенный инструмент форматирования, поддерживающий большинство рекомендаций по оформлению. Доста-

---

[1] Краткий пересказ на русском можно найти по адресу: http://golang-club.blogspot.ru/2016/03/effective-go.html. – *Прим. ред.*

точно запустить команду `go fmt` в корневом каталоге пакета, чтобы Go обошел все файлы `.go` в пакете и привел их в соответствие с каноническим стилем. В команде `go fmt` можно дополнительно указать путь к пакету или `./...` (для обхода всех подкаталогов).

Многие редакторы поддерживают автоматическое форматирование через встроенные средства или в виде дополнительных пакетов. К ним относятся: Vim, Sublime Text, Eclipse и многие другие. Например, пакет GoSublime для редактора Sublime Text производит форматирование файла при его сохранении.

1.3. Место языка Go среди других языков программирования

Популярный репозиторий GitHub хранит проекты, написанные на сотнях разных языков. Рейтинговый список TIOBE языков программирования показывает снижение популярности наиболее распространенных языков. Это дает шанс другим языкам. Давайте посмотрим, какое место занимает Go среди множества других языков программирования.

Изначально Go разрабатывался как системный язык. То, что часто называют *облачными вычислениями*, обычно относят к одной из форм системного программирования. Язык Go был разработан с учетом особенностей систем, в которых он должен был использоваться.

Системные языки должны обладать определенными особенностями. Например, Go можно применять в случаях, где обычно используются языки C или C++, но он не годится для встраиваемых систем. Среда выполнения Go обладает системой сборки мусора, которая не будет нормально работать во встраиваемых системах с ограниченными ресурсами.

Сравнение Go с другими популярными языками программирования позволит определить его положение по отношению к этим языкам. Мы считаем, что Go оптимально подходит для разработки определенных приложений, но не собираемся вести дебаты о выборе языка программирования. Для правильного выбора необходимо учитывать не только характеристики языков.

1.3.1. C и Go

Язык Go с самого начала рассматривался как альтернатива языку C для разработки приложений. Поскольку источником вдохновения

при разработке этого языка был язык C (C по-прежнему остается одним из популярных языков, если не самым популярным), имеет смысл рассмотреть сходства и различия этих языков.

Программы на обоих языках – Go и C – компилируются в машинный код целевой операционной системы и архитектуры. Оба языка схожи по своему стилю, но Go далеко выходит за рамки языка C по своим возможностям.

Язык Go имеет среду выполнения, поддерживающую такие функции, как управление потоками выполнения и сборка мусора. Разрабатывая приложения на языке Go, вы освобождаетесь от обязанности управлять потоками выполнения и выполнять операции по сборке мусора, как это принято в других языках, оснащенных сборщиком мусора. В языке C управление потоками и памятью возлагается на программиста. Вся работа с потоками и связанные с ними действия выполняются приложением. А для работы с памятью сборщик мусора в принципе не предусмотрен.

Язык C и его объектно-ориентированные производные, такие как C++, обеспечивают разработку весьма широкого спектра приложений. На языке C можно писать высокопроизводительные встраиваемые системы, крупномасштабные облачные и сложные настольные приложения. Язык Go в основном предназначен для использования в качестве системного языка и языка создания облачных платформ. Преимуществом Go является высокая продуктивность.

Среда выполнения и набор инструментов Go доступны изначально. Их возможности позволяют очень быстро разрабатывать приложения и тратить на их создание гораздо меньше усилий, чем требуется для написания сопоставимого приложения на языке C. Например, для использования всех четырех ядер процессора на сервере Go-приложение может использовать сопрограммы. В аналогичном приложении на C придется вдобавок написать код, запускающий потоки выполнения, управляющий их работой и выполняющий переключение между ними.

Компиляция приложений на C может занимать продолжительное время. Это особенно верно при наличии внешних зависимостей и необходимости их компиляции. Высокая скорость компиляции Go-приложений была одной из целей разработки языка Go, и она выполняется быстрее, чем компиляция кода на языке C. Когда приложение разрастается до таких размеров, что его компиляция начинает занимать минуты или даже часы, экономия на времени компиляции может существенно повлиять на производительность разработки.

