

# Go Рецепты программирования

Второе издание

Более 85 рецептов для создания модульных, удобочитаемых и тестируемых приложений Golang в различных областях



**Packt>**

[www.packt.com](http://www.packt.com)

Аарон Торрес

## **Go. Рецепты программирования** ***Второе издание***

Более 85 рецептов для создания модульных, удобочитаемых и тестируемых приложений Golang в различных областях.

Аарон Торрес



**БИРМИНГЕМ — МУМБАИ**

# **Go. Рецепты программирования Вторая редакция**

Copyright © 2019 Packt Publishing

Первая редакция: Июнь 2017

Второе издание: Июль 2019

ISBN 978-1-78980-098-2

[www.packtpub.com](http://www.packtpub.com)

*Моей жене Кейли и моим дочерям Хейзел, Олеандр и Арании. Спасибо  
за ваше терпение, любовь и поддержку. Эта книга была бы  
невозможна без вас.*

# Contributors

## Об авторе

**Аарон Торрес** получил степень магистра компьютерных наук в Горно-технологическом институте Нью-Мексико. Он работал над распределенными системами в области высокопроизводительных вычислений, а также над крупномасштабными веб-приложениями и микросервисами. В настоящее время он возглавляет команду разработчиков Go, которая совершенствует и фокусируется на лучших практиках Go с упором на непрерывную доставку и автоматическое тестирование.

Аарон опубликовал ряд статей и имеет несколько патентов в области хранения и ввода/вывода. Он страстно любит делиться своими знаниями и идеями с другими. Он также является большим поклонником языка Go и программного обеспечения с открытым исходным кодом для серверных систем и разработки.

## О рецензенте

**Эдуард Бондаренко** — разработчик программного обеспечения, проживающий в Киеве, Украина. Он давно начал программировать на BASIC на ZX Spectrum. Позже он работал в сфере веб-разработки. Он использует Ruby on Rails более 8 лет. Долгое время используя Ruby, он открыл для себя Clojure в начале 2009 года, и ему понравилась простота языка. Помимо Ruby и Clojure, он интересуется разработкой Go и ReasonML.

*Я хочу поблагодарить мою замечательную жену, детей и родителей за всю любовь, поддержку и помощь, которые они мне оказывают.*

# Table of Contents

## Оглавление

- Contributors
  - Об авторе
  - О рецензенте
- Предисловие
  - Для кого эта книга
  - Что охватывает эта книга
  - Чтобы получить максимальную отдачу от этой книги
    - Загрузите файлы примеров кода
    - Код в действии
    - Используемые соглашения
  - Разделы
    - Подготовка
    - Как это сделать...
    - Как это работает...
  - Как связаться
    - Отзывы
- 1. Ввод-вывод и файловые системы
  - Технические требования
  - Использование общих интерфейсов ввода/вывода
    - Как это сделать...
    - Как это работает...
  - Использование пакетов bytes и strings
    - Как это сделать...
    - Как это работает...
  - Работа с каталогами и файлами
    - Как это сделать...
    - Как это работает...

- Работа с форматом CSV
  - Как это сделать...
  - Как это работает...
- Работа с временными файлами
  - Как это сделать...
  - Как это работает...
- Работа с text/template и html/template
  - Как это сделать...
  - Как это работает...
- 2. Инструменты командной строки
  - Технические требования
  - Использование флагов командной строки
    - Как это сделать...
    - Как это работает...
  - Использование аргументов командной строки
    - Как это сделать...
    - Как это работает...
  - Чтение и установка переменных среды
    - Как это сделать...
    - Как это работает...
  - Конфигурация с использованием TOML, YAML и JSON
    - Как это сделать...
    - Как это работает...
  - Работа с каналами Unix
    - Как это сделать...
    - Как это работает...
  - Перехват и обработка сигналов
    - Как это сделать...
    - Как это работает...
  - Приложение для раскрашивания ANSI
    - Как это сделать...
    - Как это работает...
- 3. Преобразование данных и композиция

- Технические требования
- Преобразование типов данных и приведение интерфейсов
  - Как это сделать...
  - Как это работает...
- Работа с числовыми типами данных с использованием `math` и `math/big`
  - Как это сделать...
  - Как это работает...
- Преобразование валюты и рассмотрение `float64`
  - Как это сделать...
  - Как это работает...
- Использование указателей и `SQL NullTypes` для кодирования и декодирования
  - Как это сделать...
  - Как это работает...
- Кодирование и декодирование данных `Go`
  - Как это сделать...
  - Как это работает...
- Структурные теги и базовая рефлексия в `Go`
  - Как это сделать...
  - Как это работает...
- Реализация коллекций через замыкания
  - Как это сделать...
  - Как это работает...
- 4. Обработка ошибок в `Go`
  - Технические требования
  - Обработка ошибок и интерфейс ошибок
    - Как это сделать...
    - Как это работает...
  - Использование пакета `pkg/errors` и перенос ошибок
    - Как это сделать...
    - Как это работает...



- Использование пакета журнала и понимание того, когда следует регистрировать ошибки
  - Как это сделать...
  - Как это работает...
- Структурированное ведение журналов с помощью пакетов arch и logrus
  - Как это сделать...
  - Как это работает...
- Ведение журнала с пакетом context
  - Как это сделать...
  - Как это работает...
- Использование глобальных переменных уровня пакета
  - Как это сделать...
  - Как это работает...
- Отлов паники для долго выполняющихся процессов
  - Как это сделать...
  - Как это работает...
- 5. Сетевое программирование
  - Технические требования
  - Написание эхо-сервера и клиента TCP/IP
    - Как это сделать...
    - Как это работает...
  - Написание UDP-сервера и клиента
    - Как это сделать...
    - Как это работает...
  - Работа с разрешением доменного имени
    - Как это сделать...
    - Как это работает...
  - Работа с веб-сокетами
    - Как это сделать...
    - Как это работает...
  - Работа с net/grpc для вызова удаленных методов
    - Как это сделать...

- Как это работает...
- Использование net/mail для разбора писем
  - Как это сделать...
  - Как это работает...
- 6. Все о базах данных и хранилищах
  - Использование пакета database/sql с MySQL
    - Подготовка
    - Как это сделать...
    - Как это работает...
  - Выполнение интерфейса транзакции базы данных
    - Подготовка
    - Как это сделать...
    - Как это работает...
  - Пул подключений, ограничение скорости и таймауты для SQL
    - Подготовка
    - Как это сделать...
    - Как это работает...
  - Работа с Redis
    - Подготовка
    - Как это сделать...
    - Как это работает...
  - Использование NoSQL с MongoDB
    - Подготовка
    - Как это сделать...
    - Как это работает...
  - Создание интерфейсов хранения для переносимости данных
    - Подготовка
    - Как это сделать...
    - Как это работает...
- 7. Веб-клиенты и API
  - Технические требования
  - Инициализация, хранение и передача структур http.Client

- Как это сделать...
  - Как это работает...
- Написание клиента для REST API
  - Как это сделать...
  - Как это работает...
- Выполнение параллельных и асинхронных клиентских запросов
  - Как это сделать...
  - Как это работает...
- Использование клиентов OAuth2
  - Подготовка
  - Как это сделать...
  - Как это работает...
- Реализация интерфейса хранения токенов OAuth2
  - Подготовка
  - Как это сделать...
  - Как это работает...
- Обертывание клиента дополнительным функционалом и функциональной композицией
  - Как это сделать...
  - Как это работает...
- Понимание клиентов GRPC
  - Подготовка
  - Как это сделать...
  - Как это работает...
- Использование twitchtv/twirp для RPC
  - Подготовка
  - Как это сделать...
  - Как это работает...
- 8. Микросервисы для приложений в Go
  - Технические требования
  - Работа с веб-обработчиками, запросами и экземплярами `ResponseWriter`

- Как это сделать...
  - Как это работает...
- Использование структур и замыканий для обработчиков состояния
  - Как это сделать...
  - Как это работает...
- Проверка входных данных для структур Go и пользовательских входных данных
  - Как это сделать...
  - Как это работает...
- Рендеринг и согласование контента
  - Как это сделать...
  - Как это работает...
- Внедрение и использование промежуточного ПО
  - Как это сделать...
  - Как это работает...
- Создание обратного прокси-приложения
  - Как это сделать...
  - Как это работает...
- Экспорт GRPC как JSON API
  - Подготовка
  - Как это сделать...
  - Как это работает...
- 9. Тестирование Go-кода
  - Технические требования
  - Мокинг с использованием стандартной библиотеки
    - Как это сделать...
    - Как это работает...
  - Использование пакета Mockgen для имитации интерфейсов
    - Подготовка
    - Как это сделать...
    - Как это работает...
  - Использование табличных тестов для улучшения покрытия

- Как это сделать...
- Как это работает...
- Использование сторонних инструментов тестирования
  - Подготовка
  - Как это сделать...
  - Как это работает...
- Поведенческое тестирование с использованием Go
  - Подготовка
  - Как это сделать...
  - Как это работает...
- 10. Параллелизм и конкурентность
  - Технические требования
  - Использование каналов и оператора select
    - Как это сделать...
    - Как это работает...
  - Выполнение асинхронных операций с sync.WaitGroup
    - Как это сделать...
    - Как это работает...
  - Использование атомарных операций и мьютекса
    - Как это сделать...
    - Как это работает...
  - Использование пакета context
    - Как это сделать...
    - Как это работает...
  - Выполнение управления состоянием для каналов
    - Как это сделать...
    - Как это работает...
  - Использование шаблона проектирования пула рабочих процессов
    - Как это сделать...
    - Как это работает...
  - Использование воркеров для создания пайплайнов
    - Как это сделать...

- Как это работает...
- 11. Распределенные системы
  - Технические требования
  - Использование службы обнаружения с Consul
    - Как это сделать...
    - Как это работает...
  - Реализация базового консенсуса с использованием Raft
    - Как это сделать...
    - Как это работает...
  - Использование контейнеризации с Docker
    - Подготовка
    - Как это сделать...
    - Как это работает...
  - Стратегии оркестрации и развертывания
    - Как это сделать...
    - Как это работает...
  - Приложения для мониторинга
    - Как это сделать...
    - Как это работает...
  - Сбор метрик
    - Подготовка
    - Как это сделать...
    - Как это работает...
- 12. Реактивное программирование и потоки данных
  - Технические требования
  - Использование Goflow для программирования потока данных
    - Как это сделать...
    - Как это работает...
  - Использование Kafka с Sarama
    - Подготовка
    - Как это сделать...
    - Как это работает...
  - Использование асинхронных продюсеров с Kafka

- Подготовка
  - Как это сделать...
  - Как это работает...
- Подключение Kafka к Goflow
  - Подготовка
  - Как это сделать...
  - Как это работает...
- Пишем сервер GraphQL на Go
  - Как это сделать...
  - Как это работает...
- 13. Бессерверное программирование
  - Go программирование на Lambda с Apex
    - Подготовка
    - Как это сделать...
    - Как это работает...
  - Бессерверное ведение журналов и метрик Apex
    - Подготовка
    - Как это сделать...
    - Как это работает...
  - Google App Engine с Go
    - Подготовка
    - Как это сделать...
    - Как это работает...
  - Работа с Firebase с помощью [firebase.google.com/go](https://firebase.google.com/go)
    - Подготовка
    - Как это сделать...
    - Как это работает...
- 14. Улучшения производительности, советы и рекомендации
  - Технические требования
  - Использование инструмента `pprof`
    - Как это сделать...
    - Как это работает...
  - Бенчмаркинг и поиск узких мест

- Как это сделать...
- Как это работает...
- Распределение памяти и управление кучей
  - Как это сделать...
  - Как это работает...
- Использование fasthttprouter и fasthttp
  - Как это сделать...
  - Как это работает...



# Предисловие

Спасибо, что выбрали эту книгу! Я надеюсь, что это будет удобным справочником для разработчиков, чтобы быстро найти шаблоны разработки Go. Он предназначен для использования в качестве дополнения к другим ресурсам и справочника, который, как мы надеемся, будет полезен спустя долгое время после его прочтения. Каждый рецепт в этой книге включает работающий, простой и проверенный код, который можно использовать в качестве справочного материала или основы для ваших собственных приложений. Книга охватывает широкий спектр тем, от базовых до продвинутых.

## Для кого эта книга

Эта книга предназначена для веб-разработчиков, программистов и корпоративных разработчиков. Предполагается базовое знание языка Go. Опыт разработки серверных приложений не обязателен, но может помочь понять мотивацию некоторых рецептов.

Эта книга служит хорошим справочником для разработчиков Go, которые уже имеют опыт, но нуждаются в быстром напоминании, примере или справке. Благодаря репозиторию с открытым исходным кодом должна быть возможность быстро поделиться этими примерами с командой. Если вы ищете быстрые решения распространенных и не очень распространенных проблем в программировании на Go, эта книга для вас.

## Что охватывает эта книга

**Глава 1**, *Ввод-вывод и файловые системы*, охватывает распространенные интерфейсы ввода-вывода Go и исследует работу с файловыми системами. Сюда входят временные файлы, шаблоны и файлы CSV.

**Глава 2**, *Инструменты командной строки*, рассматривают ввод данных пользователем через командную строку и исследуют

обработку распространенных типов данных, таких как TOML, YAML и JSON.

[Глава 3](#), *Преобразование и композиция данных*, демонстрирует методы приведения и преобразования между интерфейсами Go и типами данных. Она также демонстрирует стратегии кодирования и некоторые шаблоны функционального проектирования для Go.

[Глава 4](#), *Обработка ошибок в Go*, демонстрирует стратегии обработки ошибок в Go. Она исследует, как передавать ошибки, обрабатывать их и регистрировать.

[Глава 5](#), *Сетевое программирование*, демонстрирует использование различных сетевых примитивов, таких как UDP и TCP/IP. Она также исследует **систему доменных имен (DNS)**, работу с необработанными сообщениями электронной почты и основы **вызова удаленных процедур (RPC)**.

[Глава 6](#), *Все о базах данных и хранилище*, посвящено различным библиотекам хранения для доступа к системам хранения данных, таким как MySQL. Она также демонстрирует использование интерфейсов для отделения вашей библиотеки от логики вашего приложения.

[Глава 7](#), *Веб-клиенты и API*, реализует клиентские интерфейсы Go HTTP, клиенты REST, клиенты OAuth2, украшающие и расширяющие клиенты, а также gRPC.

[Глава 8](#), *Микросервисы для приложений в Go*, исследуют веб-обработчики, передачу состояния обработчику, проверку пользовательского ввода и промежуточное ПО.

[Глава 9](#), *Тестирование кода Go*, фокусируется на насмешках, тестовом покрытии, фаззинге, поведенческом тестировании и полезных инструментах тестирования.

[Глава 10](#), *Параллелизм и конкурентность*, предоставляет справочник по каналам и асинхронным операциям, атомарным значениям, объектам контекста Go и управлению состоянием канала.

[Глава 11](#), *Распределенные системы*, реализуют обнаружение сервисов, контейнеризацию Docker, метрики и мониторинг, а также оркестрацию. В основном это касается развертывания и производства приложений Go.

**Глава 12**, *Реактивное программирование и потоки данных*, исследует реактивные приложения и приложения потока данных, Kafka и распределенные очереди сообщений, а также серверы GraphQL.

**Глава 13**, *Бессерверное программирование*, занимается развертыванием приложений Go без обслуживания сервера. Сюда входит использование Google App Engine, Firebase, Lambda и ведение журнала в бессерверной среде.

**Глава 14**, *Улучшения производительности, советы и рекомендации*, посвящена сравнительному тестированию, выявлению узких мест, оптимизации и повышению производительности HTTP для приложений Go.

## Чтобы получить максимальную отдачу от этой книги

Для использования этой книги вам потребуется следующее:

- Среда программирования Unix.
- Последняя версия серии Go 1.x.
- Интернет-соединение.
- Разрешение на установку дополнительных пакетов, как описано в каждой главе.
- Предварительные условия и другие требования к установке для каждого рецепта указаны в разделе «*Технические требования*» соответствующих глав.

## Загрузите файлы примеров кода

Вы можете загрузить примеры файлов кода для этой книги из своей учетной записи на сайте [www.packtpub.com](http://www.packtpub.com). Если вы приобрели эту книгу в другом месте, вы можете посетить сайт [www.packtpub.com/support](http://www.packtpub.com/support) и зарегистрироваться, чтобы файлы были отправлены вам по электронной почте.

Вы можете скачать файлы кода, выполнив следующие действия:

- Войдите или зарегистрируйтесь на [www.packtpub.com](http://www.packtpub.com).
- Выберите вкладку **SUPPORT**.
- Нажмите **Code Downloads & Errata**.

- Введите название книги в поле **Search** и следуйте инструкциям на экране.

После загрузки файла убедитесь, что вы распаковали или извлекли папку, используя последнюю версию:

- WinRAR/7-Zip для Windows
- Zipeg/iZip/UnRarX для Mac
- 7-Zip/PeaZip для Linux

Пакет кода для книги также размещен на GitHub по адресу <https://github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition>. У нас также есть другие пакеты кода из нашего богатого каталога книг и видео, доступных по адресу <https://github.com/PacktPublishing/>. Проверь их!

## Код в действии

Перейдите по следующей ссылке, чтобы просмотреть видеоролики о выполнении кода: <http://bit.ly/2J2uqQ3>

## Используемые соглашения

В этой книге используется ряд текстовых соглашений.

**CodeInText**: указывает кодовые слова в тексте, имена таблиц базы данных, имена папок, имена файлов, расширения файлов, пути, фиктивные URL-адреса, пользовательский ввод и дескрипторы Twitter. Вот пример: «Библиотека **bytes** предоставляет ряд удобных функций при работе с данными».

Блок кода устанавливается следующим образом:

```

        b, err := ioutil.ReadAll(r)
        if err != nil {
            return "", err
        }
        return string(b), nil
    }

```

Когда мы хотим привлечь ваше внимание к определенной части блока кода, соответствующие строки или элементы выделяются жирным шрифтом:

```
package bytestrings

import (
    "bytes"
    "io"
    "io/ioutil"
)
```

Любой ввод или вывод командной строки записывается следующим образом:

```
$ go mod init github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/Chapter01/interfaces
```

**Жирный:** Обозначает новый термин, важное слово или слова, которые вы видите на экране. Например, слова в меню или диалоговых окнах отображаются в тексте следующим образом. Вот пример: «Выберите Информация о системе в панели администратора».



*Предупреждения или важные примечания выглядят следующим образом.*



*Советы и рекомендации выглядят следующим образом.*

## Разделы

В этой книге вы найдете несколько часто встречающихся заголовков (*Подготовка*, *Как это сделать...* и *Как это работает...*).

Чтобы дать четкие инструкции по завершению рецепта, используйте эти разделы следующим образом:

### Подготовка

В этом разделе рассказывается, чего ожидать от рецепта, и описывается, как установить любое программное обеспечение или какие-либо предварительные настройки, необходимые для рецепта.

## Как это сделать...

Этот раздел содержит шаги, необходимые для выполнения рецепта.

## Как это работает...

Этот раздел обычно состоит из подробного объяснения того, что произошло в предыдущем разделе.

## Как связаться

Отзывы наших читателей всегда приветствуются.

**Общий отзыв:** напишите на почту [feedback@packtpub.com](mailto:feedback@packtpub.com) и укажите название книги в теме сообщения. Если у вас есть вопросы по какому-либо аспекту этой книги, пожалуйста, напишите нам по адресу [questions@packtpub.com](mailto:questions@packtpub.com).

**Исправления:** хотя мы приложили все усилия, чтобы обеспечить точность нашего контента, ошибки случаются. Если вы нашли ошибку в этой книге, мы будем признательны, если вы сообщите нам об этом. Посетите веб-сайт [www.packtpub.com/submit-errata](http://www.packtpub.com/submit-errata), выберите свою книгу, нажмите на ссылку **Errata Submission Form** и введите данные.

**Если вы заинтересованы в том, чтобы стать автором:** Если есть тема, в которой у вас есть опыт, и вы хотите написать или внести свой вклад в книгу, посетите [authors.packtpub.com](http://authors.packtpub.com).

## Отзывы

Пожалуйста, оставьте отзыв. После того как вы прочитали и воспользовались этой книгой, почему бы не оставить отзыв на сайте, где вы ее приобрели? После этого потенциальные читатели смогут увидеть ваше непредвзятое мнение и использовать его для принятия решения о покупке, мы в Packt можем понять, что вы думаете о наших продуктах, а наши авторы смогут увидеть ваши отзывы об их книге. Спасибо!

Для получения дополнительной информации о Packt посетите сайт [packtpub.com](http://packtpub.com).

# 1. Ввод-вывод и файловые системы

Go обеспечивает отличную поддержку как базового, так и сложного ввода-вывода. Рецепты в этой главе исследуют общие интерфейсы Go, которые используются для работы с вводом-выводом, и показывают, как их использовать. Стандартная библиотека Go часто использует эти интерфейсы, и они будут использоваться в рецептах на протяжении всей книги.

Вы научитесь работать с данными в памяти и в виде потоков. Вы увидите примеры работы с файлами, каталогами и форматом CSV. В рецепте временных файлов рассматривается механизм работы с файлами без накладных расходов, связанных с конфликтами имен и т. д. Наконец, мы рассмотрим стандартные шаблоны Go как для простого текста, так и для HTML.

Эти рецепты должны заложить основу для использования интерфейсов для представления и изменения данных и должны помочь вам мыслить о данных абстрактно и гибко.

В этой главе будут рассмотрены следующие рецепты:

- Использование общих интерфейсов ввода/вывода
- Использование пакетов `bytes` и `strings`
- Работа с каталогами и файлами
- Работа с форматом CSV
- Работа с временными файлами
- Работа с `text/template` и `html/template`

## Технические требования

Чтобы продолжить выполнение всех рецептов в этой главе, настройте свою среду в соответствии со следующими шагами:

- Загрузите и установите Go 1.12.6 или более позднюю версию в своей операционной системе по адресу <https://golang.org/doc/install>.
- Откройте терминал или консольное приложение, создайте и перейдите в каталог проекта, например `~/projects/go-programming-cookbook`. Весь код будет запускаться и изменяться из этого каталога.
- Скопируйте последний код в `~/projects/go-programming-cookbook-original`, как показано в следующем коде. Рекомендуется работать с

этим каталогом, а не вводить примеры вручную:

```
$ git clone git@github.com:PacktPublishing/Go-Programming-
Cookbook-Second-Edition.git go-programming-cookbook-original
```

## Использование общих интерфейсов ввода/вывода

Язык Go предоставляет ряд интерфейсов ввода-вывода, которые используются во всей стандартной библиотеке. Лучше всего использовать эти интерфейсы везде, где это возможно, а не передавать структуры или другие типы напрямую. Два мощных интерфейса, которые мы рассмотрим в этом рецепте, — это интерфейсы `io.Reader` и `io.Writer`. Эти интерфейсы используются во всей стандартной библиотеке, и понимание того, как их использовать, сделает вас лучшим разработчиком Go.

Интерфейсы `Reader` и `Writer` выглядят следующим образом:

```
type Reader interface {
    Read(p []byte) (n int, err error)
}

type Writer interface {
    Write(p []byte) (n int, err error)
}
```

Go также позволяет легко комбинировать интерфейсы. Например, взгляните на следующий код:

```
type Seeker interface {
    Seek(offset int64, whence int) (int64, error)
}

type ReadSeeker interface {
    Reader
    Seeker
}
```

В этом рецепте также исследуется функция `io` под названием `Pipe()`, как показано в следующем коде:

```
func Pipe() (*PipeReader, *PipeWriter)
```

Оставшаяся часть этой книги будет использовать эти интерфейсы.



## Как это сделать...

Следующие шаги описывают, как написать и запустить ваше приложение:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter1/interfaces`.
- Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter1/interfaces
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter1/interfaces
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter1/interfaces` или используйте это как упражнение для написания собственного кода!
- Создайте файл `interfaces.go` со следующим содержимым:

```
package interfaces

import (
    "fmt"
    "io"
    "os"
)

// Copy copies data from in to out first directly,
// then using a buffer. It also writes to stdout
func Copy(in io.ReadSeeker, out io.Writer) error {
    // we write to out, but also Stdout
    w := io.MultiWriter(out, os.Stdout)

    // a standard copy, this can be dangerous if
there's a
    // lot of data in in
    if _, err := io.Copy(w, in); err != nil {
        return err
    }

    in.Seek(0, 0)
```

```

        // buffered write using 64 byte chunks
        buf := make([]byte, 64)
        if _, err := io.CopyBuffer(w, in, buf); err !=
nil {
            return err
        }

        // lets print a new line
        fmt.Println()

        return nil
    }
}

```

- Создайте файл с именем `pipe.go` со следующим содержимым:

```

package interfaces

import (
    "io"
    "os"
)

// PipeExample helps give some more examples of using
io
//interfaces
func PipeExample() error {
    // the pipe reader and pipe writer implement
    // io.Reader and io.Writer
    r, w := io.Pipe()

    // this needs to be run in a separate go
routine
    // as it will block waiting for the reader
    // close at the end for cleanup
    go func() {
        // for now we'll write something basic,
        // this could also be used to encode json
        // base64 encode, etc.
        w.Write([]byte("test\n"))
        w.Close()
    }()

    if _, err := io.Copy(os.Stdout, r); err != nil
{
        return err
    }
}

```

```
        return nil
    }
}
```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл `main.go` со следующим содержимым:

```
package main

import (
    "bytes"
    "fmt"

    "github.com/PacktPublishing/
Go-Programming-Cookbook-Second-Edition/
chapter1/bytestrings"
)

func main() {
    in := bytes.NewReader([]byte("example"))
    out := &bytes.Buffer{}
    fmt.Print("stdout on Copy = ")
    if err := interfaces.Copy(in, out); err != nil
{
        panic(err)
    }

    fmt.Println("out bytes buffer =", out.String())

    fmt.Print("stdout on PipeExample = ")
    if err := interfaces.PipeExample(); err != nil
{
        panic(err)
    }
}
```

- Выполните `go run ..`
- Вы также можете запустить следующее:

```
$ go build
$ ./example
```

Вы должны увидеть следующий вывод:

```
$ go run .
stdout on Copy = exampleexample
out bytes buffer = exampleexample
stdout on PipeExample = test
```

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test` и убедитесь, что все тесты пройдены.

## Как это работает...

Функция `Copy()` копирует байты между интерфейсами и обрабатывает эти данные как поток. Представление о данных как о потоках имеет множество практических применений, особенно при работе с сетевым трафиком или файловыми системами. Функция `Copy()` также создает интерфейс `MultiWriter`, который объединяет два потока записи и дважды записывает в них с помощью `ReadSeeker`. Если бы вместо этого использовался интерфейс `Reader`, вместо того, чтобы видеть `example`, вы бы видели только пример, несмотря на двойное копирование в интерфейс `MultiWriter`. Вы также можете использовать буферизованную запись, если ваш поток не помещается в память.

Структуры `PipeReader` и `PipeWriter` реализуют интерфейсы `io.Reader` и `io.Writer`. Они связаны, создавая конвейер в памяти. Основная цель канала — чтение из потока при одновременной записи из того же потока в другой источник. По сути, он объединяет два потока в трубу.

Интерфейсы Go — это чистая абстракция для переноса данных, которые выполняют общие операции. Это становится очевидным при выполнении операций ввода-вывода, поэтому пакет `io` — отличный ресурс для изучения композиции интерфейса. Пакет `pipe` часто используется недостаточно, но обеспечивает большую гибкость с безопасностью потоков при связывании входных и выходных потоков.

## Использование пакетов `bytes` и `strings`

Пакеты `bytes` и `strings` имеют ряд полезных помощников для работы и преобразования данных из строковых типов в байтовые и наоборот. Они позволяют создавать буферы, которые работают с рядом общих интерфейсов ввода/вывода.

## Как это сделать...

Следующие шаги описывают, как написать и запустить ваше приложение:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter1/bytestrings`.

- Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter1/bytestrings
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter1/bytestrings
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter1/bytestrings` или используйте это как упражнение для написания собственного кода!
- Создайте файл с именем `buffer.go` со следующим содержимым:

```
package bytestrings

import (
    "bytes"
    "io"
    "io/ioutil"
)

// Buffer demonstrates some tricks for initializing
bytes //Buffers
// These buffers implement an io.Reader interface
func Buffer(rawString string) *bytes.Buffer {

    // we'll start with a string encoded into raw
bytes    rawBytes := []byte(rawString)

    // there are a number of ways to create a
buffer from // the raw bytes or from the original string
    var b = new(bytes.Buffer)
    b.Write(rawBytes)

    // alternatively
    b = bytes.NewBuffer(rawBytes)

    // and avoiding the initial byte array
altogether
```

```

        b = bytes.NewBufferString(rawString)

        return b
    }

    // ToString is an example of taking an io.Reader and
    consuming
    // it all, then returning a string
    func toString(r io.Reader) (string, error) {
        b, err := ioutil.ReadAll(r)
        if err != nil {
            return "", err
        }
        return string(b), nil
    }
}

```

- Создайте файл с именем `bytes.go` со следующим содержимым:

```

package bytestrings

import (
    "bufio"
    "bytes"
    "fmt"
)

// WorkWithBuffer will make use of the buffer created
by the
// Buffer function
func WorkWithBuffer() error {
    rawString := "it's easy to encode unicode into
a byte
                                array"

    b := Buffer(rawString)

    // we can quickly convert a buffer back into
bytes with
    // b.Bytes() or a string with b.String()
    fmt.Println(b.String())

    // because this is an io Reader we can make use
of
    // generic io reader functions such as
    s, err := toString(b)
    if err != nil {
        return err
    }
}

```

```

    }
    fmt.Println(s)

    // we can also take our bytes and create a
bytes reader    // these readers implement io.Reader,
io.ReaderAt,    // io.WriterTo, io.Seeker, io.ByteScanner, and
                // io.RuneScanner interfaces
                reader := bytes.NewReader([]byte(rawString))

allows          // we can also plug it into a scanner that

                // buffered reading and tokenization
                scanner := bufio.NewScanner(reader)
                scanner.Split(bufio.ScanWords)

                // iterate over all of the scan events
                for scanner.Scan() {
                    fmt.Print(scanner.Text())
                }

                return nil
    }

```

- Создайте файл с именем `string.go` со следующим содержимым:

```

package bytestrings

import (
    "fmt"
    "io"
    "os"
    "strings"
)

// SearchString shows a number of methods
// for searching a string
func SearchString() {
    s := "this is a test"

    // returns true because s contains
    // the word this
    fmt.Println(strings.Contains(s, "this"))

    // returns true because s contains the letter a
    // would also match if it contained b or c

```

```

    fmt.Println(strings.ContainsAny(s, "abc"))

    // returns true because s starts with this
    fmt.Println(strings.HasPrefix(s, "this"))

    // returns true because s ends with this
    fmt.Println(strings.HasSuffix(s, "test"))
}

// ModifyString modifies a string in a number of ways
func ModifyString() {
    s := "simple string"

    // prints [simple string]
    fmt.Println(strings.Split(s, " "))

    // prints "Simple String"
    fmt.Println(strings.Title(s))

    // prints "simple string"; all trailing and
    // leading white space is removed
    s = " simple string "
    fmt.Println(strings.TrimSpace(s))
}

// StringReader demonstrates how to create
// an io.Reader interface quickly with a string
func StringReader() {
    s := "simple stringn"
    r := strings.NewReader(s)

    // prints s on Stdout
    io.Copy(os.Stdout, r)
}

```

- Создайте новый каталог с именем **example** и перейдите к нему.
- Создайте файл **main.go** со следующим содержимым:

```

package main

import "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter1/bytestrings"

func main() {
    err := bytestrings.WorkWithBuffer()
}

```



```

    if err != nil {
        panic(err)
    }

    // each of these print to stdout
    bytestrings.SearchString()
    bytestrings.ModifyString()
    bytestrings.StringReader()
}

```

- Выполните `go run ..`
- Вы также можете запустить следующее:

```

$ go build
$ ./example

```

Вы должны увидеть следующий вывод:

```

$ go run .
it's easy to encode unicode into a byte array ??
it's easy to encode unicode into a byte array ??
it'seasytoencodeunicodeintoa bytearray??true
true
true
true
[simple string]
Simple String
simple string
simple string

```

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test` и убедитесь, что все тесты пройдены.