Компиляция Go-приложений выполняется настолько быстро, что компиляция многих приложений и пакетов их зависимостей занимает считанные секунды или даже еще меньше.

C + Go = cgo

Язык Go поддерживает использование библиотек, написанных на C, в программах на Go. В состав Go входит библиотека инструментов поддержки совместимости с языком C. Эта библиотека облегчает, например, переход от строк в стиле C к строкам языка Go. Кроме того, инструменты Go позволяют компоновать программы из исходного кода на языках C и Go. Язык Go также поддерживает обертки Simplified Wrapper and Interface Generator (SWIG). Получить общее представление об имеющихся возможностях можно с помощью команд `go help c` и `go doc cgo`.

1.3.2. Java и Go

Java долгое время является одним из самых популярных языков программирования и используется для разработки широкого спектра проектов, от серверных до мобильных Android-приложений и кроссплатформенных приложений для настольных компьютеров. Go изначально разрабатывался как системный язык. С течением времени стало возможным использовать его для разработки мобильных и веб-приложений, тем не менее Go не позволяет с легкостью писать настольные приложения. Его превосходные качества проявляются только при использовании по назначению.

Учитывая популярность Java и возможность его применения для создания широкого спектра приложений, почему у кого-то может возникнуть желание пользоваться языком Go? Языки Java и Go имеют схожий синтаксис, но в действительности они сильно отличаются. Программы на Go компилируются в единственный двоичный файл, предназначенный для определенной операционной системы. Он содержит среду выполнения Go, все импортированные пакеты и само приложение, то есть все, что необходимо для выполнения программы. В Java используется другой подход. Приложения на Java требуют установки среды выполнения в операционной системе. Они упаковываются в файл, который может выполняться в любой системе с соответствующей средой выполнения. Приложения требуют наличия совместимой версии среды выполнения.

Это различие подходов, проиллюстрированное на рис. 1.5, определяет способ использования приложений. Для развертывания Go-

приложения на сервере достаточно установить один файл. Чтобы развернуть Java-приложение, вместе с самим приложением требуется установить и настроить среду выполнения Java.

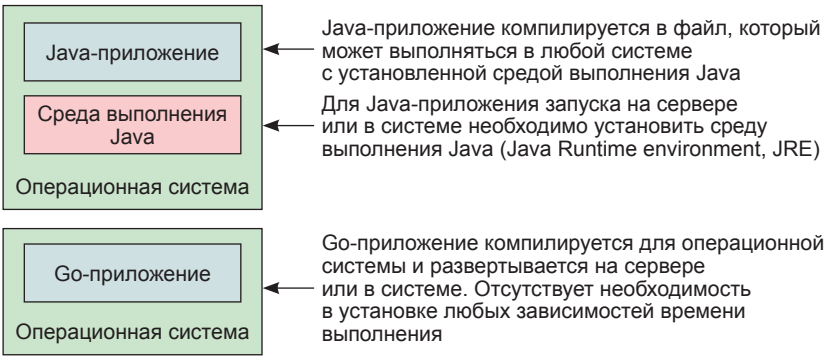

Рис. 1.5 ❖ *Выполнение программ на языках Java и Go в операционной системе*

Другое ключевое различие между языками Java и Go заключается в самом порядке выполнения приложения. Программы на языке Go компилируются в двоичный код и выполняются операционной системой. Java-приложения выполняются виртуальной машиной (VM), обычно содержащей динамический компилятор (Just-In-Time, JIT). JIT-компилятор способен анализировать выполнение кода в текущем контексте и оптимизировать его.

Возникает важный вопрос: что работает быстрее – код, выполняемый виртуальной машиной с JIT-компилятором, или предварительно скомпилированный двоичный код? Ответ на него не прост, поскольку многое зависит от JIT-компилятора, выполняемого кода и многого другого. Сравнительные тесты программ с аналогичной функциональностью не выявили явного лидера.

1.3.3. Python, PHP и Go

Python и PHP – два наиболее популярных динамических языка. Python превратился в один из самых популярных языков, изучаемых и используемых в университетах. Его можно применять в различных целях, от создания облачных приложений до разработки веб-сайтов и утилит, запускаемых из командной строки. PHP – один из самых популярных языков программирования для создания веб-сайтов. Хотя эти два языка имеют множество различий, между ними наблю-

даются определенные сходства, подчеркивающие некоторые из подходов, используемых в Go.

Python и PHP – языки с динамической типизацией, в то время как Go является языком со статической типизацией, поддерживающим некоторые черты, характерные для динамической типизации. Динамические языки осуществляют проверку типов данных во время выполнения и даже выполняют их преобразование на лету. В языках со статической типизацией проверка типов данных выполняется на основании статического анализа кода. Язык Go также поддерживает преобразование типов в определенных границах. В некоторых случаях переменные одного типа могут преобразовываться в переменные другого типа. Это может выглядеть необычным для языка со статической типизацией, но иногда очень удобно.

Обычно, когда PHP и Python используются для создания веб-сайтов или приложений, они располагаются за веб-сервером, таким как Nginx или Apache. Веб-браузер подключается к веб-серверу, обрабатывающему соединения, а веб-сервер передает информацию среде выполнения и программе, написанной на этих языках.

Go имеет встроенный веб-сервер, как показано на рис. 1.6. Такие приложения, как веб-браузеры, подключаются напрямую к Go-приложению, и оно само управляет соединением. Это обеспечивает более низкоуровневое управление и взаимодействие с подключившимися приложениями. Веб-сервер, встроенный в Go, в состоянии параллельно обрабатывать множество соединений, используя все функциональные преимущества языка.

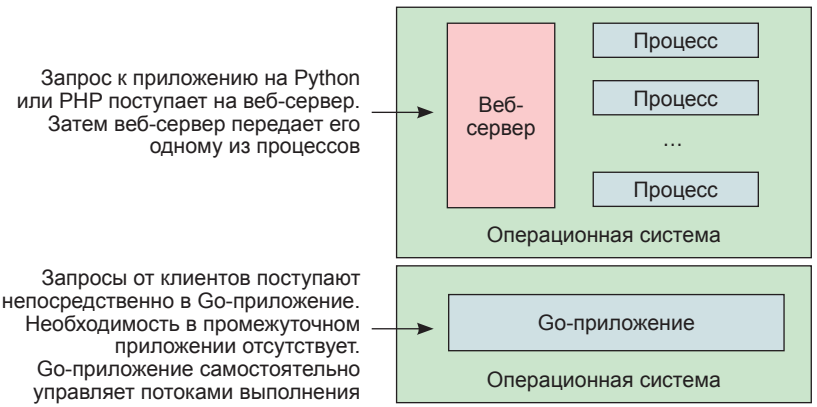

Рис. 1.6 ❖ *Прохождение запросов от клиентов к приложениям на Python, PHP и Go*

Одно из преимуществ размещения приложений на Python и PHP за веб-сервером заключается в том, что он берет на себя обслуживание потоков выполнения и параллельных подключений. В Python предусмотрена глобальная блокировка интерпретатора, позволяющая одновременно выполняться только одному потоку. PHP-приложение, как правило, выполняется от начала до конца процесса. Для одновременного доступа к приложению нескольких клиентов перед приложением должен находиться веб-сервер, запускающий обработку соединений в отдельных процессах.

Для конкурентного обслуживания подключений веб-сервер, встроенный в Go, использует сопрограммы. Среда выполнения Go распределяет сопрограммы между потоками выполнения. Более подробно об этом рассказывается в главе 3. Для обслуживания нескольких соединений запускается несколько процессов с приложениями на Python и PHP, но Go-приложение использует одну общую среду, что позволяет совместно использовать ее ресурсы там, где это имеет смысл.

Внутренняя реализация Python и PHP написана на C. Их встроенные объекты, методы и функции, реализованные на C, выполняются быстрее, чем объекты, методы и функции приложений. Исходный прикладной код преобразуется в промежуточную интерпретируемую форму. Один из советов по повышению производительности критических участков кода, написанных на Python и PHP, заключается в переводе их на язык C.