## Как это работает...

Библиотека `bytes` предоставляет ряд удобных функций при работе с данными. Например, буфер гораздо более гибок, чем массив байтов, при работе с библиотеками или методами потоковой обработки. После того как вы создали буфер, его можно использовать для удовлетворения требований интерфейса `io.Reader`, чтобы вы могли воспользоваться преимуществами функций `ioutil` для управления данными. Для потоковых приложений вы, вероятно, захотите использовать буфер и сканер. В таких случаях пригодится пакет `bufio`. Иногда использование массива или среза больше подходит для небольших наборов данных или когда на вашем компьютере много памяти.

Go обеспечивает большую гибкость в преобразовании данных между интерфейсами при использовании этих базовых типов — преобразование между строками и байтами относительно простое. При работе со строками пакет `strings` предоставляет ряд удобных функций для работы, поиска и управления строками. В некоторых случаях может подойти хорошее регулярное выражение, но в большинстве случаев достаточно пакетов `strings` и `strconv`. Пакет `strings` позволяет сделать строку похожей на заголовок, разбить ее на массив или обрезать пробелы. Он также предоставляет собственный интерфейс `Reader`, который можно использовать вместо типа чтения пакетов `bytes`.

## Работа с каталогами и файлами

Работа с каталогами и файлами может быть затруднена при переключении между платформами (например, Windows и Linux). Go обеспечивает кроссплатформенную поддержку для работы с файлами и каталогами в пакетах `os` и `ioutil`. Мы уже видели примеры `ioutil`, но теперь мы рассмотрим, как использовать их по-другому!!

### Как это сделать...

Следующие шаги описывают, как написать и запустить ваше приложение:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter1/filedirs`.
- Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-  
Cookbook-Second-Edition/chapter1/filedirs
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее содержимое:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-  
Second-Edition/chapter1/filedirs
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter1/filedirs` или используйте это как упражнение для написания собственного кода!
- Создайте файл `dirs.go` со следующим содержимым:

```
package filedirs
```

```

import (
    "errors"
    "io"
    "os"
)

// Operate manipulates files and directories
func Operate() error {
    // this 0755 is similar to what you'd see with
    Chown
    // on a command line this will create a
    director
    // /tmp/example, you may also use an absolute
    path
    // instead of a relative one
    if err := os.Mkdir("example_dir",
os.FileMode(0755));
    err != nil {
        return err
    }

    // go to the /tmp directory
    if err := os.Chdir("example_dir"); err != nil {
        return err
    }

    // f is a generic file object
    // it also implements multiple interfaces
    // and can be used as a reader or writer
    // if the correct bits are set when opening
    f, err := os.Create("test.txt")
    if err != nil {
        return err
    }

    // we write a known-length value to the file
    and
    // validate that it wrote correctly
    value := []byte("hellon")
    count, err := f.Write(value)
    if err != nil {
        return err
    }
    if count != len(value) {
        return errors.New("incorrect length
returned

```

```

        from write")
    }

    if err := f.Close(); err != nil {
        return err
    }

    // read the file
    f, err = os.Open("test.txt")
    if err != nil {
        return err
    }

    io.Copy(os.Stdout, f)

    if err := f.Close(); err != nil {
        return err
    }

    // go to the /tmp directory
    if err := os.Chdir(".."); err != nil {
        return err
    }

    // cleanup, os.RemoveAll can be dangerous if
    // point at the wrong directory, use user
    // and especially if you run as root
    if err := os.RemoveAll("example_dir"); err !=
nil {
        return err
    }

    return nil
}

```

- Создайте файл с именем `files.go` со следующим содержимым:

```

package filedirs

import (
    "bytes"
    "io"
    "os"
    "strings"
)

```

```

// Capitalizer opens a file, reads the contents,
// then writes those contents to a second file
func Capitalizer(f1 *os.File, f2 *os.File)
error {
    if _, err := f1.Seek(0, io.SeekStart); err !=
nil {
        return err
    }

    var tmp = new(bytes.Buffer)

    if _, err := io.Copy(tmp, f1); err != nil {
        return err
    }

    s := strings.ToUpper(tmp.String())

    if _, err := io.Copy(f2, strings.NewReader(s));
err !=
    nil {
        return err
    }
    return nil
}

// CapitalizerExample creates two files, writes to one
//then calls Capitalizer() on both
func CapitalizerExample() error {
    f1, err := os.Create("file1.txt")
    if err != nil {
        return err
    }

    if _, err := f1.Write([]byte(`this file
contains a
number of words and new lines`)); err != nil {
        return err
    }

    f2, err := os.Create("file2.txt")
    if err != nil {
        return err
    }

    if err := Capitalizer(f1, f2); err != nil {

```

```

        return err
    }

    if err := os.Remove("file1.txt"); err != nil {
        return err
    }

    if err := os.Remove("file2.txt"); err != nil {
        return err
    }

    return nil
}

```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл `main.go` со следующим содержимым:

```

package main

import "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter1/filedirs"

func main() {
    if err := filedirs.Operate(); err != nil {
        panic(err)
    }

    if err := filedirs.CapitalizerExample(); err !=
nil {
        panic(err)
    }
}

```

- Выполните `go run ..`
- Вы также можете запустить следующее:

```

$ go build
$ ./example

```

Вы должны увидеть следующий вывод:

```

$ go run .
hello

```

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test` и убедитесь, что все тесты

пройденны.

## Как это работает...

Если вы знакомы с файлами в Unix, библиотека Go `os` должна показаться вам очень знакомой. Вы можете выполнять практически все стандартные операции — `Stat` файл для сбора атрибутов, собирать файл с различными разрешениями, а также создавать и изменять каталоги и файлы. В этом рецепте мы проделали ряд манипуляций с директориями и файлами и потом подчистили за собой.

Работа с файловыми объектами очень похожа на работу с потоками в памяти. Файлы также напрямую предоставляют ряд удобных функций, таких как `Chown`, `Stat` и `Truncate`. Самый простой способ освоиться с файлами — использовать их. Во всех предыдущих рецептах мы должны тщательно убирать за нашими программами.

Работа с файлами — очень распространенная операция при создании серверных приложений. Файлы можно использовать для конфигурации, секретных ключей, в качестве временного хранилища и многого другого. Go оборачивает системные вызовы ОС с помощью пакета `os` и позволяет работать одним и тем же функциям независимо от того, используете ли вы Windows или Unix.

Как только ваш файл открыт и сохранен в структуре `File`, его можно легко передать в ряд интерфейсов (мы обсуждали эти интерфейсы ранее). Все предыдущие примеры могут использовать структуры `os.File` напрямую вместо буферов и потоков данных в памяти, чтобы работать с данными, хранящимися на диске. Это может быть полезно для определенных методов, таких как одновременная запись всех журналов в `stderr` и в файл с помощью одного вызова записи.

## Работа с форматом CSV

CSV — это распространенный формат, который используется для манипулирования данными. Например, часто приходится импортировать или экспортировать файл CSV в Excel. Пакет Go `CSV` работает с интерфейсами данных, поэтому данные легко записывать в буфер, `stdout`, файл или сокет. Примеры в этом разделе покажут некоторые распространенные способы получения данных в формате CSV и из него.

## Как это сделать...

Эти шаги описывают, как написать и запустить ваше приложение:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter1/csvformat`.
- Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter1/csvformat
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее содержимое:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter1/csvformat
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter1/csvformat` или используйте это как упражнение для написания собственного кода!
- Создайте файл с именем `read_csv.go` со следующим содержимым:

```
package csvformat

import (
    "bytes"
    "encoding/csv"
    "fmt"
    "io"
    "strconv"
)

// Movie will hold our parsed CSV
type Movie struct {
    Title string
    Director string
    Year int
}

// ReadCSV gives shows some examples of processing CSV
// that is passed in as an io.Reader
func ReadCSV(b io.Reader) ([]Movie, error) {

    r := csv.NewReader(b)

    // These are some optional configuration
options
    r.Comma = ';'
}
```



```

        r.Comment = '-'

        var movies []Movie

        // grab and ignore the header for now
        // we may also want to use this for a
dictionary key or
        // some other form of lookup
        _, err := r.Read()
        if err != nil && err != io.EOF {
            return nil, err
        }

        // loop until it's all processed
        for {
            record, err := r.Read()
            if err == io.EOF {
                break
            } else if err != nil {
                return nil, err
            }

            year, err :=
strconv.ParseInt(record[2], 10,
64)
            if err != nil {
                return nil, err
            }

            m := Movie{record[0], record[1],
int(year)}

            movies = append(movies, m)
        }
        return movies, nil
    }
}

```

- Добавьте эту дополнительную функцию в `read_csv.go` следующим образом:

```

// AddMoviesFromText uses the CSV parser with a string
func AddMoviesFromText() error {
    // this is an example of us taking a string,
converting
    // it into a buffer, and reading it
    // with the csv package
    in := `

```

```

- first our headers
movie title;director;year released

- then some data
Guardians of the Galaxy Vol. 2;James Gunn;2017
Star Wars: Episode VIII;Rian Johnson;2017
`

```

```

b := bytes.NewBufferString(in)
m, err := ReadCSV(b)
if err != nil {
    return err
}
fmt.Printf("%#vn", m)
return nil

```

```

}

```

- Создайте файл с именем `write_csv.go` со следующим содержимым:

```

package csvformat

import (
    "bytes"
    "encoding/csv"
    "io"
    "os"
)

// A Book has an Author and Title
type Book struct {
    Author string
    Title string
}

// Books is a named type for an array of books
type Books []Book

// ToCSV takes a set of Books and writes to an
io.Writer
// it returns any errors
func (books *Books) ToCSV(w io.Writer) error {
    n := csv.NewWriter(w)
    err := n.Write([]string{"Author", "Title"})
    if err != nil {
        return err
    }
    for _, book := range *books {

```

```

        err := n.Write([]string{book.Author,
        book.Title})
        if err != nil {
            return err
        }
    }

    n.Flush()
    return n.Error()
}

```

- Добавьте эти дополнительные функции в `write_csv.go` следующим образом:

```

// WriteCSVOutput initializes a set of books
// and writes the to os.Stdout
func WriteCSVOutput() error {
    b := Books{
        Book{
            Author: "F Scott Fitzgerald",
            Title: "The Great Gatsby",
        },
        Book{
            Author: "J D Salinger",
            Title: "The Catcher in the
Rye",
        },
    }

    return b.ToCSV(os.Stdout)
}

// WriteCSVBuffer returns a buffer csv for
// a set of books
func WriteCSVBuffer() (*bytes.Buffer, error) {
    b := Books{
        Book{
            Author: "F Scott Fitzgerald",
            Title: "The Great Gatsby",
        },
        Book{
            Author: "J D Salinger",
            Title: "The Catcher in the
Rye",
        },
    }
}

```

```

        w := &bytes.Buffer{}
        err := b.ToCSV(w)
        return w, err
    }

```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл `main.go` со следующим содержимым:

```

package main

import (
    "fmt"

    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter1/csvformat"
)

func main() {
    if err := csvformat.AddMoviesFromText(); err !=
nil {
        panic(err)
    }

    if err := csvformat.WriteCSVOutput(); err !=
nil {
        panic(err)
    }

    buffer, err := csvformat.WriteCSVBuffer()
    if err != nil {
        panic(err)
    }

    fmt.Println("Buffer = ", buffer.String())
}

```

- Выполните `go run ..`
- Вы также можете запустить следующее:

```

$ go build
$ ./example

```

Вы должны увидеть следующий вывод:

```
$ go run .
[]csvformat.Movie{csvformat.Movie{Title:"Guardians of the
Galaxy Vol. 2", Director:"James Gunn", Year:2017},
csvformat.Movie{Title:"Star Wars: Episode VIII",
Director:"Rian
Johnson", Year:2017}}
Author,Title
F Scott Fitzgerald,The Great Gatsby
J D Salinger,The Catcher in the Rye
Buffer = Author,Title
F Scott Fitzgerald,The Great Gatsby
J D Salinger,The Catcher in the Rye
```

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test` и убедитесь, что все тесты пройдены.

## Как это работает...

Чтобы научиться читать формат CSV, мы сначала представляем наши данные в виде структуры. В Go очень полезно форматировать данные как структуру, поскольку это делает такие вещи, как маршалинг и кодирование, относительно простыми. В нашем примере чтения в качестве типа данных используются фильмы. Функция принимает интерфейс `io.Reader`, который содержит наши данные CSV в качестве входных данных. Это может быть файл или буфер. Затем мы используем эти данные для создания и заполнения структуры `Movie`, включая преобразование года в целое число. Мы также добавляем параметры для парсера CSV для использования `;` (точка с запятой) в качестве разделителя и `-` (дефис) в качестве строки комментария.

Далее мы исследуем ту же идею, но в обратном порядке. Романы представлены с названием и автором. Мы инициализируем массив романов, а затем записываем определенные романы в формате CSV в интерфейс `io.Writer`. Опять же, это может быть файл, `stdout` или буфер.

Пакет `CSV` — отличный пример того, почему потоки данных в Go можно рассматривать как реализацию общих интерфейсов. Легко изменить источник и место назначения наших данных с помощью небольших однострочных настроек, и мы можем легко манипулировать данными CSV, не используя чрезмерного объема памяти или времени. Например, можно было бы читать из потока данных по одной записи за раз и записывать в отдельный поток в модифицированном формате по одной записи за раз. Это не приведет к значительному использованию памяти или процессора.

Позже, когда мы будем изучать конвейеры данных и рабочие пулы, вы увидите, как эти идеи можно комбинировать и как обрабатывать эти потоки параллельно.

## Работа с временными файлами

Мы уже создали и использовали файлы для ряда примеров. Нам также приходилось вручную справляться с очисткой, конфликтами имен и многим другим. Временные файлы и каталоги — более быстрый и простой способ справиться с такими случаями.

### Как это сделать...

Следующие шаги описывают, как написать и запустить ваше приложение:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter1/tempfiles`.
- Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-  
Cookbook-Second-Edition/chapter1/tempfiles
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее содержимое:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-  
Second-Edition/chapter1/tempfiles
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter1/tempfiles` или используйте это как упражнение для написания собственного кода!
- Создайте файл с именем `temp_files.go` со следующим содержимым:

```
package tempfiles  
  
import (  
        "fmt"  
        "io/ioutil"  
        "os"  
)  
  
// WorkWithTemp will give some basic patterns for  
working // with temporary files and directories
```

```

func WorkWithTemp() error {
    // If you need a temporary place to store files
    with
    directory
    argument
    directory
    // the same name ie. template1-10.html a temp
    // is a good way to approach it, the first
    // being blank means it will use create the
    // in the location returned by
    // os.TempDir()
    t, err := ioutil.TempDir("", "tmp")
    if err != nil {
        return err
    }

    // This will delete everything inside the temp
    // when this function exits if you want to do
    // later, be sure to return the directory name
    // calling function
    defer os.RemoveAll(t)

    // the directory must exist to create the
    tempfile
    // created. t is an *os.File object.
    tf, err := ioutil.TempFile(t, "tmp")
    if err != nil {
        return err
    }

    fmt.Println(tf.Name())

    // normally we'd delete the temporary file
    here, but
    // because we're placing it in a temp
    directory, it
    // gets cleaned up by the earlier defer

    return nil
}

```

- Создайте новый каталог с именем **example** и перейдите к нему.
- Создайте файл **main.go** со следующим содержимым:

```

package main

import "github.com/PacktPublishing/
        Go-Programming-Cookbook-Second-Edition/
        chapter1/tempfiles"

func main() {
    if err := tempfiles.WorkWithTemp(); err != nil
{
        panic(err)
    }
}

```

- Выполните `go run ..`
- Вы также можете запустить следующее:

```

$ go build
$ ./example

```

Вы должны увидеть следующий вывод (с другим путем):

```

$ go run .
/var/folders/kd/ygq5L_0d1xq1lzk_c7htft900000gn/T
/tmp764135258/tmp588787953

```

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test` и убедитесь, что все тесты пройдены.

## Как это работает...

Создание временных файлов и каталогов можно выполнить с помощью пакета `ioutil`. Хотя вы по-прежнему должны удалять файлы самостоятельно, использование `RemoveAll` является соглашением, и оно делает это за вас всего с одной дополнительной строкой кода.

При написании тестов настоятельно рекомендуется использовать временные файлы. Это также полезно для таких вещей, как создание артефактов и многое другое. Пакет Go `ioutil` по умолчанию попытается учесть настройки ОС, но при необходимости позволит вам вернуться к другим каталогам.

## Работа с `text/template` и `html/template`

Go предоставляет богатую поддержку шаблонов. Легко вкладывать шаблоны, импортировать функции, представлять переменные, перебирать



данные и так далее. Если вам нужно что-то более сложное, чем средство записи CSV, отличным решением могут стать шаблоны.

Еще одно применение шаблонов — для веб-сайтов. Когда мы хотим отобразить данные на стороне сервера для клиента, шаблоны прекрасно подходят для этого. Поначалу шаблоны Go могут показаться запутанными. В этом разделе рассматривается работа с шаблонами, сбор шаблонов внутри каталога и работа с шаблонами HTML.

## Как это сделать...

Эти шаги описывают, как написать и запустить ваше приложение:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter1/templates`.
- Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-  
Cookbook-Second-Edition/chapter1/templates
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-  
Second-Edition/chapter1/templates
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter1/templates` или используйте это как упражнение, чтобы написать свой собственный код!
- Создайте файл `templates.go` со следующим содержимым:

```
package templates  
  
import (  
    "os"  
    "strings"  
    "text/template"  
)  
  
const sampleTemplate = `  
    This template demonstrates printing a {{  
.Variable |  
    printf "%#v" }}.  
  
    {{if .Condition}}
```

```

If condition is set, we'll print this
{{else}}
Otherwise, we'll print this instead
{{end}}

```

```

Next we'll iterate over an array of strings:
{{range $index, $item := .Items}}
{{$index}}: {{$item}}
{{end}}

```

```

We can also easily import other functions like
strings.Split
then immediately used the array created as a

```

result:

```

{{ range $index, $item := split .Words ","}}
{{$index}}: {{$item}}
{{end}}

```

another

```

Blocks are a way to embed templates into one
{{ block "block_example" .}}
No Block defined!
{{end}}

```

```

{/*
This is a way
to insert a multi-line comment
*/}

```

,

```

const secondTemplate = `
    {{ define "block_example" }}
    {{.OtherVariable}}
    {{end}}
`

```

- Добавьте функцию в конец `templates.go` следующим образом:

```

// RunTemplate initializes a template and demonstrates
a // variety of template helper functions
func RunTemplate() error {
    data := struct {
        Condition bool
        Variable string
        Items []string
    }

```

```

        Words string
        OtherVariable string
    }{
        Condition: true,
        Variable: "variable",
        Items: []string{"item1", "item2",
"item3"},
        Words:
"another_item1,another_item2,another_item3",
        OtherVariable: "I'm defined in a second
        template!",
    }

    funcmap := template.FuncMap{
        "split": strings.Split,
    }

    // these can also be chained
    t := template.New("example")
    t = t.Funcs(funcmap)

    // We could use Must instead to panic on error
    // template.Must(t.Parse(sampleTemplate))
    t, err := t.Parse(sampleTemplate)
    if err != nil {
        return err
    }

    // to demonstrate blocks we'll create another
    template
    a second

    // by cloning the first template, then parsing
    t2, err := t.Clone()
    if err != nil {
        return err
    }

    t2, err = t2.Parse(secondTemplate)
    if err != nil {
        return err
    }

    // write the template to stdout and populate it
    // with data
    err = t2.Execute(os.Stdout, &data)

```

```

        if err != nil {
            return err
        }

        return nil
    }

```

- Создайте файл с именем `template_files.go` со следующим содержимым:

```

package templates

import (
    "io/ioutil"
    "os"
    "path/filepath"
    "text/template"
)

//CreateTemplate will create a template file that
contains data
func CreateTemplate(path string, data string) error {
    return ioutil.WriteFile(path, []byte(data),
        os.FileMode(0755))
}

// InitTemplates sets up templates from a directory
func InitTemplates() error {
    tempdir, err := ioutil.TempDir("", "temp")
    if err != nil {
        return err
    }
    defer os.RemoveAll(tempdir)

    err = CreateTemplate(filepath.Join(tempdir,
"t1.tmpl"),
`Template 1! {{ .Var1 }}
{{ block "template2" .}} {{end}}
{{ block "template3" .}} {{end}}
`)
    if err != nil {
        return err
    }

    err = CreateTemplate(filepath.Join(tempdir,
"t2.tmpl"),

```

```

    `{{ define "template2"}}Template 2! {{ .Var2 }}
{{end}}
    `)
    if err != nil {
        return err
    }

    err = CreateTemplate(filepath.Join(tempdir,
"t3.tmpl"),
    `{{ define "template3"}}Template 3! {{ .Var3 }}
{{end}}
    `)
    if err != nil {
        return err
    }

    pattern := filepath.Join(tempdir, "*.tmpl")

    // Parse glob will combine all the files that
match
    // glob and combine them into a single template
    tpl, err := template.ParseGlob(pattern)
    if err != nil {
        return err
    }

    // Execute can also work with a map instead
    // of a struct
    tpl.Execute(os.Stdout, map[string]string{
        "Var1": "Var1!!",
        "Var2": "Var2!!",
        "Var3": "Var3!!",
    })

    return nil
}

```

- Создайте файл с именем `html_templates.go` со следующим содержимым:

```

package templates

import (
    "fmt"
    "html/template"
    "os"

```

```

    )

    // HTMLDifferences highlights some of the differences
    // between html/template and text/template
    func HTMLDifferences() error {
        t := template.New("html")
        t, err := t.Parse("<h1>Hello! {{.Name}}</h1>n")
        if err != nil {
            return err
        }

        // html/template auto-escapes unsafe operations
        like
        // javascript injection this is contextually
        aware and
        // will behave differently
        // depending on where a variable is rendered
        err = t.Execute(os.Stdout,
map[string]string{"Name": "
        <script>alert('Can you see me?')
        </script>"}))
        if err != nil {
            return err
        }

        // you can also manually call the escapers
        fmt.Println(template.JSEscaper(`example
        <example@example.com>`))
        fmt.Println(template.HTMLEscaper(`example
        <example@example.com>`))
        fmt.Println(template.URLQueryEscaper(`example
        <example@example.com>`))

        return nil
    }

```

- Создайте новый каталог с именем **example** и перейдите к нему.
- Создайте файл **main.go** со следующим содержимым:

```

package main

import "github.com/PacktPublishing/
        Go-Programming-Cookbook-Second-Edition/
        chapter1/templates"

func main() {

```

```
        if err := templates.RunTemplate(); err != nil {
            panic(err)
        }
        if err := templates.InitTemplates(); err != nil
    {
        panic(err)
    }
    if err := templates.HTMLDifferences(); err !=
nil {
        panic(err)
    }
}
```

- Выполните `go run ..`
- Вы также можете запустить следующее:

```
$ go build
$ ./example
```

Вы должны увидеть следующий вывод (с другим путем):

```

This template demonstrates printing a "variable".

If condition is set, we'll print this

Next we'll iterate over an array of strings:

    0: item1
    1: item2
    2: item3

We can also easily import other functions like strings.Split
then immediately used the array created as a result:

    0: another_item1
    1: another_item2
    2: another_item3

Blocks are a way to embed templates into one another

    I'm defined in a second template!

Template 1! Var1!!
Template 2! Var2!!
Template 3! Var3!!
<h1>Hello! &lt;script&gt;alert(&#39;Can you see me?&#39;)&lt;/script&gt;</h1>
example \x3Cexample@example.com\x3E
example &lt;example@example.com&gt;
example+%3Cexample%40example.com%3E

```

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите **go test** и убедитесь, что все тесты пройдены.

## Как это работает...

В Go есть два пакета шаблонов: **text/template** и **html/template**. Они разделяют функциональность и разнообразие функций. В общем, вы должны использовать **html/template** для рендеринга веб-сайтов и **text/template** для всего остального. Шаблоны представляют собой обычный



текст, но внутри фигурных скобок можно использовать переменные и функции.

Пакеты шаблонов также предоставляют удобные методы для работы с файлами. Пример, который мы использовали здесь, создает несколько шаблонов во временном каталоге, а затем считывает их все с помощью одной строки кода.

Пакет `html/template` является оболочкой пакета `text/template`. Все примеры шаблонов работают с пакетом `html/template` напрямую, без каких-либо модификаций и только с изменением оператора импорта. HTML-шаблоны обеспечивают дополнительное преимущество контекстно-зависимой безопасности; это предотвращает нарушения безопасности, такие как внедрение JavaScript.

Пакеты шаблонов предоставляют то, что вы ожидаете от современной библиотеки шаблонов. Легко комбинировать шаблоны, добавлять логику приложения и обеспечивать безопасность при передаче результатов в HTML и JavaScript.

## 2. Инструменты командной строки

Приложения командной строки — один из самых простых способов обработки пользовательского ввода и вывода. В этой главе основное внимание будет уделено взаимодействиям на основе командной строки, таким как аргументы командной строки, конфигурация и переменные среды. Мы закончим библиотекой для раскрашивания вывода текста в Unix и Bash для Windows.

С помощью рецептов, приведенных в этой главе, вы должны быть готовы справиться с ожидаемыми и неожиданными действиями пользователя. Рецепт *«Перехват и обработка сигналов»* является примером случаев, когда пользователи могут отправлять неожиданные сигналы вашему приложению, а рецепт каналов — это хорошая альтернатива принятию пользовательских входных данных по сравнению с флагами или аргументами командной строки.

Мы надеемся, что рецепт цвета ANSI предоставит пользователям несколько примеров очистки вывода. Например, при ведении журналов возможность раскрашивать текст в зависимости от его назначения иногда может сделать большие блоки текста значительно четче.

В этой главе мы рассмотрим следующие рецепты:

- Использование флагов командной строки
- Использование аргументов командной строки
- Чтение и установка переменных среды
- Конфигурация с использованием TOML, YAML и JSON
- Работа с каналами Unix
- Перехват и обработка сигналов
- Приложение для раскрашивания ANSI

### Технические требования

Чтобы продолжить выполнение всех рецептов в этой главе, настройте свою среду в соответствии со следующими шагами:

- Загрузите и установите Go 1.12.6 или более позднюю версию в своей операционной системе по адресу <https://golang.org/doc/install>.
- Откройте терминал или консольное приложение, создайте и перейдите в каталог проекта, например `~/projects/go-programming-cookbook`. Весь наш код будет запускаться и изменяться из этого каталога.
- Скопируйте последний код в `~/projects/go-programming-cookbook-original` и работайте с этим каталогом, а не вводите примеры вручную:

```
$ git clone git@github.com:PacktPublishing/Go-Programming-Cookbook-Second-Edition.git go-programming-cookbook-original
```

## Использование флагов командной строки

Пакет `flag` упрощает добавление аргументов флага командной строки в приложение Go. У него есть несколько недостатков — вы склонны дублировать большой объем кода, чтобы добавить сокращенные версии флагов, и они упорядочены в алфавитном порядке из подсказки справки. Существует ряд сторонних библиотек, которые пытаются устранить эти недостатки, но в этой главе основное внимание будет уделено версии стандартной библиотеки, а не этим библиотекам.

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter2/flags`.
- Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter2/flags
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter2/flags
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter2/flags` или используйте это как возможность написать свой собственный код!
- Создайте файл `flags.go` со следующим содержимым:

```
package main

import (
    "flag"
    "fmt"
)

// Config will be the holder for our flags
type Config struct {
    subject string
    isAwesome bool
    howAwesome int
    countTheWays CountTheWays
}

// Setup initializes a config from flags that
// are passed in
func (c *Config) Setup() {
    // you can set a flag directly like so:
    // var someVar = flag.String("flag_name",
"default_val",
    // "description")
    // but in practice putting it in a struct is
generally
    // better longhand
    flag.StringVar(&c.subject, "subject", "",
"subject is a
    string, it defaults to empty")
    // shorthand
    flag.StringVar(&c.subject, "s", "", "subject
is a string,
    it defaults to empty (shorthand)")
}
```

```

    flag.BoolVar(&c.isAwesome, "isawesome", false,
"is it
    awesome or what?")
    flag.IntVar(&c.howAwesome, "howawesome", 10,
"how awesome
    out of 10?")

    // custom variable type
    flag.Var(&c.countTheWays, "c", "comma separated
list of
    integers")
}

// GetMessage uses all of the internal
// config vars and returns a sentence
func (c *Config) GetMessage() string {
    msg := c.subject
    if c.isAwesome {
        msg += " is awesome"
    } else {
        msg += " is NOT awesome"
    }

    msg = fmt.Sprintf("%s with a certainty of %d
out of 10. Let
    me count the ways %s", msg, c.howAwesome,
    c.countTheWays.String())
    return msg
}

```

- Создайте файл `custom.go` со следующим содержимым:

```

package main

import (
    "fmt"
    "strconv"
    "strings"
)

// CountTheWays is a custom type that
// we'll read a flag into

```

```

type CountTheWays []int

func (c *CountTheWays) String() string {
    result := ""
    for _, v := range *c {
        if len(result) > 0 {
            result += " ... "
        }
        result += fmt.Sprint(v)
    }
    return result
}

// Set will be used by the flag package
func (c *CountTheWays) Set(value string) error {
    values := strings.Split(value, ",")

    for _, v := range values {
        i, err := strconv.Atoi(v)
        if err != nil {
            return err
        }
        *c = append(*c, i)
    }

    return nil
}

```

- Выполните следующую команду:

**\$ go mod tidy**

- Создайте файл с именем **main.go** со следующим содержимым:

```

package main

import (
    "flag"
    "fmt"
)

func main() {
    // initialize our setup
    c := Config{}

```

```

    c.Setup()

    // generally call this from main
    flag.Parse()

    fmt.Println(c.GetMessage())
}

```

- Выполните следующие команды в командной строке:

```

$ go build
$ ./flags -h

```

- Попробуйте эти и некоторые другие аргументы; вы должны увидеть следующий вывод:

```

$ go build
$ ./flags -h
Usage of ./flags:
-c value
comma separated list of integers
-howawesome int
how awesome out of 10? (default 10)
-isawesome
is it awesome or what? (default false)
-s string
subject is a string, it defaults to empty (shorthand)
-subject string
subject is a string, it defaults to empty
$ ./flags -s Go -isawesome -howawesome 10 -c 1,2,3
Go is awesome with a certainty of 10 out of 10. Let me
count
the ways 1 ... 2 ... 3

```

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test` и убедитесь, что все тесты пройдены.

## Как это работает...

Этот рецепт пытается продемонстрировать большинство распространенных способов использования пакета `flag`. Он показывает пользовательские типы переменных, множество

встроенных переменных, сокращенные флаги и запись всех флагов в общую структуру. Это первый рецепт, требующий функции `main`, так как основное использование флага (`flag.Parse()`) должно вызываться из `main`. В результате нормальный каталог примера опущен.

Пример использования этого приложения показывает, что вы получаете `-h` автоматически, чтобы получить список включенных флагов. Некоторые другие вещи, на которые следует обратить внимание, это логические флаги, которые вызываются без аргументов, и что порядок флагов не имеет значения.

Пакет `flag` — это быстрый способ структурировать входные данные для приложений командной строки и предоставить гибкие средства для указания предварительного пользовательского ввода для таких вещей, как настройка уровней ведения журнала или детализация приложения. В рецепте «*Использование аргументов командной строки*» мы рассмотрим наборы флагов и переключаемся между ними с помощью аргументов.

## Использование аргументов командной строки

Флаги из предыдущего рецепта являются типом аргумента командной строки. В этой главе будут рассмотрены другие варианты использования этих аргументов путем создания команды, поддерживающей вложенные подкоманды. Это продемонстрирует наборы флагов, а также использует позиционные аргументы, которые передаются в ваше приложение.

Как и в предыдущем рецепте, для этого требуется основная функция. Существует ряд сторонних пакетов, которые имеют дело со сложными вложенными аргументами и флагами, но мы рассмотрим, как это сделать, используя только стандартную библиотеку.

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-`



`cookbook/chapter2/cmdargs.`

- Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter2/cmdargs
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter2/cmdargs
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter2/cmdargs` или используйте это как возможность написать свой собственный код! или используйте это как возможность написать свой собственный код!
- Создайте файл `cmdargs.go` со следующим содержимым:

```
package main
import (
    "flag"
    "fmt"
    "os"
)
const version = "1.0.0"
const usage = `Usage:
%s [command]
Commands:
    Greet
    Version
`
const greetUsage = `Usage:
%s greet name [flag]
Positional Arguments:
    name
        the name to greet
Flags:
`
// MenuConf holds all the levels
// for a nested cmd line argument
```

```

type MenuConf struct {
    Goodbye bool
}
// SetupMenu initializes the base flags
func (m *MenuConf) SetupMenu() *flag.FlagSet {
    menu := flag.NewFlagSet("menu",
flag.ExitOnError)
    menu.Usage = func() {
        fmt.Printf(usage, os.Args[0])
        menu.PrintDefaults()
    }
    return menu
}
// GetSubMenu return a flag set for a submenu
func (m *MenuConf) GetSubMenu() *flag.FlagSet {
    submenu := flag.NewFlagSet("submenu",
flag.ExitOnError)
    submenu.BoolVar(&m.Goodbye, "goodbye", false,
"Say goodbye
instead of hello")
    submenu.Usage = func() {
        fmt.Printf(greetUsage, os.Args[0])
        submenu.PrintDefaults()
    }
    return submenu
}
// Greet will be invoked by the greet command
func (m *MenuConf) Greet(name string) {
    if m.Goodbye {
        fmt.Println("Goodbye " + name + "!")
    } else {
        fmt.Println("Hello " + name + "!")
    }
}
// Version prints the current version that is
// stored as a const
func (m *MenuConf) Version() {
    fmt.Println("Version: " + version)
}

```

- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "fmt"
    "os"
    "strings"
)

func main() {
    c := MenuConf{}
    menu := c.SetupMenu()

    if err := menu.Parse(os.Args[1:]); err != nil {
        fmt.Printf("Error parsing params %s, error:
%v", os.Args[1:], err)
        return
    }

    // we use arguments to switch between commands
    // flags are also an argument
    if len(os.Args) > 1 {
        // we don't care about case
        switch strings.ToLower(os.Args[1]) {
        case "version":
            c.Version()
        case "greet":
            f := c.GetSubMenu()
            if len(os.Args) < 3 {
                f.Usage()
                return
            }
            if len(os.Args) > 3 {
                if err := f.Parse(os.Args[3:]); err != nil
{
                    fmt.Fprintf(os.Stderr, "Error parsing
params %s, error: %v", os.Args[3:], err)
                    return
                }
            }
            c.Greet(os.Args[2])

        default:

```

```

        fmt.Println("Invalid command")
        menu.Usage()
        return
    }
} else {
    menu.Usage()
    return
}
}

```

- Выполните `go build`.
- Запустите следующие команды и попробуйте несколько других комбинаций аргументов:

```
$ ./cmdargs -h
Usage:
```

```
./cmdargs [command]
```

```
Commands:
Greet
Version
```

```
$ ./cmdargs version
Version: 1.0.0
```

```
$ ./cmdargs greet
Usage:
```

```
./cmdargs greet name [flag]
```

```
Positional Arguments:
name
the name to greet
```

```
Flags:
-goodbye
Say goodbye instead of hello
```

```
$ ./cmdargs greet reader
Hello reader!
```

```
$/cmdargs greet reader -goodbye  
Goodbye reader!
```

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test` и убедитесь, что все тесты пройдены.

## Как это работает...

Наборы флагов можно использовать для настройки независимых списков ожидаемых аргументов, строк использования и многого другого. Разработчик должен выполнить проверку ряда аргументов, анализируя правильное подмножество аргументов для команд и определяя строки использования. Это может быть подвержено ошибкам и требует много итераций, чтобы получить все правильно.

Пакет `flag` значительно упрощает синтаксический анализ аргументов и включает удобные методы для получения количества флагов, аргументов и многого другого. Этот рецепт демонстрирует основные способы создания сложного приложения командной строки с использованием аргументов, включая конфигурацию на уровне пакета, требуемые позиционные аргументы, использование многоуровневых команд и то, как при необходимости разделить эти вещи на несколько файлов или пакетов.

## Чтение и установка переменных среды

Переменные среды — это еще один способ передачи состояния в приложение помимо чтения данных из файла или явной передачи их через командную строку. Этот рецепт исследует некоторые очень простые способы получения и установки переменных среды, а затем поработает с очень полезной сторонней библиотекой `envconfig` (<https://github.com/kelseyhightower/envconfig>).

Мы создадим приложение, которое может читать файл `config` через JSON или через переменные среды. В следующем рецепте будут рассмотрены альтернативные форматы, включая TOML и YAML.

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter2/envvar`.
- Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter2/envvar
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter2/envvar
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter2/envvar` или используйте это как возможность написать свой собственный код!
- Создайте файл с именем `config.go` со следующим содержимым:

```
package envvar

import (
    "encoding/json"
    "os"

    "github.com/kelseychightower/envconfig"
    "github.com/pkg/errors"
)

// LoadConfig will load files optionally from the
json file
// stored at path, then will override those values
based on the
// envconfig struct tags. The envPrefix is how we
prefix our
// environment variables.
func LoadConfig(path, envPrefix string, config
interface{})
error {
    if path != "" {
```

```

        err := LoadFile(path, config)
        if err != nil {
            return errors.Wrap(err, "error loading
config from
        file")
        }
    }
    err := envconfig.Process(envPrefix, config)
    return errors.Wrap(err, "error loading config
from env")
}

// LoadFile unmarshalls a json file into a config
struct
func LoadFile(path string, config interface{})
error {
    configFile, err := os.Open(path)
    if err != nil {
        return errors.Wrap(err, "failed to read
config file")
    }
    defer configFile.Close()

    decoder := json.NewDecoder(configFile)
    if err = decoder.Decode(config); err != nil {
        return errors.Wrap(err, "failed to decode
config file")
    }
    return nil
}

```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "bytes"
    "fmt"
    "io/ioutil"
    "os"

    "github.com/PacktPublishing/

```

```

        Go-Programming-Cookbook-Second-Edition/
        chapter2/envvar"
    )

    // Config will hold the config we
    // capture from a json file and env vars
    type Config struct {
        Version string `json:"version"
required:"true"`
        IsSafe bool `json:"is_safe" default:"true"`
        Secret string `json:"secret"`
    }

    func main() {
        var err error

        // create a temporary file to hold
        // an example json file
        tf, err := ioutil.TempFile("", "tmp")
        if err != nil {
            panic(err)
        }
        defer tf.Close()
        defer os.Remove(tf.Name())

        // create a json file to hold
        // our secrets
        secrets := `{
            "secret": "so so secret"
        }`

        if _, err =
tf.Write(bytes.NewBufferString(secrets).Bytes());
        err != nil {
            panic(err)
        }

        // We can easily set environment variables
        // as needed
        if err = os.Setenv("EXAMPLE_VERSION",
"1.0.0"); err != nil
        {

```



```

        panic(err)
    }
    if err = os.Setenv("EXAMPLE_ISSAFE", "false");
err != nil {
        panic(err)
    }

    c := Config{}
    if err = envvar.LoadConfig(tf.Name(),
"EXAMPLE", &c);
    err != nil {
        panic(err)
    }

    fmt.Println("secrets file contains =",
secrets)

    // We can also read them
    fmt.Println("EXAMPLE_VERSION =",
os.Getenv("EXAMPLE_VERSION"))
    fmt.Println("EXAMPLE_ISSAFE =",
os.Getenv("EXAMPLE_ISSAFE"))

    // The final config is a mix of json and
environment
    // variables
    fmt.Printf("Final Config: %#v\n", c)
}

```

- Выполните `go run main.go`.
- Вы также можете запустить следующие команды:

```

go build
./example

```

- Вы должны увидеть следующий вывод:

```

$ go run main.go
secrets file contains = {
"secret": "so so secret"
}
EXAMPLE_VERSION = 1.0.0
EXAMPLE_ISSAFE = false

```

**Final Config:** `main.Config{Version:"1.0.0", IsSafe:false, Secret:"so so secret"}`

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test` и убедитесь, что все тесты пройдены.

## Как это работает...

Чтение и запись переменных среды с пакетом `os` довольно просто. Сторонняя библиотека `envconfig`, которую использует этот рецепт, — это умный способ захвата переменных среды и указания определенных требований с помощью тегов `struct`.

Функция `LoadConfig` — это гибкий способ получения информации о конфигурации из различных источников без больших накладных расходов или слишком большого количества дополнительных зависимостей. Было бы просто преобразовать основной `config` в другой формат, кроме JSON, или просто всегда использовать переменные среды.

Также обратите внимание на использование ошибок. В этом рецепте мы обернули ошибки по всему коду, чтобы мы могли аннотировать ошибки, не теряя исходной информации об ошибке. Подробнее об этом будет рассказано в [Главе 4, Обработка ошибок в Go](#).

## Конфигурация с использованием TOML, YAML и JSON

Go поддерживает множество форматов конфигурации с использованием сторонних библиотек. Три самых популярных формата данных — это TOML, YAML и JSON. Go может поддерживать JSON «из коробки», а у других есть подсказки о том, как `marshal/unmarshal` или `encode/decode` данные для этих форматов. Эти форматы имеют много преимуществ помимо конфигурации, но в этой главе основное внимание будет уделено преобразованию структуры Go в форму структуры конфигурации. В этом рецепте будут рассмотрены основные операции ввода и вывода с использованием этих форматов.

Эти форматы также предоставляют интерфейс, с помощью которого Go и приложения, написанные на других языках, могут использовать одну и ту же конфигурацию. Также есть ряд инструментов, которые работают с этими форматами и упрощают работу с ними.

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter2/confformat`.
- Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-  
Cookbook-Second-Edition/chapter2/confformat
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-  
Cookbook-Second-Edition/chapter2/confformat
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter2/confformat` или используйте это как возможность написать свой собственный код!
- Создайте файл с именем `toml.go` со следующим содержимым:

```
package confformat  
  
import (  
    "bytes"  
  
    "github.com/BurntSushi/toml"  
)  
  
// TOMLData is our common data struct  
// with TOML struct tags  
type TOMLData struct {  
    Name string `toml:"name"`  
    Age int `toml:"age"`  
}
```

```

// ToTOML dumps the TOMLData struct to
// a TOML format bytes.Buffer
func (t *TOMLData) ToTOML() (*bytes.Buffer, error)
{
    b := &bytes.Buffer{}
    encoder := toml.NewEncoder(b)

    if err := encoder.Encode(t); err != nil {
        return nil, err
    }
    return b, nil
}

// Decode will decode into TOMLData
func (t *TOMLData) Decode(data []byte)
(toml.MetaData, error) {
    return toml.Decode(string(data), t)
}

```

- Создайте файл `yaml.go` со следующим содержимым:

```

package confformat

import (
    "bytes"

    "github.com/go-yaml/yaml"
)

// YAMLData is our common data struct
// with YAML struct tags
type YAMLData struct {
    Name string `yaml:"name"`
    Age int `yaml:"age"`
}

// ToYAML dumps the YAMLData struct to
// a YAML format bytes.Buffer
func (t *YAMLData) ToYAML() (*bytes.Buffer, error)
{
    d, err := yaml.Marshal(t)
    if err != nil {

```

```

        return nil, err
    }

    b := bytes.NewBuffer(d)

    return b, nil
}

// Decode will decode into TOMLData
func (t *YAMLData) Decode(data []byte) error {
    return yaml.Unmarshal(data, t)
}

```

- Создайте файл `json.go` со следующим содержимым:

```

package confformat

import (
    "bytes"
    "encoding/json"
    "fmt"
)

// JSONData is our common data struct
// with JSON struct tags
type JSONData struct {
    Name string `json:"name"`
    Age  int  `json:"age"`
}

// ToJSON dumps the JSONData struct to
// a JSON format bytes.Buffer
func (t *JSONData) ToJSON() (*bytes.Buffer, error)
{
    d, err := json.Marshal(t)
    if err != nil {
        return nil, err
    }

    b := bytes.NewBuffer(d)

    return b, nil
}

```

```

// Decode will decode into JSONData
func (t *JSONData) Decode(data []byte) error {
    return json.Unmarshal(data, t)
}

// OtherJSONExamples shows ways to use types
// beyond structs and other useful functions
func OtherJSONExamples() error {
    res := make(map[string]string)
    err := json.Unmarshal([]byte(`{"key":
"value"}`), &res)
    if err != nil {
        return err
    }

    fmt.Println("We can unmarshal into a map
instead of a
    struct:", res)

    b := bytes.NewReader([]byte(`{"key2":
"value2"}`))
    decoder := json.NewDecoder(b)

    if err := decoder.Decode(&res); err != nil {
        return err
    }

    fmt.Println("we can also use decoders/encoders
to work with
    streams:", res)

    return nil
}

```

- Создайте файл `marshal.go` со следующим содержимым:

```

package confformat

import "fmt"

// MarshalAll takes some data stored in structs
// and converts them to the various data formats

```

```

func MarshalAll() error {
    t := TOMLData{
        Name: "Name1",
        Age: 20,
    }

    j := JSONData{
        Name: "Name2",
        Age: 30,
    }

    y := YAMLData{
        Name: "Name3",
        Age: 40,
    }

    tomlRes, err := t.ToTOML()
    if err != nil {
        return err
    }

    fmt.Println("TOML Marshal =",
tomlRes.String())

    jsonRes, err := j.ToJSON()
    if err != nil {
        return err
    }

    fmt.Println("JSON Marshal=", jsonRes.String())

    yamlRes, err := y.ToYAML()
    if err != nil {
        return err
    }

    fmt.Println("YAML Marshal =",
yamlRes.String())
    return nil
}

```

- Создайте файл `unmarshal.go` со следующим содержимым:

```

package confformat
import "fmt"
const (
    exampleTOML = `name="Example1"
age=99
`
    exampleJSON = `{"name":"Example2","age":98}`
    exampleYAML = `name: Example3
age: 97
`
)
// UnmarshalAll takes data in various formats
// and converts them into structs
func UnmarshalAll() error {
    t := TOMLData{}
    j := JSONData{}
    y := YAMLData{}
    if _, err := t.Decode([]byte(exampleTOML));
err != nil {
        return err
    }
    fmt.Println("TOML Unmarshal =", t)

    if err := j.Decode([]byte(exampleJSON)); err
!= nil {
        return err
    }
    fmt.Println("JSON Unmarshal =", j)

    if err := y.Decode([]byte(exampleYAML)); err
!= nil {
        return err
    }
    fmt.Println("Yaml Unmarshal =", y)
    return nil
}

```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл `main.go` со следующим содержимым:

```

package main

import "github.com/PacktPublishing/

```



## Go-Programming-Cookbook-Second-Edition/ chapter2/confformat"

```
func main() {
    if err := confformat.MarshalAll(); err != nil
{
    panic(err)
}

    if err := confformat.UnmarshalAll(); err !=
nil {
    panic(err)
}

    if err := confformat.OtherJSONExamples(); err
!= nil {
    panic(err)
}
}
```

- Выполните `go run main.go`.
- Вы также можете запустить следующие команды:

```
$ go build
$ ./example
```

- Вы должны увидеть следующий вывод:

```
$ go run main.go
TOML Marshal = name = "Name1"
age = 20
```

```
JSON Marshal= {"name":"Name2","age":30}
YAML Marshal = name: Name3
age: 40
```

```
TOML Unmarshal = {Example1 99}
JSON Unmarshal = {Example2 98}
Yaml Unmarshal = {Example3 97}
```

We can unmarshal into a map instead of a struct:

```
map[key:value]
```

we can also use decoders/encoders to work with streams:

```
map[key:value key2:value2]
```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Этот рецепт предоставил нам примеры того, как использовать синтаксические анализаторы TOML, YAML и JSON как для записи необработанных данных в структуру `go`, так и для чтения данных из нее в соответствующий формат. Как и в рецептах из [Главы 1](#), «*Ввод-вывод и файловые системы*», мы увидели, как часто приходится быстро переключаться между `[]byte`, `string`, `bytes.Buffer` и другими интерфейсами ввода-вывода.

Пакет `encoding/json` является наиболее полным в обеспечении кодирования, маршалинга и других методов для работы с форматом JSON. Мы абстрагировали их с помощью наших функций `ToFormat`, и было бы очень просто присоединить несколько методов, таких как этот, чтобы мы могли использовать единую структуру, которую можно быстро преобразовать в любой из этих типов или из него.

Этот рецепт также касается тегов структуры и их использования. В предыдущей главе они также использовались, и это обычный способ дать подсказки пакетам и библиотекам о том, как обрабатывать данные, содержащиеся в структуре.

## Работа с каналами Unix

Каналы Unix полезны, когда мы передаем вывод одной программы на ввод другой. Например, взгляните на следующий код:

```
$ echo "test case" | wc -l
1
```

В приложении Go левая часть канала может быть прочитана с помощью `os.Stdin`, который действует как файловый дескриптор. Чтобы продемонстрировать это, этот рецепт примет ввод с левой

стороны канала и вернет список слов и их количество вхождений. Эти слова будут токенизированы на пустом пространстве.

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter2/pipes`.
- Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter2/pipes
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter2/pipes
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter2/pipes` или используйте это как возможность написать свой собственный код!
- Создайте файл с именем `pipe.go` со следующим содержимым:

```
package main

import (
    "bufio"
    "fmt"
    "io"
    "os"
)

// WordCount takes a file and returns a map
// with each word as a key and it's number of
// appearances as a value
func WordCount(f io.Reader) map[string]int {
    result := make(map[string]int)

    // make a scanner to work on the file
```

```

// io.Reader interface
scanner := bufio.NewScanner(f)
scanner.Split(bufio.ScanWords)

for scanner.Scan() {
    result[scanner.Text()]++
}

if err := scanner.Err(); err != nil {
    fmt.Fprintln(os.Stderr, "reading input:",
err)
}

return result
}

func main() {
    fmt.Printf("string:
number_of_occurrences\n\n")
    for key, value := range WordCount(os.Stdin) {
        fmt.Printf("%s: %d\n", key, value)
    }
}

```

- Выполните `echo "some string" | go run pipes.go`.
- Вы также можете запустить следующие команды:

```

$ go build
echo "some string" | ./pipes

```

Вы должны увидеть следующий вывод:

```

$ echo "test case" | go run pipes.go
string: number_of_occurrences

```

```

test: 1
case: 1

```

```

$ echo "test case test" | go run pipes.go
string: number_of_occurrences

```

```

test: 2
case: 1

```

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Работать с каналами в Go довольно просто, особенно если вы знакомы с работой с файлами. Например, вы можете использовать рецепт конвейера из [Главы 1](#), «Ввод-вывод и файловые системы», чтобы создать приложение `tee` ([https://en.wikipedia.org/wiki/Tee\\_\(command\)](https://en.wikipedia.org/wiki/Tee_(command))), где все, что передается по конвейеру, немедленно записывается в `stdout` и в файл.

В этом рецепте используется сканер для токенизации интерфейса `io.Reader` файлового объекта `os.Stdin`. Вы можете видеть, как вы должны проверять наличие ошибок после завершения всех операций чтения.

## Перехват и обработка сигналов

Сигналы — это полезный способ для пользователя или ОС убить запущенное приложение. Иногда имеет смысл обрабатывать эти сигналы более изящно, чем поведение по умолчанию. Go предоставляет механизм для захвата и обработки сигналов. В этом рецепте мы рассмотрим обработку сигналов с помощью сигнала, который обрабатывает горутины.

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter2/signals`.
- Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter2/signals
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter2/signals
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter2/signals` или используйте это как возможность написать свой собственный код!
- Создайте файл `signal.go` со следующим содержимым:

```
package main

import (
    "fmt"
    "os"
    "os/signal"
    "syscall"
)

// CatchSig sets up a listener for
// SIGINT interrupts
func CatchSig(ch chan os.Signal, done chan bool) {
    // block on waiting for a signal
    sig := <-ch
    // print it when it's received
    fmt.Println("nsig received:", sig)

    // we can set up handlers for all types of
    // sigs here
    switch sig {
    case syscall.SIGINT:
        fmt.Println("handling a SIGINT now!")
    case syscall.SIGTERM:
        fmt.Println("handling a SIGTERM in an
entirely
        different way!")
    default:
        fmt.Println("unexpected signal received")
    }

    // terminate
    done <- true
}
```

```

    }

    func main() {
        // initialize our channels
        signals := make(chan os.Signal)
        done := make(chan bool)

        // hook them up to the signals lib
        signal.Notify(signals, syscall.SIGINT,
syscall.SIGTERM)

        // if a signal is caught by this go routine
        // it will write to done
        go CatchSig(signals, done)

        fmt.Println("Press ctrl-c to terminate...")
        // the program blocks until someone writes to
done
        <-done
        fmt.Println("Done!")
    }

```

- Выполните следующие команды:

```

$ go build
$ ./signals

```

- Попробуйте запустить код, а затем нажмите *Ctrl* + *C*. Вы должны увидеть следующее:

```

$./signals
Press ctrl-c to terminate...
^C
sig received: interrupt
handling a SIGINT now!
Done!

```

- Попробуйте запустить его снова. Затем из отдельного Терминала определяем PID и закрываем приложение:

```

$./signals
Press ctrl-c to terminate...

```

```
# in a separate terminal
$ ps -ef | grep signals
501 30777 26360 0 5:00PM ttys000 0:00.00 ./signals
```

```
$ kill -SIGTERM 30777
```

```
# in the original terminal
```

```
sig received: terminated
handling a SIGTERM in an entirely different way!
Done!
```

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

В этом рецепте используются каналы, которые более подробно рассматриваются в [Главе 10 «Параллелизм и конкурентность»](#). Для функции `signal.Notify` требуется канал для отправки уведомлений о сигналах, а также типы сигналов, о которых мы заботимся. Затем мы настраиваем функцию в подпрограмме Go для обработки любых действий на канале, который мы передали этой функции. Получив сигнал, мы можем обращаться с ним, как захотим. Мы можем завершить приложение, ответить сообщением и по-разному вести себя для разных сигналов. Команда `kill` — хороший способ проверить передачу сигналов приложениям.

Мы также используем `done` канал, чтобы заблокировать завершение приложения до тех пор, пока не будет получен сигнал. В противном случае программа немедленно завершится. В этом нет необходимости для долго работающих приложений, таких как веб-приложения. Может быть очень полезно создать соответствующие подпрограммы обработки сигналов для выполнения очистки, особенно в приложениях с большим количеством подпрограмм Go, которые содержат значительное количество состояний. Практическим примером корректного завершения работы может быть разрешение текущим обработчикам завершать свои HTTP-запросы, не завершая их на полпути.



# Приложение для раскрашивания ANSI

Раскрашивание терминального приложения ANSI выполняется с помощью различных кодов до и после раздела текста, который вы хотите раскрасить. Этот рецепт исследует базовый механизм окраски, который окрашивает текст в красный или обычный цвет. Для полного приложения взгляните на <https://github.com/agtorre/gocolorize>, который поддерживает гораздо больше цветов и типов текста, а также реализует интерфейс `fmt.Formatter` для упрощения печати.

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter2/ansicolor`.
- Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter2/ansicolor
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter2/ansicolor
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter2/ansicolor` или используйте это как возможность написать свой собственный код!
- Создайте файл с именем `color.go` со следующим содержимым:

```
package ansicolor

import "fmt"

//Color of text
```

```

type Color int

const (
    // ColorNone is default
    ColorNone = iota
    // Red colored text
    Red
    // Green colored text
    Green
    // Yellow colored text
    Yellow
    // Blue colored text
    Blue
    // Magenta colored text
    Magenta
    // Cyan colored text
    Cyan
    // White colored text
    White
    // Black colored text
    Black Color = -1
)

// ColorText holds a string and its color
type ColorText struct {
    TextColor Color
    Text      string
}

func (r *ColorText) String() string {
    if r.TextColor == ColorNone {
        return r.Text
    }

    value := 30
    if r.TextColor != Black {
        value += int(r.TextColor)
    }
    return fmt.Sprintf("\033[0;%dm%s\033[0m", value,
r.Text)
}

```

- Создайте новый каталог с именем `example` и перейдите к нему.

- Создайте файл `main.go` со следующим содержимым:

```
package main

import (
    "fmt"

    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter2/ansicolor"
)

func main() {
    r := ansicolor.ColorText{
        TextColor: ansicolor.Red,
        Text:      "I'm red!",
    }

    fmt.Println(r.String())

    r.TextColor = ansicolor.Green
    r.Text = "Now I'm green!"

    fmt.Println(r.String())

    r.TextColor = ansicolor.ColorNone
    r.Text = "Back to normal..."

    fmt.Println(r.String())
}
```

- Выполните `go run main.go`.
- Вы также можете запустить следующие команды:

```
$ go build
$ ./example
```

- Вы должны увидеть следующий вывод с цветным текстом, если ваш Терминал поддерживает формат окраски ANSI:

```
$ go run main.go
I'm red!
```

**Now I'm green!**  
**Back to normal...**

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Это приложение использует структуру для сохранения состояния цветного текста. В этом случае он сохраняет цвет текста и значение текста. Окончательная строка визуализируется при вызове метода `String()`, который возвращает либо цветной текст, либо обычный текст, в зависимости от значений, хранящихся в структуре. По умолчанию текст будет простым.

## 3. Преобразование данных и композиция

Понимание системы типизации Go — важный шаг к освоению всех уровней разработки Go. В этой главе будут показаны некоторые примеры преобразования между типами данных, работы с очень большими числами, работы с валютой, использования различных типов кодирования и декодирования, включая Base64 и `gob`, а также создания пользовательских коллекций с использованием замыканий. В этой главе будут рассмотрены следующие рецепты:

- Преобразование типов данных и приведение интерфейсов
- Работа с числовыми типами данных с использованием `math` и `math/big`
- Преобразование валюты и рассмотрение `float64`
- Использование указателей и SQL NullTypes для кодирования и декодирования
- Кодирование и декодирование данных Go
- Структурные теги и базовое отражение в Go
- Реализация коллекций с использованием замыканий

### Технические требования

Чтобы продолжить выполнение всех рецептов в этой главе, настройте свою среду в соответствии со следующими шагами:

- Загрузите и установите Go 1.12.6 или более позднюю версию в своей операционной системе по адресу <https://golang.org/doc/install>.
- Откройте терминал или консольное приложение, создайте и перейдите в каталог проекта, например `~/projects/go-programming-cookbook`. Весь наш код будет запускаться и изменяться из этого каталога.
- Скопируйте последний код в `~/projects/go-programming-cookbook-original` и работайте с этим каталогом, а не вводите примеры вручную:

```
$ git clone git@github.com:PacktPublishing/Go-Programming-Cookbook-Second-Edition.git go-programming-cookbook-original
```

## Преобразование типов данных и приведение интерфейсов

Go обычно очень гибок, когда используется для преобразования данных из одного типа в другой. Тип может наследовать другой тип следующим образом:

```
type A int
```

Мы всегда можем вернуться к типу, который мы унаследовали, следующим образом:

```
var a A = 1
fmt.Println(int(a))
```

Существуют также удобные функции для преобразования чисел с помощью приведения, между строками и другими типами с помощью `fmt.Sprint` и `strconv`, а также между интерфейсами и типами с использованием отражения. Этот рецепт исследует некоторые из этих основных преобразований, которые будут использоваться на протяжении всей книги.