Go-приложения компилируются в выполняемый двоичный код. То есть программный код из стандартной библиотеки и из приложения компилируется в машинный код. Между ними нет никакой разницы.

1.3.4. JavaScript, Node.js и Go

Язык JavaScript был создан практически за 10 дней. Он стал одним из самых популярных языков благодаря встраиванию во все основные веб-браузеры. В последнее время JavaScript используется на серверах, в настольных приложениях и в других областях. Это стало возможным благодаря платформе Node.js.

Языки Go и JavaScript, главным образом благодаря Node.js, могут заполнять аналогичные ниши, но делают это по-разному. Исследование этих различий поможет лучше разобраться в роли языка Go.

JavaScript использует однопоточную модель. Несмотря на поддержку асинхронного ввода/вывода, который может выполняться в отдельных потоках, основная программа выполняется только в одном потоке. Когда выполнение кода в основном потоке требует зна-

чительного количества времени, он блокирует выполнение другого кода. В Go используется многопоточная модель, где среда выполнения управляет выполнением сопрограмм в отдельных потоках. Модель Go с возможностью выполнения потоков на нескольких ядрах способна более полно использовать доступное оборудование, чем однопоточная модель JavaScript.

Платформа Node.js использует движок V8, разработанный в компании Google и входящий в состав в Google Chrome. Движок V8 содержит виртуальную машину с JIT-компилятором. Концептуально это решение схоже с Java. Виртуальная машина и JIT-компилятор способны несколько повысить производительность. Движок V8 следит за длительностью выполнения программ и с течением времени повышает производительность. Например, он может распознать выполнение цикла и для увеличения производительности преобразовать часть программного кода непосредственно в машинный код. Движок V8 способен определять типы переменных, даже при том, что JavaScript является динамически типизированным языком. Чем дольше работает программа, тем больше движку V8 удается узнать о ней и применить собранную информацию в целях улучшения производительности.

Программы на Go, напротив, сразу компилируются в машинный код и выполняются со скоростью машинного кода со статической типизацией. Здесь нет нужды в JIT-компиляторе для увеличения скорости выполнения. Как и в языке C, нет необходимости в применении JIT-компиляции.

Работа с пакетами

Экосистема Node.js включает сообщество и инструментальные средства для обработки и распространения пакетов. Они могут варьироваться от библиотек до утилит командной строки и полноценных приложений. В комплект Node.js обычно входит диспетчер пакетов `npm`. Центральный репозиторий с информацией о доступных пакетах находится на сайте www.npmjs.org. Когда поступает команда загрузить пакет, диспетчер извлекает метаданные о нем из центрального репозитория и загружает пакет из исходного месторасположения.

Как упоминалось ранее, язык Go имеет собственную систему работы с пакетами. В отличие от Node.js, где метаданные и дополнительная информация располагаются в центральном репозитории, язык Go не имеет центрального хранилища, и пакеты загружаются из исходного местоположения.

1.4. Подготовка и запуск программы на языке Go

Имеется несколько вариантов знакомства с языком Go, в зависимости от ваших предпочтений.

Самый простой способ начать работу с языком Go – пройти тур на странице http://tour.golang.org, демонстрирующий использование некоторых основных функций. Что выделяет тур по языку Go в ряду типичных пособий, так это действующие примеры. Примеры можно выполнять непосредственно в браузере. При желании можно даже внести в них изменения и выполнить.

Поэкспериментировать с простыми Go-приложениями можно на странице https://play.golang.org. Эта площадка для экспериментов позволяет опробовать примеры из тура. Здесь можно проверить код и поделиться ссылкой на него. Кроме того, на площадке можно также опробовать примеры кода из книги.

1.4.1. Установка Go

Установка Go – довольно простая процедура. Вся необходимая информация о получении и установке Go приводится на странице http://golang.org/doc/install. Здесь перечисляются поддерживаемые операционные системы, аппаратное обеспечение и многое другое.

Для Microsoft Windows и Mac OS X имеются инсталляторы, которые позаботятся об установке. Этот процесс настолько же прост, как установка любой другой программы. Пользователи Homebrew на OS X могут установить Go командой `brew install go`.