### Как это сделать...

Следующие шаги описывают, как написать и запустить ваше приложение:

- В приложении терминала/консоли создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter3/dataconv`.
- Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter3/dataconv
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter3/dataconv
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter3/dataconv` или используйте это как упражнение для написания собственного кода!
- Создайте файл с именем `dataconv.go` со следующим содержимым:

```
package dataconv

import "fmt"

// ShowConv demonstrates some type conversion
func ShowConv() {
    // int
    var a = 24

    // float 64
    var b = 2.0

    // convert the int to a float64 for this
calculation    c := float64(a) * b
               fmt.Println(c)

    // fmt.Sprintf is a good way to convert to
strings        precision := fmt.Sprintf("%.2f", b)

               // print the value and the type
               fmt.Printf("%s - %T\n", precision, precision)
    }
}
```

- Создайте файл с именем `strconv.go` со следующим содержимым:

```
package dataconv

import (
    "fmt"
```

```

        "strconv"
    )

    // Strconv demonstrates some strconv
    // functions
    func Strconv() error {
        //strconv is a good way to convert to and from
strings
        s := "1234"
        // we can specify the base (10) and precision
        // 64 bit
        res, err := strconv.ParseInt(s, 10, 64)
        if err != nil {
            return err
        }

        fmt.Println(res)

        // lets try hex
        res, err = strconv.ParseInt("FF", 16, 64)
        if err != nil {
            return err
        }

        fmt.Println(res)

        // we can do other useful things like:
        val, err := strconv.ParseBool("true")
        if err != nil {
            return err
        }

        fmt.Println(val)

        return nil
    }

```

- Создайте файл `interfaces.go` со следующим содержимым:

```

package dataconv

import "fmt"

```



```

// CheckType will print based on the
// interface type
func CheckType(s interface{}) {
    switch s.(type) {
    case string:
        fmt.Println("It's a string!")
    case int:
        fmt.Println("It's an int!")
    default:
        fmt.Println("not sure what it is...")
    }
}

// Interfaces demonstrates casting
// from anonymous interfaces to types
func Interfaces() {
    CheckType("test")
    CheckType(1)
    CheckType(false)

    var i interface{}
    i = "test"

    // manually check an interface
    if val, ok := i.(string); ok {
        fmt.Println("val is", val)
    }

    // this one should fail
    if _, ok := i.(int); !ok {
        fmt.Println("uh oh! glad we handled this")
    }
}

```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import "github.com/PacktPublishing/
        Go-Programming-Cookbook-Second-Edition/
        chapter3/dataconv"

```

```
func main() {
    dataconv.ShowConv()
    if err := dataconv.Strconv(); err != nil {
        panic(err)
    }
    dataconv.Interfaces()
}
```

- Выполните `go run main.go`. Вы также можете запустить следующие команды:

```
$ go build
$ ./example
```

Вы должны увидеть следующий вывод:

```
$ go run main.go
48
2.00 - string
1234
255
true
It's a string!
It's an int!
not sure what it is...
val is test
uh oh! glad we handled this
```

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

В этом рецепте показано, как выполнять приведение типов, заключая их в новый тип с помощью пакета `strconv` и отражения интерфейса. Эти методы позволяют разработчикам Go быстро конвертировать различные абстрактные типы Go. Эти первые два метода будут возвращать ошибки во время компиляции, но ошибки в отражении интерфейса могут быть обнаружены только во время выполнения. Если вы неправильно отразите неподдерживаемый тип, ваша программа запаникует. Переключение между типами — это способ обобщения, который также демонстрируется в этом рецепте.

Преобразование становится важным для таких пакетов, как `math`, которые работают исключительно с `float64`.

## Работа с числовыми типами данных с использованием `math` и `math/big`

Пакеты `math` и `math/big` ориентированы на то, чтобы предоставить языку Go более сложные математические операции, такие как `Pow`, `Sqrt` и `Cos`. Сам пакет `math` работает преимущественно с `float64`, если в функции не указано иное. Пакет `math/big` предназначен для чисел, которые слишком велики для представления в 64-битном значении. Этот рецепт покажет некоторые основные принципы использования пакета `math` и продемонстрирует, как использовать `math/big` для последовательности Фибоначчи.

### Как это сделать...

Следующие шаги описывают, как написать и запустить ваше приложение:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter3/math`.
- Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter3/math
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter3/math
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter3/math` или используйте это как упражнение для написания собственного кода!
- Создайте файл с именем `fib.go` со следующим содержимым:

```

package math

import "math/big"

// global to memoize fib
var memoize map[int]*big.Int

func init() {
    // initialize the map
    memoize = make(map[int]*big.Int)
}

// Fib prints the nth digit of the fibonacci
sequence
// it will return 1 for anything < 0 as well...
// it's calculated recursively and use big.Int
since
// int64 will quickly overflow
func Fib(n int) *big.Int {
    if n < 0 {
        return big.NewInt(1)
    }

    // base case
    if n < 2 {
        memoize[n] = big.NewInt(1)
    }

    // check if we stored it before
    // if so return with no calculation
    if val, ok := memoize[n]; ok {
        return val
    }

    // initialize map then add previous 2 fib
values
    memoize[n] = big.NewInt(0)
    memoize[n].Add(memoize[n], Fib(n-1))
    memoize[n].Add(memoize[n], Fib(n-2))

    // return result
    return memoize[n]
}

```

- Создайте файл с именем `math.go` со следующим содержимым:

```
package math

import (
    "fmt"
    "math"
)

// Examples demonstrates some of the functions
// in the math package
func Examples() {
    //sqrt Examples
    i := 25

    // i is an int, so convert
    result := math.Sqrt(float64(i))

    // sqrt of 25 == 5
    fmt.Println(result)

    // ceil rounds up
    result = math.Ceil(9.5)
    fmt.Println(result)

    // floor rounds down
    result = math.Floor(9.5)
    fmt.Println(result)

    // math also stores some consts:
    fmt.Println("Pi:", math.Pi, "E:", math.E)
}
```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```
package main

import (
    "fmt"

    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/"
```

```

        chapter3/math"
    )

    func main() {
        math.Examples()

        for i := 0; i < 10; i++ {
            fmt.Printf("%v ", math.Fib(i))
        }
        fmt.Println()
    }

```

- Выполните `go run main.go`. Вы также можете запустить следующее:

```

$ go build
$ ./example

```

Вы должны увидеть следующий вывод:

```

$ go run main.go
5
10
9
Pi: 3.141592653589793 E: 2.718281828459045
1 1 2 3 5 8 13 21 34 55

```

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Пакет `math` позволяет выполнять в Go сложные математические операции. Этот рецепт следует использовать в сочетании с этим пакетом для выполнения сложных операций с плавающей запятой и преобразования между типами по мере необходимости. Стоит отметить, что даже с `float64` могут быть ошибки округления для некоторых чисел с плавающей запятой; следующий рецепт демонстрирует некоторые методы решения этой проблемы.

Раздел `math/big` демонстрирует рекурсивную последовательность Фибоначчи. Если вы измените `main.go` так, чтобы число циклов было

намного больше 10, вы быстро переполните `int64`, если он использовался вместо `big.Int`. В пакете `big.Int` также есть вспомогательные методы для преобразования больших типов в другие типы.

## Преобразование валюты и рассмотрение `float64`

Работа с валютой всегда сложный процесс. Может показаться заманчивым представить деньги как `float64`, но это может привести к довольно сложным (и неправильным) ошибкам округления при выполнении вычислений. По этой причине предпочтительнее думать о деньгах в центах и хранить цифру как экземпляр `int64`.

При сборе пользовательского ввода из форм, командной строки или других источников деньги обычно представляются в долларовой форме. По этой причине лучше обрабатывать ее как строку и преобразовывать эту строку непосредственно в центы без преобразования с плавающей запятой. В этом рецепте представлены способы преобразования строкового представления валюты в экземпляр типа `int64` (центы) и обратно.

### Как это сделать...

Следующие шаги описывают, как написать и запустить ваше приложение:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter3/currency`
- Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter3/currency
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

`module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter3/currency`

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter3/currency` или используйте это как упражнение для написания собственного кода!
- Создайте файл с именем `dollars.go` со следующим содержимым:

```
package currency

import (
    "errors"
    "strconv"
    "strings"
)

// ConvertStringDollarsToPennies takes a dollar
amount // as a string, i.e. 1.00, 55.12 etc and converts
it // into an int64
(int64, func ConvertStringDollarsToPennies(amount string)
error) {
float // check if amount can convert to a valid
    _, err := strconv.ParseFloat(amount, 64)
    if err != nil {
        return 0, err
    }

    // split the value on "."
    groups := strings.Split(amount, ".")

    // if there is no . result will still be
    // captured here
    result := groups[0]

    // base string
    r := ""

    // handle the data after the "."
    if len(groups) == 2 {
```



```

        if len(groups[1]) != 2 {
            return 0, errors.New("invalid cents")
        }
        r = groups[1]
    }

    // pad with 0, this will be
    // 2 0's if there was no .
    for len(r) < 2 {
        r += "0"
    }

    result += r

    // convert it to an int
    return strconv.ParseInt(result, 10, 64)
}

```

- Создайте файл с именем `pennies.go` со следующим содержимым:

```

package currency

import (
    "strconv"
)

// ConvertPenniesToDollarString takes a penny
amount as
// an int64 and returns a dollar string
representation
func ConvertPenniesToDollarString(amount int64)
string {
    // parse the pennies as a base 10 int
    result := strconv.FormatInt(amount, 10)

    // check if negative, will set it back later
    negative := false
    if result[0] == '-' {
        result = result[1:]
        negative = true
    }

    // left pad with 0 if we're passed in value <

```

100

```

        for len(result) < 3 {
            result = "0" + result
        }
        length := len(result)

        // add in the decimal
        result = result[0:length-2] + "." +
result[length-2:]

        // from the negative we stored earlier!
        if negative {
            result = "-" + result
        }

        return result
    }

```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "fmt"

    "github.com/PacktPublishing/
Go-Programming-Cookbook-Second-Edition/
chapter3/currency"
)

func main() {
    // start with our user input
    // of fifteen dollars and 93 cents
    userInput := "15.93"

    pennies, err :=
currency.ConvertStringDollarsToPennies(userInput)
    if err != nil {
        panic(err)
    }
}

```

```

        fmt.Printf("User input converted to %d
pennies\n", pennies)

        // adding 15 cents
        pennies += 15

        dollars :=
currency.ConvertPenniesToDollarString(pennies)

        fmt.Printf("Added 15 cents, new values is %s
dollars\n",
dollars)
    }

```

- Выполните `go run main.go`. Вы также можете запустить следующее:

```

$ go build
$ ./example

```

Вы должны увидеть следующий вывод:

```

$ go run main.go
User input converted to 1593 pennies
Added 15 cents, new values is 16.08 dollars

```

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

В этом рецепте используются пакеты `strconv` и `strings` для преобразования валюты между долларами в строковом формате и пенни в формате `int64`. Он делает это без преобразования в тип `float64`, что может привести к ошибке округления, и делает это только для проверки.

Функции `strconv.ParseInt` и `strconv.FormatInt` очень полезны для преобразования в `int64` и строки и из них. Мы также использовали тот факт, что строки Go можно легко добавлять и нарезать по мере необходимости.

# Использование указателей и SQL `NullTypes` для кодирования и декодирования

Когда вы кодируете или декодируете объект в Go, для типов, которые не заданы явно, будут установлены значения по умолчанию. Строки по умолчанию будут пустой строкой (`"`), а целые числа по умолчанию будут равны `0`, например. Обычно это нормально, если только `0` не означает что-то для вашего API или службы, которая использует пользовательский ввод или возвращает его.

Кроме того, если вы используете теги `struct`, такие как `json omitempty`, значение `0` будет игнорироваться, даже если оно допустимо. Другим примером этого является `Null`, который возвращается из SQL. Какое значение лучше всего представляет `Null` для `Int`? В этом рецепте будут рассмотрены некоторые способы решения этой проблемы разработчиками Go.

## Как это сделать...

Следующие шаги описывают, как написать и запустить ваше приложение:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter3/nulls`.
- Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter3/nulls
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter3/nulls
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter3/nulls` или используйте это как упражнение

для написания собственного кода!

- Создайте файл с именем `base.go` со следующим содержимым:

```
package nulls

import (
    "encoding/json"
    "fmt"
)

// json that has name but not age
const (
    jsonBlob = `{"name": "Aaron"}`
    fulljsonBlob = `{"name": "Aaron", "age": 0}`
)

// Example is a basic struct with age
// and name fields
type Example struct {
    Age int `json:"age,omitempty"`
    Name string `json:"name"`
}

// BaseEncoding shows encoding and
// decoding with normal types
func BaseEncoding() error {
    e := Example{}

    // note that no age = 0 age
    if err := json.Unmarshal([]byte(jsonBlob),
&e); err != nil
    {
        return err
    }
    fmt.Printf("Regular Unmarshal, no age: %+v\n",
e)

    value, err := json.Marshal(&e)
    if err != nil {
        return err
    }
    fmt.Println("Regular Marshal, with no age:",
```

```

string(value))

    if err := json.Unmarshal([]byte(fulljsonBlob),
&e);
    err != nil {
        return err
    }
    fmt.Printf("Regular Unmarshal, with age = 0:
%+v\n", e)

    value, err = json.Marshal(&e)
    if err != nil {
        return err
    }
    fmt.Println("Regular Marshal, with age = 0:",
string(value))

    return nil
}

```

- Создайте файл с именем `pointer.go` со следующим содержимым:

```

package nulls

import (
    "encoding/json"
    "fmt"
)

// ExamplePointer is the same, but
// uses a *Int
type ExamplePointer struct {
    Age *int `json:"age,omitempty"`
    Name string `json:"name"`
}

// PointerEncoding shows methods for
// dealing with nil/omitted values
func PointerEncoding() error {

    // note that no age = nil age
    e := ExamplePointer{}
    if err := json.Unmarshal([]byte(jsonBlob),

```

```

&e); err != nil
    {
        return err
    }
    fmt.Printf("Pointer Unmarshal, no age: %+v\n",
e)

    value, err := json.Marshal(&e)
    if err != nil {
        return err
    }
    fmt.Println("Pointer Marshal, with no age:",
string(value))

    if err := json.Unmarshal([]byte(fulljsonBlob),
&e);
    err != nil {
        return err
    }
    fmt.Printf("Pointer Unmarshal, with age = 0:
%+v\n", e)

    value, err = json.Marshal(&e)
    if err != nil {
        return err
    }
    fmt.Println("Pointer Marshal, with age = 0:",
string(value))

    return nil
}

```

- Создайте файл с именем `nullencoding.go` со следующим содержимым:

```

package nulls

import (
    "database/sql"
    "encoding/json"
    "fmt"
)

```

```

type nullInt64 sql.NullInt64

// ExampleNullInt is the same, but
// uses a sql.NullInt64
type ExampleNullInt struct {
    Age *nullInt64 `json:"age,omitempty"`
    Name string `json:"name"`
}

func (v *nullInt64) MarshalJSON() ([]byte, error)
{
    if v.Valid {
        return json.Marshal(v.Int64)
    }
    return json.Marshal(nil)
}

func (v *nullInt64) UnmarshalJSON(b []byte) error
{
    v.Valid = false
    if b != nil {
        v.Valid = true
        return json.Unmarshal(b, &v.Int64)
    }
    return nil
}

// NullEncoding shows an alternative method
// for dealing with nil/omitted values
func NullEncoding() error {
    e := ExampleNullInt{}

    // note that no means an invalid value
    if err := json.Unmarshal([]byte(jsonBlob),
&e); err != nil
    {
        return err
    }
    fmt.Printf("nullInt64 Unmarshal, no age:
%+v\n", e)

    value, err := json.Marshal(&e)
    if err != nil {

```



```

        return err
    }
    fmt.Println("nullInt64 Marshal, with no age:",
string(value))

    if err := json.Unmarshal([]byte(fulljsonBlob),
&e);
    err != nil {
        return err
    }
    fmt.Printf("nullInt64 Unmarshal, with age = 0:
%+v\n", e)

    value, err = json.Marshal(&e)
    if err != nil {
        return err
    }
    fmt.Println("nullInt64 Marshal, with age =
0:",
string(value))

    return nil
}

```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "fmt"

    "github.com/PacktPublishing/
Go-Programming-Cookbook-Second-Edition/
chapter3/nulls"
)

func main() {
    if err := nulls.BaseEncoding(); err != nil {
        panic(err)
    }
    fmt.Println()
}

```

```

        if err := nulls.PointerEncoding(); err != nil
    {
        panic(err)
    }
    fmt.Println()

    if err := nulls.NullEncoding(); err != nil {
        panic(err)
    }
}

```

- Выполните `go run main.go`. Вы также можете запустить следующее:

```

$ go build
$ ./example

```

Вы должны увидеть следующий вывод:

```

$ go run main.go
Regular Unmarshal, no age: {Age:0 Name:Aaron}
Regular Marshal, with no age: {"name":"Aaron"}
Regular Unmarshal, with age = 0: {Age:0 Name:Aaron}
Regular Marshal, with age = 0: {"name":"Aaron"}

Pointer Unmarshal, no age: {Age:<nil> Name:Aaron}
Pointer Marshal, with no age: {"name":"Aaron"}
Pointer Unmarshal, with age = 0: {Age:0xc42000a610
Name:Aaron}
Pointer Marshal, with age = 0: {"age":0,"name":"Aaron"}

nullInt64 Unmarshal, no age: {Age:<nil> Name:Aaron}
nullInt64 Marshal, with no age: {"name":"Aaron"}
nullInt64 Unmarshal, with age = 0: {Age:0xc42000a750
Name:Aaron}
nullInt64 Marshal, with age = 0: {"age":0,"name":"Aaron"}

```

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Переключение со значения на указатель — это быстрый способ выражения нулевых значений при маршалинге и демаршалинге. Установка этих значений может быть немного сложной, так как вы не можете присвоить их непосредственно указателю, `-- *a := 1`, но в остальном это гибкий способ работы с этим.

В этом рецепте также продемонстрирован альтернативный метод с использованием типа `sql.NullInt64`. Это обычно используется с SQL, и устанавливается значение `valid`, если возвращается что-либо, кроме `Null`; в противном случае он устанавливает значение `Null`. Мы добавили метод `MarshalJSON` и метод `UnmarshalJSON`, чтобы позволить этому типу взаимодействовать с пакетом `JSON`, и решили использовать указатель, чтобы `omitempty` продолжала работать должным образом.

## Кодирование и декодирование данных Go

В Go есть ряд альтернативных типов кодирования, отличных от JSON, TOML и YAML. Они в основном предназначены для передачи данных между процессами Go с такими вещами, как проводные протоколы и RPC, или в случаях, когда некоторые форматы символов ограничены.

Этот рецепт исследует, как кодировать и декодировать формат `gob` и `base64`. В последующих главах будут рассмотрены такие протоколы, как GRPC.

### Как это сделать...

Следующие шаги описывают, как написать и запустить ваше приложение:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter3/encoding`.
- Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter3/encoding
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

`module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter3/encoding`

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter3/encoding` или используйте это как упражнение, чтобы написать свой собственный код!
- Создайте файл с именем `gob.go` со следующим содержимым:

```
package encoding

import (
    "bytes"
    "encoding/gob"
    "fmt"
)

// pos stores the x, y position
// for Object
type pos struct {
    X      int
    Y      int
    Object string
}

// GobExample demonstrates using
// the gob package
func GobExample() error {
    buffer := bytes.Buffer{}

    p := pos{
        X:      10,
        Y:      15,
        Object: "wrench",
    }

    // note that if p was an interface
    // we'd have to call gob.Register first

    e := gob.NewEncoder(&buffer)
    if err := e.Encode(&p); err != nil {
```

```

        return err
    }

    // note this is a binary format so it wont
print well    fmt.Println("Gob Encoded valued length: ",
                    len(buffer.Bytes()))

    p2 := pos{}
    d := gob.NewDecoder(&buffer)
    if err := d.Decode(&p2); err != nil {
        return err
    }

    fmt.Println("Gob Decode value: ", p2)

    return nil
}

```

- Создайте файл **base64.go** со следующим содержимым:

```

package encoding

import (
    "bytes"
    "encoding/base64"
    "fmt"
    "io/ioutil"
)

// Base64Example demonstrates using
// the base64 package
func Base64Example() error {
    // base64 is useful for cases where
    // you can't support binary formats
    // it operates on bytes/strings

    // using helper functions and URL encoding
    value :=
base64.URLEncoding.EncodeToString([]byte("encoding
    some data!"))
    fmt.Println("With EncodeToString and
URLEncoding: ", value)

```

```

        // decode the first value
        decoded, err :=
base64.URLEncoding.DecodeString(value)
        if err != nil {
            return err
        }
        fmt.Println("With DecodeToString and
URLEncoding: ",
            string(decoded))

        return nil
    }

    // Base64ExampleEncoder shows similar examples
    // with encoders/decoders
    func Base64ExampleEncoder() error {
        // using encoder/ decoder
        buffer := bytes.Buffer{}

        // encode into the buffer
        encoder :=
base64.NewEncoder(base64.StdEncoding, &buffer)

        if _, err := encoder.Write([]byte("encoding
some other
data")); err != nil {
            return err
        }

        // be sure to close
        if err := encoder.Close(); err != nil {
            return err
        }

        fmt.Println("Using encoder and StdEncoding: ",
            buffer.String())

        decoder :=
base64.NewDecoder(base64.StdEncoding, &buffer)
        results, err := ioutil.ReadAll(decoder)
        if err != nil {
            return err
        }
    }

```

```

    }

    fmt.Println("Using decoder and StdEncoding: ",
        string(results))

    return nil
}

```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "github.com/PacktPublishing/
        Go-Programming-Cookbook-Second-Edition/
        chapter3/encoding"
)

func main() {
    if err := encoding.Base64Example(); err != nil
{
        panic(err)
    }

    if err := encoding.Base64ExampleEncoder(); err
!= nil {
        panic(err)
    }

    if err := encoding.GobExample(); err != nil {
        panic(err)
    }
}

```

- Выполните `go run main.go`. Вы также можете запустить следующее:

```

$ go build
$ ./example

```

Вы должны увидеть следующий вывод:

```
$ go run main.go
With EncodeToString and URLEncoding:
ZW5jb2Rpbmcgc29tZSBkYXRhIQ==
With DecodeToString and URLEncoding: encoding some data!
Using encoder and StdEncoding:
ZW5jb2Rpbmcgc29tZSBvdGhlcjBkYXRh
Using decoder and StdEncoding: encoding some other data
Gob Encoded valued length: 57
Gob Decode value: {10 15 wrench}
```

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Кодирование Gob — это формат потоковой передачи, созданный с учетом типов данных Go. Это наиболее эффективно при отправке и кодировании множества последовательных элементов. Для одного элемента другие форматы кодирования, такие как JSON, потенциально более эффективны и переносимы. Несмотря на это, гоб-кодирование упрощает упорядочение больших сложных структур и их реконструкцию в отдельном процессе. Хотя это не было показано здесь, gob также может работать с пользовательскими типами или неэкспортированными типами с помощью пользовательских методов `MarshalBinary` и `UnmarshalBinary`.

Кодировка Base64 полезна для связи через URL-адреса в запросах `GET` или для создания кодировки строкового представления двоичных данных. Большинство языков могут поддерживать этот формат и демаршализовать данные на другом конце. В результате обычно кодируются такие вещи, как полезные данные JSON, в тех случаях, когда формат JSON не поддерживается.

## Структурные теги и базовая рефлексия в Go

Рефлексия — сложная тема, которую нельзя охватить в одном рецепте; однако практическое применение рефлексии связано со структурными тегами. По своей сути `struct` теги — это просто строки ключ-



значение: вы ищете ключ, а затем имеете дело со значением. Как вы можете себе представить, для чего-то вроде маршалинга и демаршалинга JSON очень сложно работать с этими значениями.

Пакет `reflect` предназначен для опроса и понимания объектов интерфейса. У него есть вспомогательные методы для просмотра различных типов структур, значений, тегов структур и многого другого. Если вам нужно нечто большее, чем простое преобразование интерфейса, как в начале этой главы, вам следует обратить внимание на этот пакет.

## Как это сделать...

Следующие шаги описывают, как написать и запустить ваше приложение:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter3/tags`.
- Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter3/tags
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter3/tags
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter3/tags` или используйте это как упражнение для написания собственного кода!
- Создайте файл с именем `serialize.go` со следующим содержимым:

```
package tags

import "reflect"

// SerializeStructStrings converts a struct
// to our custom serialization format
```

```

types    // it honors serialize struct tags for string
func SerializeStructStrings(s interface{})
(string, error) {
    result := ""

    // reflect the interface into
    // a type
    r := reflect.TypeOf(s)
    value := reflect.ValueOf(s)

    // if a pointer to a struct is passed
    // in, handle it appropriately
    if r.Kind() == reflect.Ptr {
        r = r.Elem()
        value = value.Elem()
    }

    // loop over all of the fields
    for i := 0; i < r.NumField(); i++ {
        field := r.Field(i)
        // struct tag found
        key := field.Name
        if serialize, ok :=
field.Tag.Lookup("serialize"); ok {
            // ignore "-" otherwise that whole
value
            // becomes the serialize 'key'
            if serialize == "-" {
                continue
            }
            key = serialize
        }

        switch value.Field(i).Kind() {
            // this recipe only supports strings!
            case reflect.String:
                result += key + ":" +
value.Field(i).String() + ";"
                // by default skip it
            default:
                continue
        }
    }
}

```

```
    }
    return result, nil
}
```

- Создайте файл с именем `deserialize.go` со следующим содержимым:

```
package tags

import (
    "errors"
    "reflect"
    "strings"
)

// DeSerializeStructStrings converts a serialized
// string using our custom serialization format
// to a struct
func DeSerializeStructStrings(s string, res
interface{}) error
{
    r := reflect.TypeOf(res)

    // we're setting using a pointer so
    // it must always be a pointer passed
    // in
    if r.Kind() != reflect.Ptr {
        return errors.New("res must be a pointer")
    }

    // dereference the pointer
    r = r.Elem()
    value := reflect.ValueOf(res).Elem()

    // split our serialization string into
    // a map
    vals := strings.Split(s, ";")
    valMap := make(map[string]string)
    for _, v := range vals {
        keyval := strings.Split(v, ":")
        if len(keyval) != 2 {
            continue
        }
    }
}
```

```

        valMap[keyval[0]] = keyval[1]
    }

    // iterate over fields
    for i := 0; i < r.NumField(); i++ {
        field := r.Field(i)

        // check if in the serialize set
        if serialize, ok :=
field.Tag.Lookup("serialize"); ok {
            // ignore "-" otherwise that whole
value
            // becomes the serialize 'key'
            if serialize == "-" {
                continue
            }
            // is it in the map
            if val, ok := valMap[serialize]; ok {
                value.Field(i).SetString(val)
            }
        } else if val, ok := valMap[field.Name]; ok
{
            // is our field name in the map
instead?
            value.Field(i).SetString(val)
        }
    }
    return nil
}

```

- Создайте файл с именем **tags.go** со следующим содержимым:

```

package tags

import "fmt"

// Person is a struct that stores a persons
// name, city, state, and a misc attribute
type Person struct {
    Name string `serialize:"name"`
    City string `serialize:"city"`
    State string
    Misc string `serialize:"- "`
}

```

```

        Year int `serialize:"year"`
    }

    // EmptyStruct demonstrates serialize
    // and deserialize for an Empty struct
    // with tags
    func EmptyStruct() error {
        p := Person{}

        res, err := SerializeStructStrings(&p)
        if err != nil {
            return err
        }
        fmt.Printf("Empty struct: %#v\n", p)
        fmt.Println("Serialize Results:", res)

        newP := Person{}
        if err := DeSerializeStructStrings(res,
&newP); err != nil
        {
            return err
        }
        fmt.Printf("Deserialize results: %#v\n", newP)
        return nil
    }

    // FullStruct demonstrates serialize
    // and deserialize for an Full struct
    // with tags
    func FullStruct() error {
        p := Person{
            Name: "Aaron",
            City: "Seattle",
            State: "WA",
            Misc: "some fact",
            Year: 2017,
        }
        res, err := SerializeStructStrings(&p)
        if err != nil {
            return err
        }
        fmt.Printf("Full struct: %#v\n", p)
        fmt.Println("Serialize Results:", res)
    }

```

```

        newP := Person{}
        if err := DeSerializeStructStrings(res,
&newP);
        err != nil {
            return err
        }
        fmt.Printf("Deserialize results: %#v\n",
newP)
        return nil
    }

```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "fmt"

    "github.com/PacktPublishing/
Go-Programming-Cookbook-Second-Edition/
chapter3/tags"
)

func main() {

    if err := tags.EmptyStruct(); err != nil {
        panic(err)
    }

    fmt.Println()

    if err := tags.FullStruct(); err != nil {
        panic(err)
    }
}

```

- Выполните `go run main.go`. Вы также можете запустить следующее:

```

$ go build
$ ./example

```

Вы должны увидеть следующий вывод:

```
$ go run main.go
Empty struct: tags.Person{Name:"", City:"", State:"",
Misc:"",
Year:0}
Serialize Results: name;;city;;State;;
Deserialize results: tags.Person{Name:"", City:"",
State:"",
Misc:"", Year:0}

Full struct: tags.Person{Name:"Aaron", City:"Seattle",
State:"WA", Misc:"some fact", Year:2017}
Serialize Results: name:Aaron;city:Seattle;State:WA;
Deserialize results: tags.Person{Name:"Aaron",
City:"Seattle",
State:"WA", Misc:"", Year:0}
```

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Этот рецепт создает формат сериализации строк, который принимает значение `struct` и сериализует все строковые поля в анализируемый формат. Этот рецепт не работает с некоторыми пограничными случаями; в частности, строки не должны содержать символы двоеточия (:) или точки с запятой (;). Вот краткое изложение его поведения:

- Если поле является строкой, оно будет сериализовано/десериализовано.
- Если поле не является строкой, оно будет проигнорировано.
- Если тег `struct` поля содержит "ключ" сериализации, то ключ будет возвращаемой сериализованной/десериализованной средой.
- Дубликаты не обрабатываются.
- Если тег `struct` не указан, вместо него используется имя поля.
- Если значением тега структуры является дефис (-), поле игнорируется, даже если это строка.

Следует также отметить, что рефлексия не полностью работает с неэкспортируемыми значениями.

## Реализация коллекций через замыкания

Если вы работали с функциональными или динамическими языками программирования, вам может показаться, что циклы `for` и операторы `if` создают многословный код. Использование функциональных конструкций, таких как `map` и `filter`, для обработки списков может быть полезным и сделать код более читабельным; однако в Go этих типов нет в стандартной библиотеке, и их может быть трудно обобщить без обобщений или очень сложного отражения и использования пустых интерфейсов. Этот рецепт предоставит вам несколько основных примеров реализации коллекций с использованием замыканий Go.

### Как это сделать...

Следующие шаги описывают, как написать и запустить ваше приложение:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter3/collections`.
- Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter3/collections
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter3/collections
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter3/collections` или используйте это как упражнение для написания собственного кода!



- Создайте файл `collections.go` со следующим содержимым:

```
package collections

// WorkWith is the struct we'll
// be implementing collections for
type WorkWith struct {
    Data    string
    Version int
}

// Filter is a functional filter. It takes a list
of
// WorkWith and a WorkWith Function that returns a
bool
// for each "true" element we return it to the
resultant
// list
func Filter(ws []WorkWith, f func(w WorkWith)
bool) []WorkWith
{
    // depending on results, smallest size for
result
    // is len == 0
    result := make([]WorkWith, 0)
    for _, w := range ws {
        if f(w) {
            result = append(result, w)
        }
    }
    return result
}

// Map is a functional map. It takes a list of
// WorkWith and a WorkWith Function that takes a
WorkWith
// and returns a modified WorkWith. The end result
is
// a list of modified WorkWiths
func Map(ws []WorkWith, f func(w WorkWith)
WorkWith) []WorkWith
{
    // the result should always be the same
```

```

    // length
    result := make([]WorkWith, len(ws))

    for pos, w := range ws {
        newW := f(w)
        result[pos] = newW
    }
    return result
}

```

- Создайте файл `functions.go` со следующим содержимым:

```

package collections

import "strings"

// LowerCaseData does a ToLower to the
// Data string of a WorkWith
func LowerCaseData(w WorkWith) WorkWith {
    w.Data = strings.ToLower(w.Data)
    return w
}

// IncrementVersion increments a WorkWiths
// Version
func IncrementVersion(w WorkWith) WorkWith {
    w.Version++
    return w
}

// OldVersion returns a closures
// that validates the version is greater than
// the specified amount
func OldVersion(v int) func(w WorkWith) bool {
    return func(w WorkWith) bool {
        return w.Version >= v
    }
}

```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

```

```

import (
    "fmt"

    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter3/collections"
)

func main() {
    ws := []collections.WorkWith{
        collections.WorkWith{"Example", 1},
        collections.WorkWith{"Example 2", 2},
    }

    fmt.Printf("Initial list: %#v\n", ws)

    // first lower case the list
    ws = collections.Map(ws,
collections.LowerCaseData)
    fmt.Printf("After LowerCaseData Map: %#v\n",
ws)

    // next increment all versions
    ws = collections.Map(ws,
collections.IncrementVersion)
    fmt.Printf("After IncrementVersion Map:
%#v\n", ws)

    // lastly remove all versions older than 3
    ws = collections.Filter(ws,
collections.OldVersion(3))
    fmt.Printf("After OldVersion Filter: %#v\n",
ws)
}

```

- Выполните `go run main.go`. Вы также можете запустить следующее:

```

$ go build
$ ./example

```

Вы должны увидеть следующий вывод:

```

$ go run main.go
Initial list:
[]collections.WorkWith{collections.WorkWith{Data:"Example
",
Version:1}, collections.WorkWith{Data:"Example 2",
Version:2}}
After LowerCaseData Map:
[]collections.WorkWith{collections.WorkWith{Data:"example
",
Version:1}, collections.WorkWith{Data:"example 2",
Version:2}}
After IncrementVersion Map:
[]collections.WorkWith{collections.WorkWith{Data:"example
",
Version:2}, collections.WorkWith{Data:"example 2",
Version:3}}
After OldVersion Filter:
[]collections.WorkWith{collections.WorkWith{Data:"example
2",
Version:3}}

```

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Замыкания в Go очень эффективны. Хотя наши функции `collections` не являются универсальными, они относительно малы и могут быть легко применены к нашей структуре `WorkWith` с минимальным добавленным кодом с использованием различных функций. Глядя на это, вы можете заметить, что мы нигде не возвращаем ошибки. Идея этих функций в том, что они чисты: у исходного списка нет побочных эффектов, за исключением того, что мы решили перезаписывать его после каждого вызова.

Если вам нужно применить уровни модификации к списку или структуре списков, то этот шаблон может избавить вас от путаницы и сделать тестирование очень простым. Также возможно связать карты и фильтры вместе для очень выразительного стиля кодирования.

## 4. Обработка ошибок в Go

Обработка ошибок важна даже для самой простой программы Go. Ошибки в Go реализуют интерфейс `Error` и должны обрабатываться на каждом уровне кода. Ошибки Go не работают как исключения, а необработанные ошибки могут вызвать огромные проблемы. Вы должны стремиться обрабатывать и учитывать ошибки всякий раз, когда они происходят.

В этой главе также рассматривается ведение журнала, так как обычно журнал регистрируется всякий раз, когда возникает фактическая ошибка. Мы также исследуем ошибки переноса, чтобы данная ошибка предоставляла дополнительный контекст по мере того, как она возвращалась вверх по стеку функций, чтобы было легче определить фактическую причину определенных ошибок.

В этой главе будут рассмотрены следующие рецепты:

- Обработка ошибок и интерфейс ошибок
- Использование пакета `pkg/errors` и перенос ошибок
- Использование пакета `log` и понимание того, когда следует регистрировать ошибки
- Структурированное ведение журналов с помощью пакетов `apex` и `logrus`
- Ведение журнала с пакетом `context`
- Использование глобальных переменных уровня пакета
- Захват паники для долго выполняющихся процессов

## Технические требования

Чтобы продолжить выполнение всех рецептов в этой главе, настройте свою среду в соответствии со следующими шагами:

- Загрузите и установите Go 1.12.6 или более позднюю версию в своей операционной системе по адресу <https://golang.org/doc/install>.
- Откройте терминал или консольное приложение, создайте и перейдите в каталог проекта, например `~/projects/go-`

`programming-cookbook`. Весь наш код будет запускаться и изменяться из этого каталога.

- Скопируйте последний код в `~/projects/go-programming-cookbook-original` и, при желании, работайте из этого каталога, вместо того, чтобы вводить примеры вручную:

```
$ git clone git@github.com:PacktPublishing/Go-Programming-Cookbook-Second-Edition.git go-programming-cookbook-original
```

## Обработка ошибок и интерфейс ошибок

Интерфейс `Error` — довольно маленький и простой интерфейс:

```
type Error interface{
    Error() string
}
```

Этот интерфейс элегантен, потому что легко сделать что угодно, чтобы удовлетворить его. К сожалению, это также создает путаницу для пакетов, которым необходимо выполнять определенные действия в зависимости от полученной ошибки.

В Go есть несколько способов создать ошибку; этот рецепт исследует создание основных ошибок, ошибок, которым присвоены значения или типы, и пользовательской ошибки с использованием структуры.

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter4/basicerrors` and Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter4/basicerrors
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter4/basicerrors
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter4/basicerrors` или используйте это как упражнение для написания собственного кода! или используйте это как упражнение для написания собственного кода!
- Создайте файл с именем `basicerrors.go` со следующим содержимым:

```
package basicerrors

import (
    "errors"
    "fmt"
)

// ErrorValue is a way to make a package level
// error to check against. I.e. if err == ErrorValue
var ErrorValue = errors.New("this is a typed error")

// TypedError is a way to make an error type
// you can do err.(type) == ErrorValue
type TypedError struct {
    error
}

//BasicErrors demonstrates some ways to create errors
func BasicErrors() {
    err := errors.New("this is a quick and easy way to
create an error")
    fmt.Println("errors.New: ", err)

    err = fmt.Errorf("an error occurred: %s", "something")
    fmt.Println("fmt.Errorf: ", err)

    err = ErrorValue
    fmt.Println("value error: ", err)
```

```
err = TypedError{errors.New("typed error")}
fmt.Println("typed error: ", err)
```

```
}
```

- Создайте файл `custom.go` со следующим содержимым:

```
package basicerrors
```

```
import (
    "fmt"
)
```

```
// CustomError is a struct that will implement
// the Error() interface
type CustomError struct {
    Result string
}
```

```
func (c CustomError) Error() string {
    return fmt.Sprintf("there was an error; %s was the
result", c.Result)
}
```

```
// SomeFunc returns an error
func SomeFunc() error {
    c := CustomError{Result: "this"}
    return c
}
```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```
package main
```

```
import (
    "fmt"
```

```
    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter4/basicerrors"
)
```



```
func main() {
    basicerrors.BasicErrors()

    err := basicerrors.SomeFunc()
    fmt.Println("custom error: ", err)
}
```

- Выполните `go run main.go`.
- Вы также можете запустить следующие команды:

```
$ go build
$ ./example
```

Вы также можете запустить следующие команды:

```
$ go run main.go
errors.New: this is a quick and easy way to create an
error
fmt.Errorf: an error occurred: something
typed error: this is a typed error
custom error: there was an error; this was the result
```

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Независимо от того, используете ли вы `error.New`, `fmt.Errorf` или пользовательскую ошибку, важно помнить, что вы никогда не должны оставлять ошибки в своем коде необработанными. Эти различные методы определения ошибок дают большую гибкость. Вы можете, например, поместить дополнительные функции в свою структуру для дальнейшего опроса ошибки и привести интерфейс к вашему типу ошибки в вызывающей функции, чтобы получить некоторые дополнительные функции.

Сам интерфейс очень прост, и единственное требование состоит в том, чтобы вы возвращали допустимую строку. Соединение этого со структурой может быть полезно для некоторых высокоуровневых приложений, которые имеют единую обработку ошибок во всем, но хотят хорошо работать с другими приложениями.

# Использование пакета `pkg/errors` и перенос ошибок

Пакет `errors`, расположенный на [github.com/pkg/errors](https://github.com/pkg/errors), является заменой стандартного пакета `errors` Go. Кроме того, он предоставляет некоторые очень полезные функции для переноса и обработки ошибок. Хорошим примером являются типизированные и объявленные ошибки в предыдущем рецепте — они могут быть полезны для добавления дополнительной информации к ошибке, но ее стандартное обертывание изменит ее тип и нарушит утверждение типа:

```
// this wont work if you wrapped it
// in a standard way. that is,
// fmt.Errorf("custom error: %s", err.Error())
if err == Package.ErrorNamed{
    //handle this error in a specific way
}
```

Этот рецепт продемонстрирует, как использовать пакет `pkg/errors` для добавления аннотаций к ошибкам в вашем коде.

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter4/errwrap` and Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter4/errwrap
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter4/errwrap
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter4/errwrap` или используйте это как упражнение

для написания собственного кода!

- Создайте файл с именем `errwrap.go` со следующим содержимым:

```
package errwrap

import (
    "fmt"

    "github.com/pkg/errors"
)

// WrappedError demonstrates error wrapping and
// annotating an error
func WrappedError(e error) error {
    return errors.Wrap(e, "An error occurred in
WrappedError")
}

// ErrorTyped is a error we can check against
type ErrorTyped struct{
    error
}

// Wrap shows what happens when we wrap an error
func Wrap() {
    e := errors.New("standard error")

    fmt.Println("Regular Error - ",
WrappedError(e))

    fmt.Println("Typed Error - ",
WrappedError(ErrorTyped{errors.New("typed
error")})))

    fmt.Println("Nil -", WrappedError(nil))
}
```

- Создайте файл `unwrap.go` со следующим содержимым:

```
package errwrap

import (
```

```

    "fmt"

    "github.com/pkg/errors"
)

// Unwrap will unwrap an error and do
// type assertion to it
func Unwrap() {
    err := error(ErrorTyped{errors.New("an error
occurred")})
    err = errors.Wrap(err, "wrapped")

    fmt.Println("wrapped error: ", err)

    // we can handle many error types
    switch errors.Cause(err).(type) {
    case ErrorTyped:
        fmt.Println("a typed error occurred: ",
err)

        default:
            fmt.Println("an unknown error occurred")
        }
    }

    // StackTrace will print all the stack for
    // the error
    func StackTrace() {
        err := error(ErrorTyped{errors.New("an error
occurred")})
        err = errors.Wrap(err, "wrapped")

        fmt.Printf("%+v\n", err)
    }
}

```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "fmt"

```

```

        "github.com/PacktPublishing/
        Go-Programming-Cookbook-Second-Edition/
        chapter4/errwrap"
    )

    func main() {
        errwrap.Wrap()
        fmt.Println()
        errwrap.Unwrap()
        fmt.Println()
        errwrap.StackTrace()
    }

```

- Выполните `go run main.go`.
- Вы также можете запустить следующие команды:

```

$ go build
$ ./example

```

Вы также можете запустить следующие команды:

```

$ go run main.go
Regular Error - An error occurred in WrappedError:
standard
error
Typed Error - An error occurred in WrappedError: typed
error
Nil - <nil>

```

```

wrapped error: wrapped: an error occurred
a typed error occurred: wrapped: an error occurred

```

```

an error occurred
github.com/PacktPublishing/Go-Programming-Cookbook-
Second-
Edition/chapter4/errwrap.StackTrace
/Users/lothamer/go/src/github.com/agtorre/go-
cookbook/chapter4/errwrap/unwrap.go:30
main.main
/tmp/go/src/github.com/agtorre/go-
cookbook/chapter4/errwrap/example/main.go:14

```

- Файл `go.mod` должен быть обновлен, а файл `go.sum` теперь должен присутствовать в каталоге рецептов верхнего уровня.

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Пакет `pkg/errors` — очень полезный инструмент. Имеет смысл оборачивать каждую возвращенную ошибку с помощью этого пакета, чтобы обеспечить дополнительный контекст при ведении журнала и отладке ошибок. Это достаточно гибко, чтобы распечатать всю трассировку стека при возникновении ошибки или просто добавить префикс к вашим ошибкам при их печати. Он также может очищать код, поскольку обернутый `nil` возвращает значение `nil`; например, рассмотрим следующий код:

```
func RetError() error{
    err := ThisReturnsAnError()
    return errors.Wrap(err, "This only does something if err
    != nil")
}
```

В некоторых случаях это может избавить вас от необходимости сначала проверять, является ли ошибка `nil`, прежде чем просто вернуть ее. Этот рецепт демонстрирует, как использовать пакет для переноса и распаковки ошибок, а также базовые функции трассировки стека. Документация к пакету также предоставляет некоторые другие полезные примеры, такие как печать частичных стеков. Дэйв Чейни, автор этой библиотеки, также написал несколько полезных блогов и выступил с докладами на эту тему; вы можете перейти на <https://dave.cheney.net/2016/04/27/dont-just-check-errors-handle-them-gracefully>, чтобы узнать больше.

## Использование пакета журнала и понимание того, когда следует регистрировать ошибки

Регистрация обычно должна происходить, когда конечным результатом является ошибка. Другими словами, полезно вести журнал, когда

происходит что-то исключительное или неожиданное. Также может быть уместно, если вы используете журнал, который предоставляет уровни журнала, для добавления операторов отладки или информации в ключевые части вашего кода, чтобы быстро устранять проблемы во время разработки. Слишком большое количество журналов затруднит поиск чего-либо полезного, но недостаточное количество журналов может привести к поломке системы без понимания основной причины. Этот рецепт продемонстрирует использование стандартного пакета `log` Go и некоторых полезных опций, а также продемонстрирует, когда, вероятно, должен появиться журнал.

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter4/log` and Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-  
Cookbook-Second-Edition/chapter4/log
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-  
Cookbook-Second-Edition/chapter4/log
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter4/log` или используйте это как упражнение для написания собственного кода!
- Создайте файл с именем `log.go` со следующим содержимым:

```
package log  
  
import (  
    "bytes"  
    "fmt"  
    "log"  
)
```

```

// Log uses the setup logger
func Log() {
    // we'll configure the logger to write
    // to a bytes.Buffer
    buf := bytes.Buffer{}

    // second argument is the prefix last argument
    // options you combine them with a logical or.
    logger := log.New(&buf, "logger: ",
        log.Lshortfile|log.Ldate)

    logger.Println("test")

    logger.SetPrefix("new logger: ")

    logger.Printf("you can also add args(%v) and
use Fatalln to
log and crash", true)

    fmt.Println(buf.String())
}

```

- Создайте файл с именем `error.go` со следующим содержимым:

```

package log

import "github.com/pkg/errors"
import "log"

// OriginalError returns the error original error
func OriginalError() error {
    return errors.New("error occurred")
}

// PassThroughError calls OriginalError and
// forwards the error along after wrapping.
func PassThroughError() error {
    err := OriginalError()
    // no need to check error
    // since this works with nil
    return errors.Wrap(err, "in passthrougherror")
}

```



```

    // FinalDestination deals with the error
    // and doesn't forward it
    func FinalDestination() {
        err := PassThroughError()
        if err != nil {
            // we log because an unexpected error
occurred!
            log.Printf("an error occurred: %s\n",
err.Error())
            return
        }
    }

```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "fmt"

    "github.com/PacktPublishing/
Go-Programming-Cookbook-Second-Edition/
chapter4/log"
)

func main() {
    fmt.Println("basic logging and modification of
logger:")
    log.Log()
    fmt.Println("logging 'handled' errors:")
    log.FinalDestination()
}

```

- Выполните `go run main.go`.
- Вы также можете запустить следующие команды:

```

$ go build
$ ./example

```

Вы также можете запустить следующие команды:

```
$ go run main.go
basic logging and modification of logger:
logger: 2017/02/05 log.go:19: test
new logger: 2017/02/05 log.go:23: you can also add
args(true)
and use Fataln to log and crash
```

```
logging 'handled' errors:
2017/02/05 18:36:11 an error occurred: in
passthrougherror:
error occurred
```

- Файл `go.mod` обновляется, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Вы можете либо инициализировать регистратор и передавать его с помощью `log.NewLogger()`, либо использовать регистратор уровня пакета журнала для регистрации сообщений. Файл `log.go` в этом рецепте делает первое, а `errlog.go` — второе. Он также показывает, когда запись в журнал может иметь смысл после того, как ошибка достигла конечного пункта назначения; в противном случае вполне вероятно, что вы будете регистрироваться несколько раз для одного события.

Есть несколько проблем с этим подходом. Во-первых, у вас может быть дополнительный контекст в одной из промежуточных функций, например, переменные, которые вы хотите регистрировать. Во-вторых, регистрация множества переменных может привести к путанице, сделать ее запутанной и трудной для чтения. В следующем рецепте исследуется структурированное ведение журналов, которое обеспечивает гибкость в регистрации переменных, а в следующем рецепте мы также рассмотрим реализацию глобального регистратора на уровне пакета.

# Структурированное ведение журналов с помощью пакетов `арех` и `logrus`

Основной причиной записи информации в журнал является проверка состояния системы, когда события происходят или произошли в прошлом. Основные сообщения журнала сложно прочесть, когда у вас есть большое количество микросервисов, которые ведут журнал.

Существует множество сторонних пакетов для просмотра журналов, если вы можете перевести журналы в формат данных, который они понимают. Эти пакеты обеспечивают функции индексирования, возможности поиска и многое другое. Пакеты `sirupsen/logrus` и `арех/log` предоставляют способ ведения структурированного журнала, в котором вы можете регистрировать ряд полей, которые можно переформатировать, чтобы они соответствовали этим сторонним программам чтения журналов. Например, можно просто создавать журналы в формате JSON для анализа различными службами.

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter4/structured` and Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter4/structured
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter4/structured
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter4/structured` или используйте это как

- упражнение для написания собственного кода!
- Создайте файл `logrus.go` со следующим содержимым:

```
package structured

import "github.com/sirupsen/logrus"

// Hook will implement the logrus
// hook interface
type Hook struct {
    id string
}

// Fire will trigger whenever you log
func (hook *Hook) Fire(entry *logrus.Entry) error
{
    entry.Data["id"] = hook.id
    return nil
}

// Levels is what levels this hook will fire on
func (hook *Hook) Levels() []logrus.Level {
    return logrus.AllLevels
}

// Logrus demonstrates some basic logrus
functionality
func Logrus() {
    // we're emitting in json format
    logrus.SetFormatter(&logrus.TextFormatter{})
    logrus.SetLevel(logrus.InfoLevel)
    logrus.AddHook(&Hook{"123"})

    fields := logrus.Fields{}
    fields["success"] = true
    fields["complex_struct"] = struct {
        Event string
        When string
    }{"Something happened", "Just now"}

    x := logrus.WithFields(fields)
    x.Warn("warning!")
}
```

```
        x.Error("error!")
    }
}
```

- Создайте файл с именем **apex.go** со следующим содержимым:

```
package structured

import (
    "errors"
    "os"

    "github.com/apex/log"
    "github.com/apex/log/handlers/text"
)

// ThrowError throws an error that we'll trace
func ThrowError() error {
    err := errors.New("a crazy failure")
    log.WithField("id",
"123").Trace("ThrowError").Stop(&err)
    return err
}

// CustomHandler splits to two streams
type CustomHandler struct {
    id string
    handler log.Handler
}

// HandleLog adds a hook and does the emitting
func (h *CustomHandler) HandleLog(e *log.Entry)
error {
    e.WithField("id", h.id)
    return h.handler.HandleLog(e)
}

// Apex has a number of useful tricks
func Apex() {
    log.SetHandler(&CustomHandler{"123",
text.New(os.Stdout)})
    err := ThrowError()

    //With error convenience function
```

```
        log.WithError(err).Error("an error occurred")
    }
}
```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```
package main

import (
    "fmt"

    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter4/structured"
)

func main() {
    fmt.Println("Logrus:")
    structured.Logrus()

    fmt.Println()
    fmt.Println("Apex:")
    structured.Apex()
}
```

- Выполните `go run main.go`.
- Вы также можете запустить следующие команды:

```
$ go build
$ ./example
```

Теперь вы должны увидеть следующий вывод:

```
$ go run main.go
Logrus:
WARN[0000] warning! complex_struct={Something happened
Just now}
id=123 success=true
ERRO[0000] error! complex_struct={Something happened Just
now}
id=123 success=true

Apex:
INFO[0000] ThrowError id=123
```

```
ERROR[0000] ThrowError duration=133ns error=a crazy
failure
id=123
ERROR[0000] an error occurred error=a crazy failure
```

- Файл `go.mod` должен быть обновлен, а файл `go.sum` теперь должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Пакеты `sirupsen/logrus` и `apex/log` являются отличными структурированными регистраторами. Оба предоставляют хуки либо для отправки нескольких событий, либо для добавления дополнительных полей в запись журнала. Например, было бы относительно просто использовать хук `logrus` или настраиваемый обработчик `apex` для добавления номеров строк ко всем вашим журналам, а также имен служб. Другое использование ловушки может включать в себя `traceID` для отслеживания запроса в разных службах.

В то время как `logrus` разделяет хук и формater, `apex` объединяет их. В дополнение к этому в `apex` добавлены некоторые удобные функции, такие как `WithError` для добавления поля `error`, а также трассировка, обе из которых демонстрируются в этом рецепте. Также относительно просто адаптировать хуки от `logrus` к обработчикам `apex`. Для обоих решений было бы простым изменением преобразовать в форматирование JSON вместо текста, окрашенного в ANSI.

## Ведение журнала с пакетом context

Этот рецепт продемонстрирует способ передачи полей журнала между различными функциями. Пакет Go `pkg/context` — отличный способ передачи дополнительных переменных и отмен между функциями. Этот рецепт исследует использование этой функциональности для распределения переменных между функциями для целей ведения журнала.

Этот стиль можно адаптировать к `logrus` или `apex` из предыдущего рецепта. Мы будем использовать вершину для этого рецепта.

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter4/context` and Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter4/context
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter4/context
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter4/context` или используйте это как упражнение для написания собственного кода!
- Создайте файл с именем `log.go` со следующим содержимым:

```
package context

import (
    "context"

    "github.com/apex/log"
)

type key int

// logFields is a key we use
// for our context logging
const logFields key = 0

func getFields(ctx context.Context) *log.Fields {
    fields, ok := ctx.Value(logFields).
(*log.Fields)
```



```

        if !ok {
            f := make(log.Fields)
            fields = &f
        }
        return fields
    }

    // FromContext takes an entry and a context
    // then returns an entry populated from the
context object
    func FromContext(ctx context.Context, l
log.Interface)
        (context.Context, *log.Entry) {
            fields := getFields(ctx)
            e := l.WithFields(fields)
            ctx = context.WithValue(ctx, logFields,
fields)
            return ctx, e
        }

    // WithField adds a log field to the context
value
    func WithField(ctx context.Context, key string,
value
        interface{}) context.Context {
            return WithFields(ctx, log.Fields{key:
value})
        }

    // WithFields adds many log fields to the context
log.Fielder)
    func WithFields(ctx context.Context, fields
        context.Context {
            f := getFields(ctx)
            for key, val := range fields.Fields() {
                (*f)[key] = val
            }
            ctx = context.WithValue(ctx, logFields, f)
            return ctx
        }

```

- Создайте файл с именем `collect.go` со следующим содержимым:

```

package context

import (
    "context"
    "os"

    "github.com/apex/log"
    "github.com/apex/log/handlers/text"
)

// Initialize calls 3 functions to set up, then
// logs before terminating
func Initialize() {
    // set basic log up
    log.SetHandler(text.New(os.Stdout))
    // initialize our context
    ctx := context.Background()
    // create a logger and link it to
    // the context
    ctx, e := FromContext(ctx, log.Log)

    // set a field
    ctx = WithField(ctx, "id", "123")
    e.Info("starting")
    gatherName(ctx)
    e.Info("after gatherName")
    gatherLocation(ctx)
    e.Info("after gatherLocation")
}

func gatherName(ctx context.Context) {
    ctx = WithField(ctx, "name", "Go Cookbook")
}

func gatherLocation(ctx context.Context) {
    ctx = WithFields(ctx, log.Fields{"city":
"Seattle",
    "state": "WA"})
}

```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```
package main

import "github.com/PacktPublishing/
        Go-Programming-Cookbook-Second-Edition/
        chapter4/context"

func main() {
    context.Initialize()
}
```

- Выполните `go run main.go`.
- Вы также можете запустить следующие команды:

```
$ go build
$ ./example
```

Вы должны увидеть следующий вывод:

```
$ go run main.go
INFO[0000] starting id=123
INFO[0000] after gatherName id=123 name=Go Cookbook
INFO[0000] after gatherLocation city=Seattle id=123
name=Go
Cookbook state=WA
```

- Файл `go.mod` обновляется, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Пакет `context` теперь появляется в различных пакетах, включая пакеты базы данных и HTTP. Этот рецепт позволит вам прикрепить поля журнала к контексту и использовать их для целей ведения журнала. Идея состоит в том, что отдельные методы могут присоединять больше полей к контексту по мере его передачи, а затем конечный сайт вызова может вести журнал и агрегировать переменные.

Этот рецепт имитирует методы `WithField` и `WithFields`, обнаруженные в пакетах протоколирования в предыдущем рецепте.

Они изменяют одно значение, хранящееся в контексте, а также обеспечивают другие преимущества использования контекста: отмену, таймауты и безопасность потоков.

## Использование глобальных переменных уровня пакета

Пакеты `apex` и `logrus` в предыдущих примерах использовали глобальную переменную уровня пакета. Иногда бывает полезно структурировать ваши библиотеки так, чтобы они поддерживали обе структуры с различными методами и функциями верхнего уровня, чтобы вы могли использовать их напрямую, не передавая друг другу.

Этот рецепт также демонстрирует использование `sync.Once`, чтобы гарантировать, что глобальное средство ведения журнала будет инициализировано только один раз. Его также можно обойти с помощью метода `Set`. Рецепт экспортирует только `WithField` и `Debug`, но вы можете представить себе экспорт каждого метода, прикрепленного к объекту `log`.

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter4/global` and Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter4/global
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter4/global
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter4/global` или используйте это как упражнение для написания собственного кода!

- Создайте файл с именем `global.go` со следующим содержимым:

```
package global

import (
    "errors"
    "os"
    "sync"

    "github.com/sirupsen/logrus"
)

// we make our global package level
// variable lower case
var (
    log *logrus.Logger
    initLog sync.Once
)

// Init sets up the logger initially
// if run multiple times, it returns
// an error
func Init() error {
    err := errors.New("already initialized")
    initLog.Do(func() {
        err = nil
        log = logrus.New()
        log.Formatter = &logrus.JSONFormatter{}
        log.Out = os.Stdout
        log.Level = logrus.DebugLevel
    })
    return err
}

// SetLog sets the log
func SetLog(l *logrus.Logger) {
    log = l
}

// WithField exports the logs withfield connected
// to our global log
func WithField(key string, value interface{})
*logrus.Entry {
```

```

        return log.WithField(key, value)
    }

    // Debug exports the logs Debug connected
    // to our global log
    func Debug(args ...interface{}) {
        log.Debug(args...)
    }

```

- Создайте файл с именем `log.go` со следующим содержимым:

```

package global

// UseLog demonstrates using our global
// log
func UseLog() error {
    if err := Init(); err != nil {
        return err
    }

    // if we were in another package these would be
    // global.WithField and
    // global.Debug
    WithField("key", "value").Debug("hello")
    Debug("test")

    return nil
}

```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import "github.com/PacktPublishing/
        Go-Programming-Cookbook-Second-Edition/
        chapter4/global"

func main() {
    if err := global.UseLog(); err != nil {
        panic(err)
    }
}

```

- Выполните `go run main.go`.
- Вы также можете запустить следующие команды:

```
$ go build
$ ./example
```

Теперь вы должны увидеть следующий вывод:

```
$ go run main.go
{"key":"value","level":"debug","msg":"hello","time":"2017-02-12T19:22:50-08:00"}
{"level":"debug","msg":"test","time":"2017-02-12T19:22:50-08:00"}
```

- Файл `go.mod` обновляется, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Обычный шаблон для этих `global` объектов уровня пакета заключается в том, чтобы не экспортировать `global` переменную и предоставлять только желаемую функциональность через методы. Как правило, вы также можете включить метод для возврата копии `global` средства ведения журнала для пакетов, которым требуется объект средства ведения журнала.

Тип `sync.Once` — это недавно введенная структура. Эта структура в сочетании с методом `Do` будет выполняться в коде только один раз. Мы используем это в нашем коде инициализации, и функция `Init` выдаст ошибку, если `Init` вызывается более одного раза. Мы используем пользовательскую функцию `Init` вместо встроенной функции `init()`, если мы хотим передать параметры в наш `global` журнал.

Хотя в этом примере используется журнал, вы также можете представить случаи, когда это может быть полезно с подключением к базе данных, потоками данных и рядом других вариантов использования.

# Отлов паники для долго выполняющихся процессов

При реализации длительных процессов некоторые пути кода могут привести к панике. Обычно это характерно для таких вещей, как неинициализированные карты и указатели, а также проблемы с делением на ноль в случае плохо проверенного пользовательского ввода.

Полный сбой программы в таких случаях часто намного хуже, чем сама паника, поэтому может быть полезно поймать и справиться с паникой.

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter4/panic` and Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter4/panic
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter4/panic
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter4/panic` или используйте это как упражнение для написания собственного кода!
- Создайте файл с именем `panic.go` со следующим содержимым:

```
package panic

import (
    "fmt"
    "strconv"
```



```

)

// Panic panics with a divide by zero
func Panic() {
    zero, err := strconv.ParseInt("0", 10, 64)
    if err != nil {
        panic(err)
    }

    a := 1 / zero
    fmt.Println("we'll never get here", a)
}

// Catcher calls Panic
func Catcher() {
    defer func() {
        if r := recover(); r != nil {
            fmt.Println("panic occurred:", r)
        }
    }()
    Panic()
}

```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "fmt"

    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter4/panic"
)

func main() {
    fmt.Println("before panic")
    panic.Catcher()
    fmt.Println("after panic")
}

```

- Выполните `go run main.go`.
- Вы также можете запустить следующие команды:

```
$ go build  
$ ./example
```

Вы должны увидеть следующий вывод:

```
$ go run main.go  
before panic  
panic occurred: runtime error: integer divide by zero  
after panic
```

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Этот рецепт является очень простым примером того, как поймать панику. С помощью более сложного промежуточного программного обеспечения вы можете представить, как можно отложить восстановление и перехватить его после выполнения множества вложенных функций. Во время восстановления вы можете, по сути, делать все, что хотите, хотя создание журнала является обычным явлением.