Установка Go в Linux предусматривает более широкий диапазон вариантов. Установить Go можно с помощью встроенного диспетчера пакетов, такого как `apt-get` или `yum`. Но, как правило, они устанавливают не самую последнюю версию, притом, что новые версии работают быстрее и поддерживают новые функциональные возможности. Следуя инструкциям по установке Go, можно загрузить самую последнюю версию языка, разместить ее в системе и добавить пути к исполняемым файлам в переменную окружения `PATH`. В версиях Linux, поддерживающих систему управления пакетами Debian, таких как Ubuntu, установить последнюю версию Go можно с помощью `godeb`. Пояснения автора `godeb` о процедуре установки можно найти на странице http://blog.labix.org/2013/06/15/in-flight-deb-packages-of-go.

1.4.2. Работа с Git, Mercurial и другими системами управления версиями

Для работы с пакетами и внешними зависимостями, хранящимися в системах управления версиями, среда Go требует локальной установки этих систем. Go не дублирует инструментов управления конфигурацией программного обеспечения (Software Configuration Management, SCM), а просто распознает их и использует при установке.

Двумя доминирующими системами управления версиями, используемыми Go-разработчиками для работы с пакетами, являются Git и Mercurial (`hg`). Система управления версиями Git очень популярна, ею пользуются многие разработчики из Google, и создаваемые ими пакеты хранятся в GitHub. От вас требуется лишь установить Git, при этом среда Go не требует какой-то определенной версии. Подойдет любая из последних.

1.4.3. Знакомство с рабочей областью

Утилита `go` предполагает, что исходный код на языке Go хранится в рабочей области. *Рабочая область* – это иерархия, включающая каталоги `src`, `pkg` и `bin`, как показано в следующем листинге.

Листинг 1.15 ❖ Структура рабочей области

```
$GOPATH/   ← ❶ Базовый каталог или $GOPATH
        src/   ← Исходный код внешних зависимостей
            github.com/
                        Masterminds/
                                    cookoo/
                                    glide/
        bin/   ← Скомпилированные программы
            glide
        pkg/   ← Скомпилированные библиотеки
            darwin_amd64/
    github.com/
                        Masterminds/
                                            cookoo.a
```

Единственной переменной окружения, которую необходимо определить, является `$GOPATH` ❶. С ее помощью `go` определяет местоположение базового каталога рабочей области. Исходный код, включая ваш собственный и всех зависимостей, должен располагаться в каталоге `src`. Управляет этим каталогом разработчик, а инструменты Go помогают управлять кодом, хранящимся во внешних репозиториях. Двумя

другими каталогами практически всегда управляют инструменты Go. Если в каталоге проекта запустить команду `go install`, она произведет сборку выполняемых файлов и сохранит их в каталоге `bin` (как показано в листинге 1.15). В данном примере проект Glide будет скомпилирован в выполняемый файл `glide`. Для проекта Cookoo, однако, автономный выполняемый файл не будет создан. Этот проект просто содержит библиотеки, которыми смогут пользоваться другие Go-программы. Выполнение команды `go install` в этом проекте приведет к созданию в каталоге `pkg` архивного файла с расширением `.a`.

1.4.4. Работа с переменными среды

Выполняемому файлу `go` требуется только одна переменная окружения – `GOPATH`. Она должна хранить путь к каталогу рабочей области, куда будут импортироваться пакеты, сохраняться выполняемые файлы и промежуточные архивы. Следующий пример демонстрирует создание рабочей области в каталоге `go` в UNIX-подобной системе:

```
$ mkdir $HOME/go
$ export GOPATH=$HOME/go
```

Внутри каталога, на который указывает переменная среды `GOPATH`, программа `go` создаст каталог `bin` для размещения выполняемых файлов. Для большего удобства желательно добавить этот каталог в переменную среды `PATH`. Например, в UNIX-подобных системах это делается так:

```
$ export PATH=$PATH:$GOPATH/bin
```

Если вы пожелаете установить двоичные файлы в альтернативный каталог, определите путь к нему в переменной окружения `GOBIN`. Определять эту переменную необязательно.