В большинстве веб-приложений принято перехватывать паники и выдавать сообщение `http.StatusInternalServerError` при возникновении паники.

## 5. Сетевое программирование

Стандартная библиотека Go обеспечивает большую поддержку сетевых операций. Он включает в себя пакеты, позволяющие управлять TCP/IP, UDP, DNS, почтой и RPC с помощью HTTP. Сторонние пакеты также могут заполнить пробелы в стандартной библиотеке, включая [gorilla/websockets](https://github.com/gorilla/websocket/) (<https://github.com/gorilla/websocket/>) для реализации `WebSocket`, которую можно использовать в обычном обработчике HTTP. В этой главе рассматриваются эти библиотеки и демонстрируются некоторые простые рецепты использования каждой из них. Эти рецепты помогут разработчикам, которые не могут использовать абстракции более высокого уровня, такие как REST или GRPC, но нуждаются в сетевом подключении. Это также полезно для приложений DevOps, которым необходимо выполнять поиск DNS или работать с необработанными электронными письмами. Прочитав эту главу, вы должны были немного освоить базовое сетевое программирование и быть готовым к более глубокому погружению.

В этой главе будут рассмотрены следующие рецепты:

- Написание эхо-сервера и клиента TCP/IP
- Написание UDP-сервера и клиента
- Работа с разрешением доменного имени
- Работа с веб-сокетами
- Работа с `net/rpc` для вызова удаленных методов
- Использование `net/mail` для разбора писем

### Технические требования

Чтобы продолжить выполнение всех рецептов в этой главе, настройте свою среду в соответствии со следующими шагами:

- Загрузите и установите Go 1.12.6 или более позднюю версию в своей операционной системе по адресу <https://golang.org/doc/install>.

- Откройте терминал или консольное приложение, создайте и перейдите в каталог проекта, например `~/projects/go-programming-cookbook`. Весь наш код будет запускаться и изменяться из этого каталога.
- Скопируйте последний код в `~/projects/go-programming-cookbook-original` и, при желании, работайте из этого каталога, вместо того, чтобы вводить примеры вручную:

```
$ git clone git@github.com:PacktPublishing/Go-Programming-Cookbook-Second-Edition.git go-programming-cookbook-original
```

## Написание эхо-сервера и клиента TCP/IP

TCP/IP — это распространенный сетевой протокол, а протокол HTTP был построен на его основе. TCP требует, чтобы клиент подключался к серверу для отправки и получения данных. Этот рецепт будет использовать пакет `net` для создания TCP-соединения между клиентом и сервером. Клиент отправит пользовательский ввод на сервер, и сервер ответит той же введенной строкой, но преобразованной в верхний регистр с использованием результатов `strings.ToUpper()`. Клиент будет печатать любые сообщения, полученные от сервера, поэтому он должен выводить версию нашего ввода в верхнем регистре.

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- From your Terminal or console application, create a new directory called `~/projects/go-programming-cookbook/chapter5/tcp` and Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter5/tcp
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

`module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter5/tcp`

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter5/tcp` или используйте это как упражнение для написания собственного кода!
- Create a new directory named `server` and navigate to it.
- Создайте файл с именем `main.go` со следующим содержимым:

```
package main

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

const addr = "localhost:8888"

func echoBackCapitalized(conn net.Conn) {
    // set up a reader on conn (an io.Reader)
    reader := bufio.NewReader(conn)

    // grab the first line of data encountered
    data, err := reader.ReadString('\n')
    if err != nil {
        fmt.Printf("error reading data: %s\n", err.Error())
        return
    }
    // print then send back the data
    fmt.Printf("Received: %s", data)
    conn.Write([]byte(strings.ToUpper(data)))
    // close up the finished connection
    conn.Close()
}

func main() {
    ln, err := net.Listen("tcp", addr)
    if err != nil {
        panic(err)
    }
    defer ln.Close()
```

```

    fmt.Printf("listening on: %s\n", addr)
    for {
        conn, err := ln.Accept()
        if err != nil {
            fmt.Printf("encountered an error accepting
connection: %s\n",
                        err.Error())
            // if there's an error try again
            continue
        }
        // handle this asynchronously
        // potentially a good use-case
        // for a worker pool
        go echoBackCapitalized(conn)
    }
}

```

- Перейдите к предыдущему каталогу.
- Создайте новый каталог с именем `client` и перейдите в него.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

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

const addr = "localhost:8888"

func main() {
    reader := bufio.NewReader(os.Stdin)
    for {
        // grab a string input from the client
        fmt.Printf("Enter some text: ")
        data, err := reader.ReadString('\n')
        if err != nil {
            fmt.Printf("encountered an error reading input:
%s\n",
                        err.Error())
            continue
        }
    }
}

```

```

    }
    // connect to the addr
    conn, err := net.Dial("tcp", addr)
    if err != nil {
        fmt.Printf("encountered an error connecting: %s\n",
            err.Error())
    }

    // write the data to the connection
    fmt.Fprintf(conn, data)

    // read back the response
    status, err := bufio.NewReader(conn).ReadString('\n')
    if err != nil {
        fmt.Printf("encountered an error reading response:
%s\n",
            err.Error())
    }
    fmt.Printf("Received back: %s", status)
    // close up the finished connection
    conn.Close()
}
}

```

- Перейдите к предыдущему каталогу.
- Запустите `go run ./server` и вы увидите следующий вывод:

```

$ go run ./server
listening on: localhost:8888

```

- В отдельном терминале запустите `go run ./client` из каталога `tcp`, и вы увидите следующий вывод:

```

$ go run ./client
Enter some text:

```

- Введите `this is a test` и нажмите *Enter*. Вы увидите следующее:

```

$ go run ./client
Enter some text: this is a test
Received back: THIS IS A TEST
Enter some text:

```

- Нажмите *Ctrl + C*, чтобы выйти.

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Сервер прослушивает порт `8888`. Всякий раз, когда приходит запрос, сервер должен принять запрос и управлять клиентским соединением. В случае с этой программой она отправляет горутину, которая читает запрос от клиента, использует полученные данные с заглавной буквы, отправляет их обратно клиенту и, наконец, закрывает соединение. Сервер сразу же снова зацикливается, ожидая получения новых клиентских подключений, в то время как предыдущее подключение обрабатывается отдельно.

Клиент считывает ввод из `STDIN`, подключается к адресу через соединение TCP, записывает сообщение, которое было прочитано из ввода, а затем распечатывает ответ от сервера. После этого он закрывает соединение и снова зацикливает чтение из `STDIN`. Вы также можете переработать этот пример, чтобы клиент оставался подключенным до выхода из программы, а не при каждом запросе.

## Написание UDP-сервера и клиента

Протокол UDP часто используется для игр и в местах, где скорость важнее надежности. UDP-серверы и клиенты не должны соединяться друг с другом. Этот рецепт создаст UDP-сервер, который будет прослушивать сообщения от клиентов, добавлять их IP-адреса в свой список и рассылать сообщения каждому из ранее замеченных клиентов.

Сервер будет писать сообщение в `STDOUT` всякий раз, когда клиент подключается, и он будет транслировать одно и то же сообщение всем своим клиентам. Текст этого сообщения должен быть `Sent <count>`, где `<count>` будет увеличиваться каждый раз, когда сервер выполняет широковещательную рассылку всем своим клиентам. В результате `count` может иметь разные значения в зависимости от того, сколько времени вам потребуется для подключения к вашему клиенту,



поскольку сервер будет транслировать сообщения независимо от количества клиентов, которым он отправляет сообщение.

## Как это сделать...

Эти шаги охватывают процесс написания и запуска вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter5/udp` и перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter5/udp
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter5/udp
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter5/udp` или используйте это как упражнение для написания собственного кода!
- Создайте новый каталог с именем `server` и перейдите к нему.
- Создайте файл с именем `broadcast.go` со следующим содержимым:

```
package main

import (
    "fmt"
    "net"
    "sync"
    "time"
)

type connections struct {
    addrs map[string]*net.UDPAddr
    // lock for modifying the map
    mu sync.Mutex
}
```

```

func broadcast(conn *net.UDPConn, conns *connections) {
    count := 0
    for {
        count++
        conns.mu.Lock()
        // loop over known addresses
        for _, retAddr := range conns.addrs {

            // send a message to them all
            msg := fmt.Sprintf("Sent %d", count)
            if _, err := conn.WriteToUDP([]byte(msg), retAddr);
err != nil {
                fmt.Printf("error encountered: %s", err.Error())
                continue
            }

        }
        conns.mu.Unlock()
        time.Sleep(1 * time.Second)
    }
}

```

- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "fmt"
    "net"
)

const addr = "localhost:8888"

func main() {
    conns := &connections{
        addrs: make(map[string]*net.UDPAddr),
    }

    fmt.Printf("serving on %s\n", addr)

    // construct a udp addr
    addr, err := net.ResolveUDPAddr("udp", addr)

```

```

    if err != nil {
        panic(err)
    }

    // listen on our specified addr
    conn, err := net.ListenUDP("udp", addr)
    if err != nil {
        panic(err)
    }
    // cleanup
    defer conn.Close()

    // async send messages to all known clients
    go broadcast(conn, conns)

    msg := make([]byte, 1024)
    for {
        // receive a message to gather the ip address
        // and port to send back to
        _, retAddr, err := conn.ReadFromUDP(msg)
        if err != nil {
            continue
        }

        //store it in a map
        conns.mu.Lock()
        conns.addrs[retAddr.String()] = retAddr
        conns.mu.Unlock()
        fmt.Printf("%s connected\n", retAddr)
    }
}

```

- Перейдите к предыдущему каталогу.
- Создайте новый каталог с именем `client` и перейдите в него.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "fmt"
    "net"
)

```

```

const addr = "localhost:8888"

func main() {
    fmt.Printf("client for server url: %s\n", addr)

    addr, err := net.ResolveUDPAddr("udp", addr)
    if err != nil {
        panic(err)
    }

    conn, err := net.DialUDP("udp", nil, addr)
    if err != nil {
        panic(err)
    }
    defer conn.Close()

    msg := make([]byte, 512)
    n, err := conn.Write([]byte("connected"))
    if err != nil {
        panic(err)
    }
    for {
        n, err = conn.Read(msg)
        if err != nil {
            continue
        }
        fmt.Printf("%s\n", string(msg[:n]))
    }
}

```

- Перейдите к предыдущему каталогу.
- Запустите `go run ./server` и вы увидите следующий вывод:

```

$ go run ./server
serving on localhost:8888

```

- В отдельном терминале запустите `go run ./client` из каталога `udp`, и вы увидите следующий вывод, хотя количество может отличаться:

```

$ go run ./client
client for server url: localhost:8888
Sent 3

```

**Sent 4****Sent 5**

- Перейдите к терминалу, на котором работает сервер, и вы должны увидеть что-то похожее на следующее:

```
$ go run ./server
serving on localhost:8888
127.0.0.1:64242 connected
```

- Нажмите **Ctrl + C**, чтобы выйти из сервера и клиента.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите **go test**. Убедитесь, что все тесты пройдены.

## Как это работает...

Сервер прослушивает порт **8888**, как и в предыдущем рецепте. Если клиент запускается, он отправляет сообщение на сервер, и сервер добавляет свой адрес в список адресов. Поскольку клиенты могут подключаться асинхронно, сервер должен использовать мьютекс перед изменением или чтением из списка.

Отдельная широкоовещательная горутина запускается отдельно и отправляет одно и то же сообщение на все адреса клиентов, которые ранее отправляли ей сообщения. Предполагая, что они все еще слушают, они получают одно и то же сообщение с сервера примерно в одно и то же время. Вы также можете подключиться к большому количеству клиентов, чтобы увидеть, как это работает.

## Работа с разрешением доменного имени

Пакет **net** предоставляет ряд полезных функций для поиска DNS. Эта информация сравнима с той, которую вы можете получить, используя команду **dig** в Unix. Эта информация может оказаться чрезвычайно полезной для реализации любого вида сетевого программирования, требующего динамического определения IP-адресов.

Этот рецепт исследует, как вы можете собрать эти данные. Чтобы продемонстрировать это, мы реализуем упрощенную команду **dig**. Мы

постараемся сопоставить URL-адрес со всеми его адресами IPv4 и IPv6. Изменив `CODEBUG=netdns=` на `go` или `cgo`, он будет использовать либо чистый преобразователь Go DNS, либо преобразователь `cgo`. По умолчанию используется чистый DNS-преобразователь Go.

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter5/dns` и перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter5/dns
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter5/dns
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter5/dns` или используйте это как упражнение для написания собственного кода!
- Создайте файл `dns.go` со следующим содержимым:

```
package dns

import (
    "fmt"
    "net"

    "github.com/pkg/errors"
)

// Lookup holds the DNS information we care about
type Lookup struct {
    cname string
    hosts []string
}
```

```
// We can use this to print the lookup object
func (d *Lookup) String() string {
    result := ""
    for _, host := range d.hosts {
        result += fmt.Sprintf("%s IN A %s\n", d.cname, host)
    }
    return result
}

// LookupAddress returns a DNSLookup consisting of a
// cname and host
// for a given address
func LookupAddress(address string) (*Lookup, error) {
    cname, err := net.LookupCNAME(address)
    if err != nil {
        return nil, errors.Wrap(err, "error looking up
CNAME")
    }
    hosts, err := net.LookupHost(address)
    if err != nil {
        return nil, errors.Wrap(err, "error looking up HOST")
    }

    return &Lookup{cname: cname, hosts: hosts}, nil
}
```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл `main.go` со следующим содержимым:

```
package main

import (
    "fmt"
    "log"
    "os"

    "github.com/PacktPublishing/Go-Programming-Cookbook-
Second-Edition/chapter5/dns"
)

func main() {
    if len(os.Args) < 2 {
        fmt.Printf("Usage: %s <address>\n", os.Args[0])
    }
}
```

```

    os.Exit(1)
}
address := os.Args[1]
lookup, err := dns.LookupAddress(address)
if err != nil {
    log.Panicf("failed to lookup: %s", err.Error())
}
fmt.Println(lookup)
}

```

- Запустите команду `go run main.go golang.org`.
- Вы также можете запустить следующее:

```

$ go build
$ ./example golang.org

```

Вы должны увидеть следующий вывод:

```

$ go run main.go golang.org
golang.org. IN A 172.217.5.17
golang.org. IN A 2607:f8b0:4009:809::2011

```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Этот рецепт выполнил `CNAME` и поиск хоста по предоставленному адресу. В нашем случае мы использовали `golang.org`. Мы сохраняем результат в структуре поиска, которая выводит результаты вывода с помощью метода `String()`. Этот метод будет вызываться автоматически, когда мы печатаем наш объект в виде строки, или мы можем вызвать метод напрямую. Мы реализуем некоторую базовую проверку аргументов в `main.go`, чтобы убедиться, что адрес предоставляется при запуске программы.

## Работа с веб-сокетами



WebSockets позволяют серверному приложению подключаться к веб-клиенту, написанному на JavaScript. Это позволяет создавать веб-приложения с двусторонней связью и создавать обновления, такие как чаты и многое другое.

В этом рецепте рассматривается написание сервера WebSocket на Go, а также демонстрируется процесс использования клиентом и взаимодействия с сервером WebSocket. Он использует [github.com/gorilla/websocket](https://github.com/gorilla/websocket) для обновления стандартного обработчика до обработчика WebSocket, а также для создания клиентского приложения.

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter5/websocket` и перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter5/websocket
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter5/websocket
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter5/websocket` или используйте это как упражнение для написания собственного кода!
- Создайте новый каталог с именем `server` и перейдите к нему.
- Создайте файл с именем `handler.go` со следующим содержимым:

```
package main
```

```
import (  
    "log"  
    "net/http"  
  
    "github.com/gorilla/websocket"
```

```

)

// upgrader takes an http connection and converts it
// to a websocket one, we're using some recommended
// basic buffer sizes
var upgrader = websocket.Upgrader{
    ReadBufferSize: 1024,
    WriteBufferSize: 1024,
}

func wsHandler(w http.ResponseWriter, r *http.Request) {
    // upgrade the connection
    conn, err := upgrader.Upgrade(w, r, nil)
    if err != nil {
        log.Println("failed to upgrade connection: ", err)
        return
    }
    for {
        // read and echo back messages in a loop
        messageType, p, err := conn.ReadMessage()
        if err != nil {
            log.Println("failed to read message: ", err)
            return
        }
        log.Printf("received from client: %#v", string(p))
        if err := conn.WriteMessage(messageType, p); err !=
nil {
            log.Println("failed to write message: ", err)
            return
        }
    }
}

```

- Создайте файл с именем **main.go** со следующим содержимым:

```

package main

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

```

```
func main() {
    fmt.Println("Listening on port :8000")
    // we mount our single handler on port localhost:8000 to
    handle all
    // requests
    log.Panic(http.ListenAndServe("localhost:8000",
    http.HandlerFunc(wsHandler)))
}
```

- Перейдите к предыдущему каталогу.
- Создайте новый каталог с именем `client` и перейдите в него.
- Создайте файл с именем `process.go` со следующим содержимым:

```
package main

import (
    "bufio"
    "fmt"
    "log"
    "os"
    "strings"

    "github.com/gorilla/websocket"
)

func process(c *websocket.Conn) {
    reader := bufio.NewReader(os.Stdin)
    for {
        fmt.Printf("Enter some text: ")
        // this will block ctrl-c, to exit press it then hit
        enter
        // or kill from another location
        data, err := reader.ReadString('\n')
        if err != nil {
            log.Println("failed to read stdin", err)
        }

        // trim off the space from reading the string
        data = strings.TrimSpace(data)

        // write the message as a byte across the websocket
        err = c.WriteMessage(websocket.TextMessage,
        []byte(data))
    }
}
```

```

    if err != nil {
        log.Println("failed to write message:", err)
        return
    }

    // this is an echo server, so we can always read
    after the write
    _, message, err := c.ReadMessage()
    if err != nil {
        log.Println("failed to read:", err)
        return
    }
    log.Printf("received back from server: %#v\n",
string(message))
}
}

```

- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "log"
    "os"
    "os/signal"

    "github.com/gorilla/websocket"
)

// catchSig cleans up our websocket conenction if we kill
the program
// with a ctrl-c
func catchSig(ch chan os.Signal, c *websocket.Conn) {
    // block on waiting for a signal
    <-ch
    err := c.WriteMessage(websocket.CloseMessage,
websocket.FormatCloseMessage(websocket.CloseNormalClosure
, ""))
    if err != nil {
        log.Println("write close:", err)
    }
    return
}

```

```

func main() {
    // connect the os signal to our channel
    interrupt := make(chan os.Signal, 1)
    signal.Notify(interrupt, os.Interrupt)

    // use the ws:// Scheme to connect to the websocket
    u := "ws://localhost:8000/"
    log.Printf("connecting to %s", u)

    c, _, err := websocket.DefaultDialer.Dial(u, nil)
    if err != nil {
        log.Fatal("dial:", err)
    }
    defer c.Close()

    // dispatch our signal catcher
    go catchSig(interrupt, c)

    process(c)
}

```

- Перейдите к предыдущему каталогу.
- Запустите `go run ./server` и вы увидите следующий вывод:

```

$ go run ./server
Listening on port :8000

```

- В отдельном терминале запустите `go run ./client` из каталога `websocket`, и вы увидите следующий вывод:

```

$ go run ./client
2019/05/26 11:53:20 connecting to ws://localhost:8000/
Enter some text:

```

- Введите `test` строку, и вы должны увидеть следующее:

```

$ go run ./client
2019/05/26 11:53:20 connecting to ws://localhost:8000/
Enter some text: test
2019/05/26 11:53:22 received back from server: "test"
Enter some text:

```

- Перейдите к терминалу, на котором запущен сервер, и вы должны увидеть что-то похожее на следующее:

```
$ go run ./server
Listening on port :8000
2019/05/26 11:53:22 received from client: "test"
```

- Нажмите *Ctrl + C*, чтобы выйти из сервера и клиента. Возможно, вам также придется нажать *Enter* после нажатия *Ctrl + C* на клиенте.
- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Сервер прослушивает порт `8000` для соединений WebSocket. Когда приходит запрос, пакет `github.com/gorilla/websocket` используется для обновления запроса до соединения WebSocket. Как и в предыдущих примерах эхо-сервера, сервер ожидает сообщения в соединении WebSocket и отвечает тем же сообщением обратно клиенту. Поскольку это обработчик, он может асинхронно обрабатывать множество подключений WebSocket, и они будут оставаться подключенными до тех пор, пока клиент не завершит работу.

В клиент мы добавили функцию `catchsig` для обработки события *Ctrl+C*. Это позволяет нам аккуратно разорвать соединение с сервером при выходе клиента. В противном случае клиент просто принимает пользовательский ввод на `STDIN` и отправляет его на сервер, регистрирует ответ, а затем повторяет.

## Работа с net/гpc для вызова удаленных методов

Go предоставляет вашей системе базовые функции RPC с помощью пакета `net/гpc`. Это потенциальная альтернатива вызовам RPC без использования GRPC или других более сложных пакетов RPC. Однако его функциональность довольно ограничена, и любая функция, которую вы хотите экспортировать, должна соответствовать очень специфической сигнатуре функции.

Комментарии в коде отмечают некоторые из этих ограничений для метода, который можно вызывать удаленно. Этот рецепт демонстрирует, как создать общую функцию, которая имеет ряд параметров, передаваемых через структуру, и может быть вызвана удаленно.

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter5/rpc` и перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-  
Cookbook-Second-Edition/chapter5/rpc
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-  
Cookbook-Second-Edition/chapter5/rpc
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter5/rpc` или используйте это как упражнение для написания собственного кода!
- Создайте новый каталог с именем `tweak` и перейдите к нему.
- Создайте файл с именем `tweak.go` со следующим содержимым:

```
package tweak
```

```
import (  
    "strings"  
)
```

```
// StringTweaker is a type of string  
// that can reverse itself  
type StringTweaker struct{}
```

```
// Args are a list of options for how to tweak  
// the string  
type Args struct {
```

```

    String string
    ToUpper bool
    Reverse bool
}

// Tweak conforms to the RPC library which require:
// - the method's type is exported.
// - the method is exported.
// - the method has two arguments, both exported (or
// builtin) types.
// - the method's second argument is a pointer.
// - the method has return type error.
func (s StringTweaker) Tweak(args *Args, resp *string)
error {

    result := string(args.String)
    if args.ToUpper {
        result = strings.ToUpper(result)
    }
    if args.Reverse {
        runes := []rune(result)
        for i, j := 0, len(runes)-1; i < j; i, j = i+1, j-1 {
            runes[i], runes[j] = runes[j], runes[i]
        }
        result = string(runes)
    }
    *resp = result
    return nil
}

```

- Перейдите к предыдущему каталогу.
- Создайте новый каталог с именем `server` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "fmt"
    "log"
    "net"
    "net/http"
    "net/rpc"

```



```

    "github.com/PacktPublishing/Go-Programming-Cookbook-
    Second-Edition/chapter5/rpc/tweak"
)

```

```

func main() {
    s := new(tweak.StringTweaker)
    if err := rpc.Register(s); err != nil {
        log.Fatal("failed to register:", err)
    }

    rpc.HandleHTTP()

    l, err := net.Listen("tcp", ":1234")
    if err != nil {
        log.Fatal("listen error:", err)
    }

    fmt.Println("listening on :1234")
    log.Panic(http.Serve(l, nil))
}

```

- Перейдите к предыдущему каталогу.
- Создайте новый каталог с именем `client` и перейдите в него.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

```

```

import (
    "fmt"
    "log"
    "net/rpc"

```

```

    "github.com/PacktPublishing/Go-Programming-Cookbook-
    Second-Edition/chapter5/rpc/tweak"
)

```

```

func main() {
    client, err := rpc.DialHTTP("tcp", "localhost:1234")
    if err != nil {
        log.Fatal("error dialing:", err)
    }
}

```

```

    args := tweak.Args{
        String: "this string should be uppercase and
reversed",
        ToUpper: true,
        Reverse: true,
    }
    var result string
    err = client.Call("StringTweaker.Tweak", args, &result)
    if err != nil {
        log.Fatal("client call with error:", err)
    }
    fmt.Printf("the result is: %s", result)
}

```

- Перейдите к предыдущему каталогу.
- Запустите `go run ./server` и вы увидите следующий вывод:

```

$ go run ./server
Listening on :1234

```

- В отдельном терминале запустите `go run ./client` из каталога `rpc`, и вы увидите следующий вывод:

```

$ go run ./client
the result is: DESREVER DNA ESACREPPU EB DLUOHS GNIRTS
SIHT

```

- Press *Ctrl* + *C* to exit the server.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Структура `StringTweaker` вынесена в отдельную библиотеку, чтобы ее экспортированные типы могли быть доступны клиенту (для установки аргументов) и серверу (для регистрации RPC и запуска сервера). Он также соответствует правилам, упомянутым в начале этого рецепта, для работы с `net/rpc`.

`StringTweaker` можно использовать для получения входной строки и, при необходимости, инвертирования и преобразования в верхний регистр всех символов, содержащихся в ней, в зависимости от

переданных параметров. Этот шаблон может быть расширен для создания гораздо более сложных функций, а также вы можете использовать дополнительные функции, чтобы сделать код более читаемым по мере его роста.

## Использование `net/mail` для разбора писем

Пакет `net/mail` предоставляет ряд полезных функций, которые помогут вам при работе с электронной почтой. Если у вас есть необработанный текст электронного письма, его можно разобрать на извлечения заголовков, информацию о дате отправки и многое другое. Этот рецепт продемонстрирует некоторые из этих функций, анализируя необработанное электронное письмо, жестко закодированное в виде строки.

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В своем терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter5/mail` и перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter5/mail
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter5/mail
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter5/mail` или используйте это как упражнение, чтобы написать свой собственный код!
- Создайте файл `header.go` со следующим содержимым:

```
package main
```

```

import (
    "fmt"
    "net/mail"
    "strings"
)

// extract header info and print it nicely
func printHeaderInfo(header mail.Header) {

    // this works because we know it's a single address
    // otherwise use ParseAddressList
    toAddress, err := mail.ParseAddress(header.Get("To"))
    if err == nil {
        fmt.Printf("To: %s <%s>\n", toAddress.Name,
            toAddress.Address)
    }
    fromAddress, err :=
mail.ParseAddress(header.Get("From"))
    if err == nil {
        fmt.Printf("From: %s <%s>\n", fromAddress.Name,
            fromAddress.Address)
    }

    fmt.Println("Subject:", header.Get("Subject"))

    // this works for a valid RFC5322 date
    // it does a header.Get("Date"), then a
    // mail.ParseDate(that_result)
    if date, err := header.Date(); err == nil {
        fmt.Println("Date:", date)
    }

    fmt.Println(strings.Repeat("=", 40))
    fmt.Println()
}

```

- Создайте файл с именем **main.go** со следующим содержимым:

```
package main
```

```
import (
    "io"
    "log"

```

```

    "net/mail"
    "os"
    "strings"
)

// an example email message
const msg string = `Date: Thu, 24 Jul 2019 08:00:00 -0700
From: Aaron <fake_sender@example.com>
To: Reader <fake_receiver@example.com>
Subject: Gophercon 2019 is going to be awesome!
`

```

Feel free to share my book with others if you're attending.  
 This recipe can be used to process and parse email information.

```

func main() {
    r := strings.NewReader(msg)
    m, err := mail.ReadMessage(r)
    if err != nil {
        log.Fatal(err)
    }

    printHeaderInfo(m.Header)

    // after printing the header, dump the body to stdout
    if _, err := io.Copy(os.Stdout, m.Body); err != nil {
        log.Fatal(err)
    }
}

```

- Выполните команду `go run ..`
- Вы также можете запустить следующее:

```

$ go build
$ ./mail

```

Вы должны увидеть следующий вывод:

```

$ go run .
To: Reader <fake_receiver@example.com>
From: Aaron <fake_sender@example.com>
Subject: Gophercon 2019 is going to be awesome!

```

Date: 2019-07-24 08:00:00 -0700 -0700

=====

Feel free to share my book with others if you're attending.

This recipe can be used to process and parse email information.

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Функция `printHeaderInfo` выполняет большую часть работы по этому рецепту. Он анализирует адреса из заголовка в структуру `*mail.Address` и анализирует заголовок даты в объект даты. Затем он берет всю информацию в сообщении и форматирует ее в удобочитаемый формат. Основная функция анализирует начальное электронное письмо и передает этот заголовок.

## 6. Все о базах данных и хранилищах

Приложения Go часто нуждаются в долговременном хранилище. Обычно это реляционные и нереляционные базы данных, а также хранилища ключей и значений и многое другое. При работе с этими приложениями хранения это помогает обернуть ваши операции в интерфейс. В этой главе рассматриваются различные интерфейсы хранения, рассматривается параллельный доступ с такими вещами, как пулы соединений, и рассматриваются общие советы по интеграции новой библиотеки, что часто бывает при использовании новой технологии хранения.

В этой главе будут рассмотрены следующие рецепты:

- Использование пакета `database/sql` с MySQL
- Выполнение интерфейса транзакции базы данных
- Пул подключений, ограничение скорости и таймауты для SQL
- Работа с Редисом
- Использование NoSQL с MongoDB
- Создание интерфейсов хранения для переносимости данных

### Использование пакета `database/sql` с MySQL

Реляционные базы данных являются одними из наиболее хорошо изученных и распространенных вариантов баз данных. MySQL и PostgreSQL — две самые популярные реляционные базы данных с открытым исходным кодом. Этот рецепт продемонстрирует пакет `database/sql`, который предоставляет хуки для ряда реляционных баз данных и автоматически обрабатывает пул соединений и продолжительность соединения, а также предоставляет доступ к ряду основных операций с базой данных.

Этот рецепт будет использовать базу данных MySQL, чтобы установить соединение, вставить некоторые простые данные и запросить их. Он очистит базу данных после использования, удалив таблицу.

### Подготовка

Настройте среду в соответствии с этими шагами:

- Загрузите и установите Go 1.12.6 или более позднюю версию в своей операционной системе по адресу <https://golang.org/doc/install>.
- Откройте терминал или консольное приложение, создайте и перейдите в каталог проекта, например `~/projects/go-programming-cookbook`. Весь наш код будет запускаться и изменяться из этого каталога.
- Скопируйте последний код в `~/projects/go-programming-cookbook-original` и, при желании, работайте из этого каталога, вместо того, чтобы вводить примеры вручную:

```
$ git clone git@github.com:PacktPublishing/Go-Programming-Cookbook-Second-Edition.git go-programming-cookbook-original
```

- Установите и настройте MySQL, используя <https://dev.mysql.com/doc/mysql-getting-started/en/>.
- Запустите команду `MYSQLEXPORT=<your mysql username>`.
- Запустите команду `export MYSQLPASSWORD=<your mysql password>`.

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter6/database` и перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter6/database
```

Вы должны увидеть файл с именем `go.mod`, содержащий следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter6/database
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter6/database` или используйте это как упражнение для написания собственного кода!
- Создайте файл с именем `config.go` со следующим содержимым:

```
package database

import (
    "database/sql"
    "fmt"
    "os"
    "time"
```



```

supported    _ "github.com/go-sql-driver/mysql" //we import
              libraries for database/sql
            )

            // Example hold the results of our queries
            type Example struct {
                Name string
                Created *time.Time
            }

            // Setup configures and returns our database
            // connection pool
            func Setup() (*sql.DB, error) {
                db, err := sql.Open("mysql",
                    fmt.Sprintf("%s:%s@gocookbook?
                    parseTime=true", os.Getenv("MYSQLUSERNAME"),
                    os.Getenv("MYSQLPASSWORD")))
                if err != nil {
                    return nil, err
                }
                return db, nil
            }

```

- Создайте файл с именем `create.go` со следующим содержимым:

```

package database

import (
    "database/sql"

supported    _ "github.com/go-sql-driver/mysql" //we import
              libraries for database/sql
            )

            // Create makes a table called example
            // and populates it
            func Create(db *sql.DB) error {
                // create the database
                if _, err := db.Exec("CREATE TABLE example (name
                VARCHAR(20), created DATETIME)"); err != nil {
                    return err
                }

                if _, err := db.Exec(`INSERT INTO example (name,

```

```

created)
    values ("Aaron", NOW()))`); err != nil {
        return err
    }

    return nil
}

```

- Создайте файл с именем `query.go` со следующим содержимым:

```

package database

import (
    "database/sql"
    "fmt"

    _ "github.com/go-sql-driver/mysql" //we import
supported    libraries for database/sql
)

// Query grabs a new connection
// creates tables, and later drops them
// and issues some queries
func Query(db *sql.DB, name string) error {
    name := "Aaron"
    rows, err := db.Query("SELECT name, created FROM
example      where name=?", name)
    if err != nil {
        return err
    }
    defer rows.Close()
    for rows.Next() {
        var e Example
        if err := rows.Scan(&e.Name, &e.Created); err !=
nil {
            return err
        }
        fmt.Printf("Results:\n\tName: %s\n\tCreated:
%v\n",
            e.Name, e.Created)
    }
    return rows.Err()
}

```

- Создайте файл с именем `exec.go` со следующим содержимым:

```

package database

// Exec replaces the Exec from the previous
// recipe
func Exec(db DB) error {

    // uncaught error on cleanup, but we always
    // want to cleanup
    defer db.Exec("DROP TABLE example")

    if err := Create(db); err != nil {
        return err
    }

    if err := Query(db, "Aaron"); err != nil {
        return err
    }
    return nil
}

```

- Создайте и перейдите в каталог `example`.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "PacktPublishing/Go-Programming-Cookbook-Second-
Edition/
    go-cookbook/chapter6/database"
    _ "github.com/go-sql-driver/mysql" //we import
supported
    libraries for database/sql
)

func main() {
    db, err := database.Setup()
    if err != nil {
        panic(err)
    }

    if err := database.Exec(db); err != nil {
        panic(err)
    }
}

```

- Выполните `go run main.go`.
- Вы также можете запустить следующую команду:

```
$ go build
$ ./example
```

Вы должны увидеть следующий вывод:

```
$ go run main.go
Results:
Name: Aaron
Created: 2017-02-16 19:02:36 +0000 UTC
```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Строка кода `_ "github.com/go-sql-driver/mysql"` — это то, как вы подключаете различные коннекторы базы данных к пакету `database/sql`. Существуют также альтернативные пакеты MySQL, которые можно импортировать таким же образом для получения аналогичных результатов. Команды были бы похожи, если бы вы подключались к PostgreSQL, SQLite или любым другим, которые реализуют интерфейсы `database/sql`.

После подключения пакет устанавливает пул соединений, который описан в рецепте «*Пул подключений, ограничение скорости и таймауты для SQL*», и вы можете либо напрямую выполнять SQL для соединения, либо создавать объекты транзакций, которые могут делать все, что может делать соединение, с помощью команд `commit` и `rollback`.

Пакет `mysql` обеспечивает некоторую удобную поддержку объектов времени Go при обращении к базе данных. Этот рецепт также извлекает имя пользователя и пароль из переменных среды `MYSQLUSERNAME` и `MYSQLPASSWORD`.

## Выполнение интерфейса транзакции базы данных

При работе с подключениями к таким службам, как база данных, написание тестов может быть затруднено. Это связано с тем, что в Go сложно имитировать или утиные вещи во время выполнения. Хотя я рекомендую использовать интерфейс хранилища при работе с базами данных, все же полезно смоделировать интерфейс транзакций базы данных внутри этого

интерфейса. Рецепт «Создание интерфейсов хранения для переносимости данных» охватывает интерфейсы хранения; этот рецепт сосредоточится на интерфейсе для соединения с базой данных и объектов транзакций.

Чтобы показать использование такого интерфейса, мы перепишем файлы создания и запроса из предыдущего рецепта, чтобы использовать наш интерфейс. Окончательный вывод будет таким же, но операции создания и запроса будут выполняться в транзакции.

## Подготовка

Обратитесь к разделу «Подготовка» в рецепте «Использование пакета *database/sql* с *MySQL*».

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter6/dbinterface` и перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter6/dbinterface
```

Вы должны увидеть файл с именем `go.mod`, содержащий следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-
Second-Edition/chapter6/dbinterface
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter6/dbinterface` или используйте это как упражнение для написания собственного кода!
- Создайте файл с именем `transaction.go` со следующим содержимым:

```
package dbinterface

import "database/sql"

// DB is an interface that is satisfied
// by an sql.DB or an sql.Transaction
type DB interface {
    Exec(query string, args ...interface{}) (sql.Result, error)
    Prepare(query string) (*sql.Stmt, error)
    Query(query string, args ...interface{}) (*sql.Rows, error)
    QueryRow(query string, args ...interface{}) *sql.Row
```

```

}

// Transaction can do anything a Query can do
// plus Commit, Rollback, or Stmt
type Transaction interface {
    DB
    Commit() error
    Rollback() error
}

```

- Создайте файл с именем `create.go` со следующим содержимым:

```

package dbinterface

import _ "github.com/go-sql-driver/mysql" //we import
supported libraries for database/sql

// Create makes a table called example
// and populates it
func Create(db DB) error {
    // create the database
    if _, err := db.Exec("CREATE TABLE example (name
VARCHAR(20), created DATETIME)"); err != nil {
        return err
    }

    if _, err := db.Exec(`INSERT INTO example (name, created)
values ("Aaron", NOW())`); err != nil {
        return err
    }

    return nil
}

```

- Создайте файл с именем `query.go` со следующим содержимым:

```

package dbinterface

import (
    "fmt"

    "github.com/PacktPublishing/Go-Programming-Cookbook-Second-
Edition/chapter6/database"
)

// Query grabs a new connection
// creates tables, and later drops them
// and issues some queries

```

```

func Query(db DB) error {
    name := "Aaron"
    rows, err := db.Query("SELECT name, created FROM example
where name=?", name)
    if err != nil {
        return err
    }
    defer rows.Close()
    for rows.Next() {
        var e database.Example
        if err := rows.Scan(&e.Name, &e.Created); err != nil {
            return err
        }
        fmt.Printf("Results:\n\tName: %s\n\tCreated: %v\n",
e.Name,
                e.Created)
    }
    return rows.Err()
}

```

- Создайте файл с именем `exec.go` со следующим содержимым:

```

package dbinterface

// Exec replaces the Exec from the previous
// recipe
func Exec(db DB) error {

    // uncaught error on cleanup, but we always
    // want to cleanup
    defer db.Exec("DROP TABLE example")

    if err := Create(db); err != nil {
        return err
    }

    if err := Query(db); err != nil {
        return err
    }
    return nil
}

```

- Перейдите к `example`.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (

```

```

    "github.com/PacktPublishing/Go-Programming-Cookbook-Second-
Edition/chapter6/database"
    "github.com/PacktPublishing/Go-Programming-Cookbook-Second-
Edition/chapter6/dbinterface"
    _ "github.com/go-sql-driver/mysql" //we import supported
libraries for database/sql
)

func main() {
    db, err := database.Setup()
    if err != nil {
        panic(err)
    }

    tx, err := db.Begin()
    if err != nil {
        panic(err)
    }
    // this wont do anything if commit is successful
    defer tx.Rollback()

    if err := dbinterface.Exec(tx); err != nil {
        panic(err)
    }
    if err := tx.Commit(); err != nil {
        panic(err)
    }
}

```

- Выполните `go run main.go`.
- Вы также можете запустить следующую команду:

```

$ go build
$ ./example

```

Вы должны увидеть следующий вывод:

```

$ go run main.go
Results:
Name: Aaron
Created: 2017-02-16 20:00:00 +0000 UTC

```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...



Этот рецепт работает очень похоже на предыдущий рецепт «*Использование пакета database/sql с MySQL*». Этот рецепт выполняет ту же операцию создания данных и запроса к ним, но также демонстрирует использование транзакций и создание универсальных функций базы данных, которые работают как с соединениями `sql.DB`, так и с объектами `sql.Transaction`.

Код, написанный таким образом, позволяет нам повторно использовать функции, выполняющие операции с базой данных, которые можно запускать индивидуально или в группах с помощью транзакции. Это позволяет повторно использовать больше кода, сохраняя при этом функциональность функций или методов, работающих с базой данных. Например, вы можете иметь функции `Update(db DB)` для нескольких таблиц и передавать им все общие транзакции для атомарного выполнения нескольких обновлений. Кроме того, эти интерфейсы проще имитировать, как вы увидите в [Главе 9](#) «*Тестирование кода Go*».

## Пул подключений, ограничение скорости и таймауты для SQL

Хотя пакет `database/sql` обеспечивает поддержку пулов соединений, ограничения скорости и таймаутов, часто важно настроить значения по умолчанию, чтобы они лучше соответствовали конфигурации вашей базы данных. Это может стать важным, если у вас есть горизонтальное масштабирование микросервисов и вы не хотите поддерживать слишком много активных подключений к базе данных.

### Подготовка

Обратитесь к разделу «*Подготовка*» в рецепте «*Использование пакета database/sql с MySQL*».

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter6/pools` и перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter6/pools
```

Вы должны увидеть файл с именем `go.mod`, содержащий следующее:

`module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter6/pools`

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter6/pools` или используйте это как упражнение для написания собственного кода!
- Создайте файл `pools.go` со следующим содержимым:

```
package pools

import (
    "database/sql"
    "fmt"
    "os"

    _ "github.com/go-sql-driver/mysql" //we import
supported    libraries for database/sql
)

// Setup configures the db along with pools
// number of connections and more
func Setup() (*sql.DB, error) {
    db, err := sql.Open("mysql",
        fmt.Sprintf("%s:%s@gocookbook?
        parseTime=true", os.Getenv("MYSQLUSERNAME"),
        os.Getenv("MYSQLPASSWORD")))
    if err != nil {
        return nil, err
    }

    // there will only ever be 24 open connections
    db.SetMaxOpenConns(24)

    // MaxIdleConns can never be less than max open
    // SetMaxOpenConns otherwise it'll default to that
value        db.SetMaxIdleConns(24)

    return db, nil
}
```

- Создайте файл `timeout.go` со следующим содержимым:

```
package pools
```

```
import (
```

```

    "context"
    "time"
)

// ExecWithTimeout will timeout trying
// to get the current time
func ExecWithTimeout() error {
    db, err := Setup()
    if err != nil {
        return err
    }

    ctx := context.Background()

    // we want to timeout immediately
    ctx, cancel := context.WithDeadline(ctx, time.Now())

    // call cancel after we complete
    defer cancel()

    // our transaction is context aware
    _, err = db.BeginTx(ctx, nil)
    return err
}

```

- Перейдите к `example`.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import "PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    go-cookbook/chapter6/pools"

func main() {
    if err := pools.ExecWithTimeout(); err != nil {
        panic(err)
    }
}

```

- Выполните `go run main.go`.
- Вы также можете запустить следующее:

```

$ go build
$ ./example

```

Вы должны увидеть следующий вывод:

```
$ go run main.go
panic: context deadline exceeded
```

```
goroutine 1 [running]:
main.main()
/go/src/PacktPublishing/Go-Programming-Cookbook-Second-
Edition/go-cookbook/chapter6/pools/example/main.go:7 +0x4e
exit status 2
```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Возможность контролировать глубину нашего пула соединений очень полезна. Это предотвратит перегрузку базы данных, но важно учитывать, что это будет означать в контексте таймаутов. Если вы применяете как установленное количество подключений, так и строгие таймауты на основе контекста, как мы сделали в этом рецепте, будут случаи, когда у вас будут запросы, которые часто будут прерываться по таймауту в перегруженном приложении, пытающемся установить слишком много подключений.

Это связано с тем, что время ожидания соединений истекает, пока соединение не станет доступным. Недавно добавленная функциональность контекста для `database/sql` значительно упрощает использование общего таймаута для всего запроса, включая шаги, связанные с выполнением запроса.

В этом и других рецептах имеет смысл использовать глобальный объект конфигурации для передачи в функцию `Setup()`, хотя в этом рецепте используются только переменные среды.

## Работа с Redis

Иногда вам нужно постоянное хранилище или дополнительные функции, предоставляемые сторонними библиотеками и сервисами. В этом рецепте мы рассмотрим Redis как форму нереляционного хранилища данных и продемонстрируем, как такой язык, как Go, может взаимодействовать с этими сторонними сервисами.

Поскольку Redis поддерживает хранение ключей и значений с простым интерфейсом, он отлично подходит для хранения сеансов или временных данных с определенной продолжительностью. Возможность указать время

ожидания для данных, хранящихся в Redis, чрезвычайно ценна. Этот рецепт исследует базовое использование Redis от настройки до запросов и использования пользовательской сортировки.

## Подготовка

Настройте среду в соответствии с этими шагами:

- Загрузите и установите Go 1.11.1 или более позднюю версию в своей операционной системе по адресу <https://golang.org/doc/install>.
- Установите Consul с <https://www.consul.io/intro/getting-started/install.html>.
- Откройте терминал или консольное приложение, создайте и перейдите в каталог проекта, например `~/projects/go-programming-cookbook`. Весь наш код будет запускаться и изменяться из этого каталога.
- Скопируйте последний код в `~/projects/go-programming-cookbook-original` и (необязательно) работайте из этого каталога, а не вводите примеры вручную:

```
$ git clone git@github.com:PacktPublishing/Go-Programming-Cookbook-Second-Edition.git go-programming-cookbook-original
```

- Установите и настройте Redis с помощью <https://redis.io/topics/quickstart>.

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter6/redis` и перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter6/redis
```

Вы должны увидеть файл с именем `go.mod`, содержащий следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter6/redis
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter6/redis` или используйте это как упражнение, чтобы написать собственный код!
- Создайте файл с именем `config.go` со следующим содержимым:

```
package redis
```

```

import (
    "os"

    redis "gopkg.in/redis.v5"
)

// Setup initializes a redis client
func Setup() (*redis.Client, error) {
    client := redis.NewClient(&redis.Options{
        Addr: "localhost:6379",
        Password: os.Getenv("REDISPASSWORD"),
        DB: 0, // use default DB
    })

    _, err := client.Ping().Result()
    return client, err
}

```

- Создайте файл с именем **exec.go** со следующим содержимым:

```

package redis

import (
    "fmt"
    "time"

    redis "gopkg.in/redis.v5"
)

// Exec performs some redis operations
func Exec() error {
    conn, err := Setup()
    if err != nil {
        return err
    }

    c1 := "value"
    // value is an interface, we can store whatever
    // the last argument is the redis expiration
    conn.Set("key", c1, 5*time.Second)

    var result string
    if err := conn.Get("key").Scan(&result); err != nil {
        switch err {
            // this means the key
            // was not found

```

```

        case redis.Nil:
            return nil
        default:
            return err
    }
}

fmt.Println("result =", result)

return nil
}

```

- Создайте файл `sort.go` со следующим содержимым:

```

package redis

import (
    "fmt"

    redis "gopkg.in/redis.v5"
)

// Sort performs a sort redis operations
func Sort() error {
    conn, err := Setup()
    if err != nil {
        return err
    }

    listkey := "list"
    if err := conn.LPush(listkey, 1).Err(); err != nil {
        return err
    }
    // this will clean up the list key if any of the subsequent
    commands error
    defer conn.Del(listkey)

    if err := conn.LPush(listkey, 3).Err(); err != nil {
        return err
    }
    if err := conn.LPush(listkey, 2).Err(); err != nil {
        return err
    }

    res, err := conn.Sort(listkey, redis.Sort{Order:
"ASC"}).Result()
    if err != nil {

```

```

    return err
}
fmt.Println(res)

return nil
}

```

- Перейдите к `example`.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import "PacktPublishing/
        Go-Programming-Cookbook-Second-Edition/
        go-cookbook/chapter6/redis"

func main() {
    if err := redis.Exec(); err != nil {
        panic(err)
    }

    if err := redis.Sort(); err != nil {
        panic(err)
    }
}

```

- Выполните `go run main.go`.
- Вы также можете запустить следующую команду:

```

$ go build
$ ./example

```

Вы должны увидеть следующий вывод:

```

$ go run main.go
result = value
[1 2 3]

```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Работа с Redis в Go очень похожа на работу с MySQL. Хотя стандартной библиотеки нет, многие из тех же соглашений соблюдаются с такими



функциями, как `Scan()`, для чтения данных из Redis в типы Go. Выбрать лучшую библиотеку для таких случаев может быть сложно, и я предлагаю периодически проверять, что доступно, поскольку все может быстро измениться.

В этом рецепте используется пакет `redis` для базовой настройки и получения, более сложной функции сортировки и базовой настройки. Как и в случае с `database/sql`, вы можете установить дополнительную конфигурацию в виде времени ожидания записи, размера пула и т. д. Сам Redis также предоставляет множество дополнительных функций, включая поддержку кластера Redis, объекты Zscore и счетчиков, а также распределенные блокировки.

Как и в предыдущем рецепте, я рекомендую использовать объект `config`, в котором хранятся ваши настройки Redis и сведения о конфигурации для простоты настройки и безопасности.

## Использование NoSQL с MongoDB

Сначала вы можете подумать, что Go лучше подходит для реляционных баз данных из-за структур Go и потому, что Go является типизированным языком. При работе с чем-то вроде пакета [github.com/mongodb/mongo-go-driver](https://github.com/mongodb/mongo-go-driver) Go может почти произвольно хранить и извлекать объекты структуры. Если вы версионите свои объекты, ваша схема может адаптироваться и может обеспечить очень гибкую среду разработки.

Некоторые библиотеки лучше скрывают или возвышают эти абстракции. Пакет `mongo-go-driver` является примером библиотеки, которая отлично справляется с первой задачей. Следующий рецепт создаст соединение аналогично Redis и MySQL, но сохранит и извлечет объект, даже не определяя конкретную схему.

## Подготовка

Настройте среду в соответствии с этими шагами:

- Загрузите и установите Go 1.11.1 или более позднюю версию в своей операционной системе с <https://golang.org/doc/install>.
- Установите Consul с <https://www.consul.io/intro/getting-started/install.html>.
- Откройте терминал или консольное приложение, создайте и перейдите в каталог проекта, например `~/projects/go-programming-cookbook`. Весь код будет запускаться и изменяться из этого каталога.
- Скопируйте последний код в `~/projects/go-programming-cookbook-original` и (необязательно) работайте из этого каталога, а не вводите примеры вручную:

```
$ git clone git@github.com:PacktPublishing/Go-Programming-
Cookbook-Second-Edition.git go-programming-cookbook-original
```

- Установите и настройте MongoDB (<https://docs.mongodb.com/getting-started/shell/>).

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter6/mongodb` и перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter6/mongodb
```

Вы должны увидеть файл с именем `go.mod`, содержащий следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-
Second-Edition/chapter6/mongodb
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter6/mongodb` или используйте это как упражнение, чтобы написать собственный код!
- Создайте файл с именем `config.go` со следующим содержимым:

```
package mongodb

import (
    "context"
    "time"

    "github.com/mongodb/mongo-go-driver/mongo"
    "go.mongodb.org/mongo-driver/mongo/options"
)

// Setup initializes a mongo client
func Setup(ctx context.Context, address string)
(*mongo.Client, error) {
    ctx, cancel := context.WithTimeout(ctx, 10*time.Second)
    // cancel will be called when setup exits
    defer cancel()

    client, err :=
mongo.NewClient(options.Client().ApplyURI(address))
    if err != nil {
```

```

    return nil, err
}

if err := client.Connect(ctx); err != nil {
    return nil, err
}
return client, nil
}

```

- Создайте файл с именем `exec.go` со следующим содержимым:

```

package mongodb

import (
    "context"
    "fmt"

    "github.com/mongodb/mongo-go-driver/bson"
)

// State is our data model
type State struct {
    Name string `bson:"name"`
    Population int `bson:"pop"`
}

// Exec creates then queries an Example
func Exec(address string) error {
    ctx := context.Background()
    db, err := Setup(ctx, address)
    if err != nil {
        return err
    }

    coll := db.Database("gocookbook").Collection("example")

    vals := []interface{}{&State{"Washington", 7062000},
        &State{"Oregon", 3970000}}

    // we can inserts many rows at once
    if _, err := coll.InsertMany(ctx, vals); err != nil {
        return err
    }

    var s State
    if err := coll.FindOne(ctx, bson.M{"name":
        "Washington"}).Decode(&s); err != nil {

```

```

    return err
}

if err := coll.Drop(ctx); err != nil {
    return err
}

fmt.Printf("State: %#v\n", s)
return nil
}

```

- Перейдите к `example`.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import "github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter6/mongodb"

func main() {
    if err := mongodb.Exec("mongodb://localhost"); err != nil {
        panic(err)
    }
}

```

- Выполните `go run main.go`.
- Вы также можете запустить следующую команду:

```

$ go build
$ ./example

```

Вы должны увидеть следующий вывод:

```

$ go run main.go
State: mongodb.State{Name:"Washington", Population:7062000}

```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Пакет `mongo-go-driver` также предоставляет пул соединений и множество способов настройки и настройки ваших соединений с базой данных `mongodb`. Примеры этого рецепта довольно просты, но они иллюстрируют, как легко

рассуждать и запрашивать базу данных на основе документов. Пакет реализует тип данных BSON, и маршалинг в него и из него очень похож на работу с JSON.

Гарантии согласованности и лучшие практики для [mongodb](#) выходят за рамки этой книги. Однако работать с ними на языке Go одно удовольствие.

## Создание интерфейсов хранения для переносимости данных

При работе с внешними интерфейсами хранения может быть полезно абстрагировать ваши операции за интерфейс. Это сделано для простоты имитации, переносимости в случае изменения серверных частей хранилища и изоляции проблем. Недостаток этого подхода может возникнуть, если вам нужно выполнить несколько операций внутри транзакции. В этом случае имеет смысл выполнять составные операции или разрешить их передачу через объект контекста или дополнительные аргументы функции.

Этот рецепт реализует очень простой интерфейс для работы с элементами в MongoDB. Эти элементы будут иметь имя и цену, и мы будем использовать интерфейс для сохранения и извлечения этих объектов.

### Подготовка

Обратитесь к шагам, приведенным в разделе «Подготовка» в рецепте «Использование NoSQL с MongoDB».

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- From your Terminal or console application, create a new directory called [~/projects/go-programming-cookbook/chapter6/storage](#) and Перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter6/storage
```

Вы должны увидеть файл с именем [go.mod](#), содержащий следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter6/storage
```

- Скопируйте тесты из [~/projects/go-programming-cookbook-original/chapter6/storage](#) или используйте это как упражнение,

чтобы написать собственный код!

- Создайте файл с именем `storage.go` со следующим содержимым:

```
package storage

import "context"

// Item represents an item at
// a shop
type Item struct {
    Name  string
    Price int64
}

// Storage is our storage interface
// We'll implement it with Mongo
// storage
type Storage interface {
    GetByName(context.Context, string) (*Item, error)
    Put(context.Context, *Item) error
}
```

- Создайте файл с именем `mongoconfig.go` со следующим содержимым:

```
package storage

import (
    "context"
    "time"

    "github.com/mongodb/mongo-go-driver/mongo"
)

// MongoStorage implements our storage interface
type MongoStorage struct {
    *mongo.Client
    DB string
    Collection string
}

// NewMongoStorage initializes a MongoStorage
func NewMongoStorage(ctx context.Context, connection, db,
collection string) (*MongoStorage, error) {
    ctx, cancel := context.WithTimeout(ctx, 10*time.Second)
    defer cancel()

    client, err := mongo.Connect(ctx, "mongodb://localhost")
```

```

    if err != nil {
        return nil, err
    }

    ms := MongoStorage{
        Client: client,
        DB: db,
        Collection: collection,
    }
    return &ms, nil
}

```

- Создайте файл с именем `mongointerface.go` со следующим содержимым:

```

package storage

import (
    "context"

    "github.com/mongodb/mongo-go-driver/bson"
)

// GetByName queries mongodb for an item with
// the correct name
func (m *MongoStorage) GetByName(ctx context.Context, name
string) (*Item, error) {
    c := m.Client.Database(m.DB).Collection(m.Collection)
    var i Item
    if err := c.FindOne(ctx, bson.M{"name": name}).Decode(&i);
err != nil {
        return nil, err
    }

    return &i, nil
}

// Put adds an item to our mongo instance
func (m *MongoStorage) Put(ctx context.Context, i *Item) error
{
    c := m.Client.Database(m.DB).Collection(m.Collection)
    _, err := c.InsertOne(ctx, i)
    return err
}

```

- Создайте файл с именем `exec.go` со следующим содержимым:

```

package storage

import (
    "context"
    "fmt"
)

// Exec initializes storage, then performs operations
// using the storage interface
func Exec() error {
    ctx := context.Background()
    m, err := NewMongoStorage(ctx, "localhost", "gocookbook",
"items")
    if err != nil {
        return err
    }
    if err := PerformOperations(m); err != nil {
        return err
    }

    if err :=
m.Client.Database(m.DB).Collection(m.Collection).Drop(ctx);
err != nil {
        return err
    }

    return nil
}

// PerformOperations creates a candle item
// then gets it
func PerformOperations(s Storage) error {
    ctx := context.Background()
    i := Item{Name: "candles", Price: 100}
    if err := s.Put(ctx, &i); err != nil {
        return err
    }

    candles, err := s.GetByName(ctx, "candles")
    if err != nil {
        return err
    }
    fmt.Printf("Result: %#v\n", candles)
    return nil
}

```

- Перейдите к `example`.



- Создайте файл с именем `main.go` со следующим содержимым:

```
package main

import "PacktPublishing/Go-Programming-Cookbook-Second-
Edition/
        go-cookbook/chapter6/storage"

func main() {
    if err := storage.Exec(); err != nil {
        panic(err)
    }
}
```

- Выполните `go run main.go`.
- Вы также можете запустить следующую команду:

```
$ go build
$ ./example
```

Вы должны увидеть следующий вывод:

```
$ go run main.go
Result: &storage.Item{Name:"candles", Price:100}
```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Наиболее важной функцией для демонстрации этого рецепта является `PerformOperations`. Эта функция принимает интерфейс `Storage` в качестве параметра. Это означает, что мы можем динамически заменить базовое хранилище, даже не изменяя эту функцию. Было бы просто, например, подключить хранилище к отдельному API, чтобы потреблять и модифицировать его.

Мы используем контекст для этих интерфейсов, чтобы добавить дополнительную гибкость и позволить интерфейсу также обрабатывать таймауты. Отделение логики вашего приложения от базового хранилища дает множество преимуществ, но может быть сложно выбрать правильные места для проведения границ, и это будет сильно различаться в зависимости от приложения.

## 7. Веб-клиенты и API

Работа с API и написание веб-клиентов может оказаться непростой задачей. Различные API имеют разные типы авторизации, аутентификации и протокола. Мы изучим объект структуры [http.Client](#), поработаем с клиентами OAuth2 и долговременным хранилищем токенов и закончим GRPC и дополнительным интерфейсом REST.

К концу этой главы вы должны иметь представление о том, как взаимодействовать со сторонними или внутренними API, и иметь некоторые шаблоны для общих операций, таких как асинхронные запросы к API.

В этой главе мы рассмотрим следующие рецепты:

- Инициализация, хранение и передача структур [http.Client](#)
- Написание клиента для REST API
- Выполнение параллельных и асинхронных клиентских запросов
- Использование клиентов OAuth2
- Реализация интерфейса хранения токенов OAuth2
- Обертывание клиента дополнительным функционалом и функциональной композицией
- Понимание клиентов GRPC
- Использование [twitchtv/twirp](#) для RPC

## Технические требования

Чтобы продолжить выполнение всех рецептов в этой главе, настройте свою среду в соответствии со следующими шагами:

- Загрузите и установите Go 1.12.6 или более позднюю версию в своей операционной системе по адресу <https://golang.org/doc/install>.
- Откройте терминал или консольное приложение, создайте и перейдите в каталог проекта, например `~/projects/go-programming-cookbook`. Весь наш код будет запускаться и изменяться из этого каталога.

- Скопируйте последний код в `~/projects/go-programming-cookbook-original` и, при желании, работайте из этого каталога, вместо того, чтобы вводить примеры вручную:

```
$ git clone git@github.com:PacktPublishing/Go-Programming-Cookbook-Second-Edition.git go-programming-cookbook-original
```

## Инициализация, хранение и передача структур `http.Client`

Пакет Go `net/http` предоставляет гибкую структуру `http.Client` для работы с HTTP API. Эта структура имеет отдельные транспортные функции и позволяет относительно просто сокращать запросы, изменять заголовки для каждой клиентской операции и обрабатывать любые операции REST. Создание клиентов — очень распространенная операция, и этот рецепт начнется с основ работы и создания объекта `http.Client`.