1.5. Приложение Hello Go

В следующем листинге представлена очередная вариация стандартной программы «Hello World», которая выводит через веб-сервер фразу `Hello, my name is Inigo Montoya`.

Листинг 1.16 ❖ Вывод Hello World через веб-сервер: `inigo.go`

```
package main  ← ❶ Приложение использует пакет main

import (          ❷ Импорт необходимых пакетов
    "fmt"
    "net/http"
)
```

```
func hello(res http.ResponseWriter, req *http.Request) {
    fmt.Fprint(res, "Hello, my name is Inigo Montoya")
}
```

Обработка HTTP-запроса

```
func main() {
    http.HandleFunc("/", hello)
    http.ListenAndServe("localhost:4000", nil)
}
```

❷ Основная логика приложения

Это простое приложение состоит из трех частей. Оно начинается с объявления пакета ❶. В отличие от библиотек, краткие имена которых описывают их назначение, например `net` или `crypto`, данное приложение называется `main`. Для вывода строк и работы в режиме веб-сервера импортируются пакеты `fmt` и `http` ❷. Импорт пакетов обеспечивает их доступность в коде и в скомпилированном приложении.

Выполнение приложения начинается с вызова функции `main` ❸, не имеющей аргументов и не возвращающей значений. В первой ее строке вызывается функция `http.HandleFunc`, сообщающая веб-серверу, что при обращении к пути `/` должна вызываться функция `hello`. Функция `hello` реализует типичный интерфейс обработчиков. Она получает объекты с HTTP-запросом и HTTP-ответом. Затем следует вызов метода `http.ListenAndServe`, который запускает веб-сервер, принимающий запросы на порту 4000 домена `localhost`.

Запустить это приложение можно двумя способами. В следующем листинге используется команда `go run`, компилирующая приложение в каталог `temp` и запускающая его.

Листинг 1.17 ❖ Запуск на выполнение файла `inigo.go`

```
$ go run inigo.go
```

Временный файл автоматически удаляется после завершения приложения. Это удобно при разработке постоянно тестируемых новых версий приложений.

После запуска приложения можно открыть веб-браузер и перейти по адресу: http://localhost:4000, чтобы посмотреть ответ, как показано на рис. 1.7.

Приложение можно также собрать и запустить другим способом, как показано в следующем листинге.

Листинг 1.18 ❖ Сборка `inigo.go`

```
$ go build inigo.go
$ ./inigo
```

Рис. 1.7 ❖ *Просмотр вывода «Hello, my name is Inigo Montoya» в веб-браузере*

Здесь сначала выполняется сборка приложения. Если вызвать команду `go build` без имен файлов, она соберет исходный код в текущем каталоге. Если указать одно или несколько имен файлов, она скомпилирует только указанные файлы. Затем собранное приложение можно запустить.

1.6. Итоги

Язык Go предназначен для работы на современном аппаратном обеспечении и разработки современных приложений. Он использует последние технологические достижения и инструменты, упрощающие процесс разработки. В этой главе мы рассмотрели основные достоинства языка программирования Go и познакомились:

- с философией языка Go, заключающейся в простоте и расширяемости, позволившей создать удобный язык и окружающую его экосистему;
- с особенностями языка Go, помогающими использовать преимущества современного оборудования, такими как сопрограммы, обеспечивающие параллельное выполнение;
- с сопровождающим язык Go набором инструментов, включающих средства тестирования, управления пакетами, форматирования и документирования;
- с характеристиками Go в сравнении с такими языками, как C, Java, JavaScript, Python, PHP и другими, а также узнали, для каких задач он подходит лучше всего;
- с порядком установки и использования Go.

Глава 2 начинается с рассмотрения основ, необходимых для создания самых разных приложений, от консольных утилит до веб-служб. Как приложения должны обрабатывать команды, аргументы и флаги? Как правильно завершать работу веб-служб? Ответы на подобные вопросы закладывают основу для создания полноценных приложений.

Далее в книге рассматриваются практические аспекты создания и выполнения приложений, написанных на языке Go. Главы будут дополнять друг друга, пока не будет достигнут кульминационный момент, когда, объединив все рассмотренные приемы, мы создадим законченное приложение.