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter7/client` и перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter7/client
```

Вы должны увидеть файл с именем `go.mod`, содержащий следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter7/client
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter7/client` или используйте это как упражнение, чтобы написать собственный код!

- Создайте файл `client.go` со следующим содержимым:

```
package client

import (
    "crypto/tls"
    "net/http"
)

// Setup configures our client and redefines
// the global DefaultClient
func Setup(isSecure, nop bool) *http.Client {
    c := http.DefaultClient

    // Sometimes for testing, we want to
    // turn off SSL verification
    if !isSecure {
        c.Transport = &http.Transport{
            TLSClientConfig: &tls.Config{
                InsecureSkipVerify: false,
            },
        }
    }
    if nop {
        c.Transport = &NopTransport{ }
    }
    http.DefaultClient = c
    return c
}

// NopTransport is a No-Op Transport
type NopTransport struct {
}

// RoundTrip Implements RoundTripper interface
func (n *NopTransport) RoundTrip(*http.Request)
(*http.Response, error) {
    // note this is an uninitialized Response
    // if you're looking at headers etc
    return &http.Response{StatusCode:
http.StatusTeapot}, nil
}
```

- Создайте файл с именем `exes.go` со следующим содержимым:

```
package client

import (
    "fmt"
    "net/http"
)

// DoOps takes a client, then fetches
// google.com
func DoOps(c *http.Client) error {
    resp, err := c.Get("http://www.google.com")
    if err != nil {
        return err
    }
    fmt.Println("results of DoOps:",
resp.StatusCode)

    return nil
}

// DefaultGetGolang uses the default client
// to get golang.org
func DefaultGetGolang() error {
    resp, err :=
http.Get("https://www.golang.org")
    if err != nil {
        return err
    }
    fmt.Println("results of DefaultGetGolang:",
resp.StatusCode)
    return nil
}
```

- Создайте файл `store.go` со следующим содержимым:

```
package client

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

```

// Controller embeds an http.Client
// and uses it internally
type Controller struct {
    *http.Client
}

// DoOps with a controller object
func (c *Controller) DoOps() error {
    resp, err :=
c.Client.Get("http://www.google.com")
    if err != nil {
        return err
    }
    fmt.Println("results of client.DoOps",
resp.StatusCode)
    return nil
}

```

- Создайте новый каталог с именем **example** и перейдите в него.
- Создайте файл с именем **main.go** со следующим содержимым:

```

package main

import "github.com/PacktPublishing/
Go-Programming-Cookbook-Second-Edition/
chapter7/client"

func main() {
    // secure and op!
    cli := client.Setup(true, false)

    if err := client.DefaultGetGolang(); err !=
nil {
        panic(err)
    }

    if err := client.DoOps(cli); err != nil {
        panic(err)
    }

    c := client.Controller{Client: cli}
    if err := c.DoOps(); err != nil {

```

```

        panic(err)
    }

    // secure and noop
    // also modifies default
    client.Setup(true, true)

    if err := client.DefaultGetGolang(); err !=
nil {
        panic(err)
    }
}

```

- Выполните `go run main.go`.
- Вы также можете запустить следующую команду:

```

$ go build
$ ./example

```

Теперь вы должны увидеть следующий вывод:

```

$ go run main.go
results of DefaultGetGolang: 200
results of DoOps: 200
results of client.DoOps 200
results of DefaultGetGolang: 418

```

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Пакет `net/http` предоставляет переменную пакета `DefaultClient`, которая используется следующими внутренними операциями: `Do`, `GET`, `POST` и т. д. Наша функция `Setup()` возвращает клиента и устанавливает тот же клиент по умолчанию. При настройке клиента большинство ваших модификаций будет происходить в транспорте, которому нужно только реализовать интерфейс `RoundTripper`.

В этом рецепте приведен пример бездействующего кругового обхода, который всегда возвращает код состояния 418. Вы можете себе представить, как это может быть полезно для тестирования. Также

демонстрируется передача клиентов в качестве аргументов функции, использование их в качестве параметров структуры и использование клиента по умолчанию для обработки запросов.

## Написание клиента для REST API

Написание клиента для REST API не только поможет вам лучше понять рассматриваемый API, но также даст вам полезный инструмент для всех будущих приложений, использующих этот API. В этом рецепте будет рассмотрено структурирование клиента и показаны некоторые стратегии, которыми вы можете сразу же воспользоваться.

Для этого клиента мы предполагаем, что аутентификация обрабатывается базовой аутентификацией, но также должна быть возможность обращения к конечной точке для получения токена и т. д. Для простоты предположим, что наш API предоставляет одну конечную точку, `GetGoogle()`, которая возвращает код состояния, возвращаемый при выполнении запроса GET на <https://www.google.com>.

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения.:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter7/rest` и перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter7/rest
```

Вы должны увидеть файл с именем `go.mod`, содержащий следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter7/rest
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter7/rest` или используйте это как упражнение, чтобы написать собственный код!
- Создайте файл `client.go` со следующим содержимым:



```

package rest

import "net/http"

// APIClient is our custom client
type APIClient struct {
    *http.Client
}

// NewAPIClient constructor initializes the client
with our
// custom Transport
func NewAPIClient(username, password string)
*APIClient {
    t := http.Transport{}
    return &APIClient{
        Client: &http.Client{
            Transport: &APITransport{
                Transport: &t,
                username: username,
                password: password,
            },
        },
    }
}

// GetGoogle is an API Call - we abstract away
// the REST aspects
func (c *APIClient) GetGoogle() (int, error) {
    resp, err := c.Get("http://www.google.com")
    if err != nil {
        return 0, err
    }
    return resp.StatusCode, nil
}

```

- Создайте файл с именем `transport.go` со следующим содержимым:

```

package rest

import "net/http"

```

```

// APITransport does a SetBasicAuth
// for every request
type APITransport struct {
    *http.Transport
    username, password string
}

// RoundTrip does the basic auth before deferring
to the
// default transport
func (t *APITransport) RoundTrip(req
*http.Request)
(*http.Response, error) {
    req.SetBasicAuth(t.username, t.password)
    return t.Transport.RoundTrip(req)
}

```

- Создайте файл с именем `exec.go` со следующим содержимым:

```

package rest

import "fmt"

// Exec creates an API Client and uses its
// GetGoogle method, then prints the result
func Exec() error {
    c := NewAPIClient("username", "password")

    StatusCode, err := c.GetGoogle()
    if err != nil {
        return err
    }
    fmt.Println("Result of GetGoogle:",
StatusCode)
    return nil
}

```

- Создайте новый каталог с именем `example` и перейдите в него.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import "github.com/PacktPublishing/
Go-Programming-Cookbook-Second-Edition/"

```

```
chapter7/rest"
```

```
func main() {
    if err := rest.Exec(); err != nil {
        panic(err)
    }
}
```

- Выполните `go run main.go`.
- Вы также можете запустить следующую команду:

```
$ go build
$ ./example
```

Вы также можете запустить следующие команды:

```
$ go run main.go
Result of GetGoogle: 200
```

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Этот код демонстрирует, как скрыть такую логику, как проверка подлинности и выполнение обновления токена с помощью интерфейса `Transport`. Он также демонстрирует, как выставить вызов API через метод. Если бы мы реализовывали что-то вроде пользовательского API, мы могли бы ожидать такие методы, как следующие:

```
type API interface{
    GetUsers() (Users, error)
    CreateUser(User) error
    UpdateUser(User) error
    DeleteUser(User)
}
```

Если вы читали [Главу 6](#) «Все о базах данных и хранилищах», это может показаться вам похожим на рецепт, озаглавленный «Выполнение интерфейса транзакций базы данных». Эта композиция через интерфейсы, особенно распространенные интерфейсы, такие как интерфейс `RoundTripper`, обеспечивает большую гибкость для

написания API. Кроме того, может быть полезно написать интерфейс верхнего уровня, как мы делали ранее, и передавать интерфейс, а не напрямую клиенту. Мы рассмотрим это более подробно в следующем рецепте, когда будем изучать написание клиента OAuth2.

## Выполнение параллельных и асинхронных клиентских запросов

Параллельное выполнение клиентских запросов в Go относительно просто. В следующем рецепте мы будем использовать клиент для получения нескольких URL-адресов с использованием буферизованных каналов Go. Ответы и ошибки будут отправляться по отдельному каналу, доступному любому, у кого есть доступ к клиенту.

В случае этого рецепта создание клиента, чтение каналов и обработка ответов и ошибок будут выполняться в файле `main.go`.

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения::

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter7/async` и перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter7/async
```

Вы должны увидеть файл с именем `go.mod`, содержащий следующее:

```
module github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter7/async
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter7/async` или используйте это как упражнение для написания собственного кода!
- Создайте файл с именем `config.go` со следующим содержимым:

```

package async

import "net/http"

// NewClient creates a new client and
// sets its appropriate channels
func NewClient(client *http.Client, bufferSize
int) *Client {
    respch := make(chan *http.Response,
bufferSize)
    errch := make(chan error, bufferSize)
    return &Client{
        Client: client,
        Resp: respch,
        Err: errch,
    }
}

// Client stores a client and has two channels to
aggregate
// responses and errors
type Client struct {
    *http.Client
    Resp chan *http.Response
    Err chan error
}

// AsyncGet performs a Get then returns
// the resp/error to the appropriate channel
func (c *Client) AsyncGet(url string) {
    resp, err := c.Get(url)
    if err != nil {
        c.Err <- err
        return
    }
    c.Resp <- resp
}

```

- Создайте файл с именем **exec.go** со следующим содержимым:

```

package async

// FetchAll grabs a list of urls

```

```
func FetchAll(urls []string, c *Client) {
    for _, url := range urls {
        go c.AsyncGet(url)
    }
}
```

- Создайте новый каталог с именем `example` и перейдите в него.
- Создайте файл с именем `main.go` со следующим содержимым:

```
package main

import (
    "fmt"
    "net/http"

    "github.com/PacktPublishing/
Go-Programming-Cookbook-Second-
Edition/chapter7/async"
)

func main() {
    urls := []string{
        "https://www.google.com",
        "https://golang.org",
        "https://www.github.com",
    }
    c := async.NewClient(http.DefaultClient,
len(urls))
    async.FetchAll(urls, c)

    for i := 0; i < len(urls); i++ {
        select {
            case resp := <-c.Resp:
                fmt.Printf("Status received for %s:
%d\n",
                    resp.Request.URL, resp.StatusCode)
            case err := <-c.Err:
                fmt.Printf("Error received: %s\n", err)
        }
    }
}
```

- Выполните `go run main.go`.
- Вы также можете запустить следующую команду:

```
$ go build
$ ./example
```

Вы также можете запустить следующие команды:

```
$ go run main.go
Status received for https://www.google.com: 200
Status received for https://golang.org: 200
Status received for https://github.com/: 200
```

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Этот рецепт создает структуру для обработки запросов `async` разветвлением с использованием одного клиента. Он попытается получить столько URL-адресов, сколько вы укажете, как можно быстрее. Во многих случаях вы захотите еще больше ограничить это с помощью чего-то вроде рабочего пула. Также может иметь смысл обрабатывать эти `async` горуты вне клиента и для определенных интерфейсов хранения или извлечения.

В этом рецепте также рассматривается использование оператора `case` для включения нескольких каналов. Поскольку выборки выполняются асинхронно, должен быть какой-то механизм ожидания их завершения. В этом случае программа завершится только тогда, когда основная функция прочитает такое же количество ответов и ошибок, сколько было URL-адресов в исходном списке. В таких случаях также важно учитывать, должно ли ваше приложение отключаться по таймауту или есть ли какой-либо другой способ досрочно отменить его работу.

## Использование клиентов OAuth2

OAuth2 — относительно распространенный протокол для взаимодействия с API. Пакет [golang.org/x/oauth2](https://golang.org/x/oauth2) предоставляет довольно гибкий клиент для работы с OAuth2. Он имеет подпакеты, которые определяют конечные точки для различных поставщиков, таких как Facebook, Google и GitHub.

Этот рецепт продемонстрирует, как создать новый клиент GitHub OAuth2 и некоторые из его основных применений.

## Подготовка

После выполнения первоначальных шагов настройки, упомянутых в разделе «*Технические требования*» в начале этой главы, выполните следующие шаги:

- Настройте клиент OAuth по адресу <https://github.com/settings/applications/new>.
- Установите переменные среды с вашим ID клиента и секретом:
  - `export GITHUB_CLIENT="your_client"`
  - `export GITHUB_SECRET="your_secret"`
- Ознакомьтесь с документацией GitHub API по адресу <https://developer.github.com/v3/>.

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения::

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter7/oauthcli` и перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter7/oauthcli
```

Вы должны увидеть файл с именем `go.mod`, содержащий следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter7/oauthcli
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter7/oauthcli` или используйте это как упражнение, чтобы написать собственный код!
- Создайте файл с именем `config.go` со следующим содержимым:

```
package oauthcli
```



```

import (
    "context"
    "fmt"
    "os"

    "golang.org/x/oauth2"
    "golang.org/x/oauth2/github"
)

// Setup return an oauth2Config configured to talk
// to github, you need environment variables set
// for your id and secret
func Setup() *oauth2.Config {
    return &oauth2.Config{
        ClientID: os.Getenv("GITHUB_CLIENT"),
        ClientSecret: os.Getenv("GITHUB_SECRET"),
        Scopes: []string{"repo", "user"},
        Endpoint: github.Endpoint,
    }
}

// GetToken retrieves a github oauth2 token
func GetToken(ctx context.Context, conf
*oauth2.Config)
(*oauth2.Token, error) {
    url := conf.AuthCodeURL("state")
    fmt.Printf("Type the following url into your
browser and
follow the directions on screen: %v\n", url)
    fmt.Println("Paste the code returned in the
redirect URL
and hit Enter:")

    var code string
    if _, err := fmt.Scan(&code); err != nil {
        return nil, err
    }
    return conf.Exchange(ctx, code)
}

```

- Создайте файл с именем `exec.go` со следующим содержимым:

```

package oauthcli

import (
    "fmt"
    "net/http"
)

// GetUsers uses an initialized oauth2 client to
get // information about a user
func GetUser(client *http.Client) error {
    url :=
    fmt.Sprintf("https://api.github.com/user")

    resp, err := client.Get(url)
    if err != nil {
        return err
    }
    defer resp.Body.Close()
    fmt.Println("Status Code from", url, ":",
resp.StatusCode)
    io.Copy(os.Stdout, resp.Body)
    return nil
}

```

- Создайте новый каталог с именем `example` и перейдите в него.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "context"

    "github.com/PacktPublishing/
Go-Programming-Cookbook-Second-Edition/
chapter7/oauthcli"
)

func main() {
    ctx := context.Background()
    conf := oauthcli.Setup()

    tok, err := oauthcli.GetToken(ctx, conf)

```

```

        if err != nil {
            panic(err)
        }
        client := conf.Client(ctx, tok)

        if err := oauthcli.GetUser(client); err != nil
    {
        panic(err)
    }
}

```

- Выполните `go run main.go`.
- Вы также можете запустить следующую команду:

```

$ go build
$ ./example

```

Вы также можете запустить следующие команды:

```

$ go run main.go
Visit the URL for the auth dialog:
https://github.com/login/oauth/authorize?
access_type=offline&client_id=
<your_id>&response_type=code&scope=repo+user&state=state
Paste the code returned in the redirect URL and hit
Enter:
<your_code>
Status Code from https://api.github.com/user: 200
{<json_payload>}

```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Стандартный поток OAuth2 основан на перенаправлении и заканчивается перенаправлением сервера на указанную вами конечную точку. Затем ваш сервер отвечает за захват кода и обмен его на токен. Этот рецепт обходит это требование, позволяя нам использовать URL-

адрес, такой как <https://localhost> или <https://a-domain-you-own>, вручную копируя/вставляя код, а затем нажимая *Enter*. Как только токен был обменян, клиент автоматически обновит токен по мере необходимости.

Важно отметить, что мы никоим образом не храним токен. Если программа дает сбой, она должна произвести повторный обмен на токен. Также важно отметить, что нам нужно явно получить токен только один раз, если срок действия токена обновления не истек, он не потерян или не поврежден. После того как клиент настроен, он должен иметь возможность выполнять все типичные HTTP-операции для API, если во время потока OAuth2 были запрошены соответствующие области. Этот рецепт запрашивает области действия `"repo"` и `"user"`, но при необходимости можно добавить больше или меньше.

## Реализация интерфейса хранения токенов OAuth2

В предыдущем рецепте мы получили токен для нашего клиента и выполнили запросы API. Недостатком этого подхода является то, что у нас нет долговременного хранилища для нашего токена. Например, на HTTP-сервере мы хотели бы иметь постоянное хранилище токена между запросами.

В этом рецепте рассматривается модификация клиента OAuth2 для хранения маркера между запросами и извлечения его по мере необходимости с помощью ключа. Для простоты этот ключ будет файлом, но это также может быть база данных, Redis и так далее.

### Подготовка

См. раздел «Подготовка» в рецепте «Использование клиентов OAuth2».

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения::

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-`

`cookbook/chapter7/oauthstore` и перейдите в этот каталог.

- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-  
Cookbook-Second-Edition/chapter7/oauthstore
```

Вы должны увидеть файл с именем `go.mod`, содержащий следующее:

```
module github.com/PacktPublishing/Go-Programming-  
Cookbook-Second-Edition/chapter7/oauthstore
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter7/oauthstore` или используйте это как упражнение, чтобы написать собственный код!
- Создайте файл с именем `config.go` со следующим содержимым:

```
package oauthstore

import (
    "context"
    "net/http"

    "golang.org/x/oauth2"
)

// Config wraps the default oauth2.Config
// and adds our storage
type Config struct {
    *oauth2.Config
    Storage
}

// Exchange stores a token after retrieval
func (c *Config) Exchange(ctx context.Context,
code string)
(*oauth2.Token, error) {
    token, err := c.Config.Exchange(ctx, code)
    if err != nil {
        return nil, err
    }
    if err := c.Storage.SetToken(token); err !=
nil {
```

```

        return nil, err
    }
    return token, nil
}

// TokenSource can be passed a token which
// is stored, or when a new one is retrieved,
// that's stored
func (c *Config) TokenSource(ctx context.Context,
t
*oauth2.Token) oauth2.TokenSource {
    return StorageTokenSource(ctx, c, t)
}

// Client is attached to our TokenSource
func (c *Config) Client(ctx context.Context, t
*oauth2.Token)
*http.Client {
    return oauth2.NewClient(ctx,
c.TokenSource(ctx, t))
}

```

- Создайте файл с именем `tokensource.go` со следующим содержимым:

```

package oauthstore

import (
    "context"

    "golang.org/x/oauth2"
)

type storageTokenSource struct {
    *Config
    oauth2.TokenSource
}

// Token satisfies the TokenSource interface
func (s *storageTokenSource) Token()
(*oauth2.Token, error) {
    if token, err := s.Config.Storage.GetToken();
err == nil &&

```

```

        token.Valid() {
            return token, err
        }
        token, err := s.TokenSource.Token()
        if err != nil {
            return token, err
        }
        if err := s.Config.Storage.SetToken(token);
err != nil {
            return nil, err
        }
        return token, nil
    }

    // StorageTokenSource will be used by out configs
TokenSource
    // function
    func StorageTokenSource(ctx context.Context, c
*Config, t
*oauth2.Token) oauth2.TokenSource {
        if t == nil || !t.Valid() {
            if tok, err := c.Storage.GetToken(); err
== nil {
                t = tok
            }
        }
        ts := c.Config.TokenSource(ctx, t)
        return &storageTokenSource{c, ts}
    }

```

- Создайте файл с именем `storage.go` со следующим содержимым:

```

package oauthstore

import (
    "context"
    "fmt"

    "golang.org/x/oauth2"
)

// Storage is our generic storage interface
type Storage interface {

```

```

        GetToken() (*oauth2.Token, error)
        SetToken(*oauth2.Token) error
    }

    // GetToken retrieves a github oauth2 token
    func GetToken(ctx context.Context, conf Config)
(*oauth2.Token,
error) {
    token, err := conf.Storage.GetToken()
    if err == nil && token.Valid() {
        return token, err
    }
    url := conf.AuthCodeURL("state")
    fmt.Printf("Type the following url into your
browser and
follow the directions on screen: %v\n", url)
    fmt.Println("Paste the code returned in the
redirect URL
and hit Enter:")

    var code string
    if _, err := fmt.Scan(&code); err != nil {
        return nil, err
    }
    return conf.Exchange(ctx, code)
}

```

- Создайте файл с именем `filestorage.go` со следующим содержимым:

```

package oauthstore

import (
    "encoding/json"
    "errors"
    "os"
    "sync"

    "golang.org/x/oauth2"
)

// FileStorage satisfies our storage interface
type FileStorage struct {

```



```

        Path string
        mu sync.RWMutex
    }

    // GetToken retrieves a token from a file
    func (f *FileStorage) GetToken() (*oauth2.Token,
error) {
        f.mu.RLock()
        defer f.mu.RUnlock()
        in, err := os.Open(f.Path)
        if err != nil {
            return nil, err
        }
        defer in.Close()
        var t *oauth2.Token
        data := json.NewDecoder(in)
        return t, data.Decode(&t)
    }

    // SetToken creates, truncates, then stores a
token
    // in a file
    func (f *FileStorage) SetToken(t *oauth2.Token)
error {
        if t == nil || !t.Valid() {
            return errors.New("bad token")
        }

        f.mu.Lock()
        defer f.mu.Unlock()
        out, err := os.OpenFile(f.Path,
os.O_RDWR|os.O_CREATE|os.O_TRUNC, 0755)
        if err != nil {
            return err
        }
        defer out.Close()
        data, err := json.Marshal(&t)
        if err != nil {
            return err
        }

        _, err = out.Write(data)

```

```
        return err
    }
}
```

- Создайте новый каталог с именем `example` и перейдите в него.
- Создайте файл с именем `main.go` со следующим содержимым:

```
package main

import (
    "context"
    "io"
    "os"

    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter7/oauthstore"

    "golang.org/x/oauth2"
    "golang.org/x/oauth2/github"
)

func main() {
    conf := oauthstore.Config{
        Config: &oauth2.Config{
            ClientID: os.Getenv("GITHUB_CLIENT"),
            ClientSecret:
os.Getenv("GITHUB_SECRET"),
            Scopes: []string{"repo", "user"},
            Endpoint: github.Endpoint,
        },
        Storage: &oauthstore.FileStorage{Path:
"token.txt"},
    }
    ctx := context.Background()
    token, err := oauthstore.GetToken(ctx, conf)
    if err != nil {
        panic(err)
    }

    cli := conf.Client(ctx, token)
    resp, err :=
cli.Get("https://api.github.com/user")
    if err != nil {
```

```

        panic(err)
    }
    defer resp.Body.Close()
    io.Copy(os.Stdout, resp.Body)
}

```

- Выполните `go run main.go`.
- Вы также можете запустить следующую команду:

```

$ go build
$ ./example

```

Теперь вы должны увидеть следующий вывод:

```

$ go run main.go
Visit the URL for the auth dialog:
https://github.com/login/oauth/authorize?
access_type=offline&client_id=
<your_id>&response_type=code&scope=repo+user&state=state
Paste the code returned in the redirect URL and hit
Enter:
<your_code>
{<json_payload>}

```

```

$ go run main.go
{<json_payload>}

```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Этот рецепт заботится о сохранении и извлечении содержимого токена в/из файла. Если это первый запуск, он должен выполнить весь обмен кодом, но последующие запуски будут повторно использовать маркер доступа, и, если он доступен, он будет обновляться с использованием маркера обновления.

В настоящее время в этом коде нет способа различать пользователей/токены, но это можно сделать с помощью файлов cookie в качестве

ключа для имени файла или строки в базе данных. Давайте рассмотрим, что делает этот код:

- Файл `config.go` содержит стандартную конфигурацию OAuth2. Для каждого метода, включающего получение токена, мы сначала проверяем, есть ли у нас допустимый токен в локальном хранилище. Если нет, мы извлекаем его с помощью стандартной конфигурации, а затем сохраняем.
- Файл `tokensource.go` реализует наш пользовательский интерфейс `TokenSource`, который сочетается с `Config`. Как и в случае с `Config`, мы всегда сначала пытаемся получить наш токен из файла; в противном случае мы устанавливаем его с новым токеном.
- Файл `storage.go` — это интерфейс `storage`, используемый `Config` и `TokenSource`. Он определяет только два метода, и мы также включаем вспомогательную функцию для начальной загрузки потока на основе кода OAuth2, аналогичного тому, что мы делали в предыдущем рецепте, но если файл с действительным токеном уже существует, он будет использоваться вместо него.
- Файл `filestorage.go` реализует интерфейс `storage`. Когда мы сохраняем новый токен, мы сначала усекаем файл и пишем JSON-представление структуры `token`. В противном случае мы декодируем файл и возвращаем `token`.

## Обертывание клиента дополнительным функционалом и функциональной композицией

В 2015 году Томас Сенарт выступил с отличным докладом о том, как обернуть структуру `http.Client` интерфейсом, что позволит вам воспользоваться промежуточным программным обеспечением и композицией функций. Вы можете узнать больше об этом на <https://github.com/gophercon/2015-talks>. Этот рецепт заимствован из его идей и демонстрирует пример выполнения того же действия в интерфейсе

Transport структуры `http.Client` аналогично нашему предыдущему рецепту «*Написание клиента для REST API*».

Следующий рецепт реализует ведение журнала и базовое промежуточное ПО для аутентификации для стандартной структуры `http.Client`. Он также включает функцию `decorate`, которую можно использовать при необходимости с большим количеством промежуточного программного обеспечения.

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения::

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter7/decorator` и перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter7/decorator
```

Вы должны увидеть файл с именем `go.mod`, содержащий следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter7/decorator
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter7/decorator` или используйте это как упражнение, чтобы написать собственный код!
- Создайте файл с именем `config.go` со следующим содержимым:

```
package decorator

import (
    "log"
    "net/http"
    "os"
)

// Setup initializes our ClientInterface
func Setup() *http.Client {
    c := http.Client{}
```

```

        t := Decorate(&http.Transport{},
            Logger(log.New(os.Stdout, "", 0)),
            BasicAuth("username", "password"),
        )
        c.Transport = t
        return &c
    }

```

- Создайте файл с именем `decorator.go` со следующим содержимым:

```

package decorator

import "net/http"

// TransportFunc implements the RoundTripper
interface {
    type TransportFunc func(*http.Request)
    (*http.Response, error)

    // RoundTrip just calls the original function
    func (tf TransportFunc) RoundTrip(r *http.Request)
    (*http.Response, error) {
        return tf(r)
    }

    // Decorator is a convenience function to
    represent our
    // middleware inner function
    type Decorator func(http.RoundTripper)
    http.RoundTripper

    // Decorate is a helper to wrap all the middleware
    func Decorate(t http.RoundTripper, rts
    ...Decorator)
    http.RoundTripper {
        decorated := t
        for _, rt := range rts {
            decorated = rt(decorated)
        }
        return decorated
    }
}

```

- Создайте файл с именем `middleware.go` со следующим содержимым:

```
package decorator

import (
    "log"
    "net/http"
    "time"
)

// Logger is one of our 'middleware' decorators
func Logger(l *log.Logger) Decorator {
    return func(c http.RoundTripper)
http.RoundTripper {
        return TransportFunc(func(r *http.Request)
(*http.Response, error) {
            start := time.Now()
            l.Printf("started request to %s at %s",
r.URL,
                start.Format("2006-01-02 15:04:05"))
            resp, err := c.RoundTrip(r)
            l.Printf("completed request to %s in
%s", r.URL,
                time.Since(start))
            return resp, err
        })
    }
}

// BasicAuth is another of our 'middleware'
decorators
func BasicAuth(username, password string)
Decorator {
    return func(c http.RoundTripper)
http.RoundTripper {
        return TransportFunc(func(r *http.Request)
(*http.Response, error) {
            r.SetBasicAuth(username, password)
            resp, err := c.RoundTrip(r)
            return resp, err
        })
    }
}
```

```
    }
}
```

- Создайте файл с именем `exec.go` со следующим содержимым:

```
package decorator

import "fmt"

// Exec creates a client, calls google.com
// then prints the response
func Exec() error {
    c := Setup()

    resp, err := c.Get("https://www.google.com")
    if err != nil {
        return err
    }
    fmt.Println("Response code:", resp.StatusCode)
    return nil
}
```

- Создайте новый каталог с именем `example` и перейдите в него.
- Создайте файл с именем `main.go` со следующим содержимым:

```
package main

import "github.com/PacktPublishing/
Go-Programming-Cookbook-Second-Edition/
chapter7/decorator"

func main() {
    if err := decorator.Exec(); err != nil {
        panic(err)
    }
}
```

- Выполните `go run main.go`.
- Вы также можете запустить следующую команду:

```
$ go build
$ ./example
```

Теперь вы должны увидеть следующий вывод:



```
$ go run main.go
started request to https://www.google.com at 2017-01-01
13:38:42
completed request to https://www.google.com in
194.013054ms
Response code: 200
```

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Этот рецепт использует преимущества замыканий как первоклассных граждан и интерфейсов. Основная хитрость в достижении этого заключается в том, чтобы функция реализовывала интерфейс. Это позволяет нам обернуть интерфейс, реализованный структурой, с интерфейсом, реализованным функцией.

Файл `middleware.go` содержит два примера клиентских функций промежуточного ПО. Они могут быть расширены, чтобы содержать дополнительное промежуточное программное обеспечение, такое как более сложная аутентификация и метрики. Этот рецепт также можно комбинировать с предыдущим рецептом для создания клиента OAuth2, который можно расширить за счет дополнительного ПО промежуточного слоя.

Функция `Decorator` — это удобная функция, которая позволяет:

```
Decorate(RoundTripper, Middleware1, Middleware2, etc)
```

vs

```
var t RoundTripper
t = Middleware1(t)
t = Middleware2(t)
etc
```

Преимущество этого подхода по сравнению с оболочкой клиента заключается в том, что мы можем сохранить разреженный интерфейс. Если вам нужен полнофункциональный клиент, вам также потребуется реализовать такие методы, как `GET`, `POST` и `PostForm`.

# Понимание клиентов GRPC

GRPC — это высокопроизводительная платформа RPC, построенная с использованием буферов протоколов (<https://developers.google.com/protocol-buffers>) и HTTP/2 (<https://http2.github.io>). Создание клиента GRPC в Go связано со многими из тех же сложностей, что и работа с HTTP-клиентами Go. Чтобы продемонстрировать базовое использование клиента, проще всего также реализовать сервер. Этот рецепт создаст службу `greeter`, которая принимает приветствие и имя и возвращает предложение `<greeting> <name>!`. Кроме того, сервер может указать, следует ли восклицать `!` или не `.` (полная остановка).

Есть некоторые детали GRPC, такие как потоковая передача, которые в этом рецепте не рассматриваются; тем не менее, мы надеемся, что он послужит введением в создание очень простого сервера и клиента.

## Подготовка

После выполнения начальных шагов настройки, упомянутых в разделе «Технические требования» в начале этой главы, установите GRPC (<https://grpc.io/docs/quickstart/go/>) и выполните следующие команды:

- `go get -u github.com/golang/protobuf/{proto,protoc-gen-go}`
- `go get -u google.golang.org/grpc`

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения::

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter7/grpc` и перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter7/grpc
```

Вы должны увидеть файл с именем `go.mod`, содержащий следующее:

`module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter7/grpc`

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter7/grpc` или используйте это как упражнение, чтобы написать собственный код!
- Создайте каталог с именем `greeter` и перейдите к нему.
- Создайте файл с именем `greeter.proto` со следующим содержимым:

```

syntax = "proto3";

package greeter;

service GreeterService{
    rpc Greet(GreetRequest) returns
(GreetResponse) {}
}

message GreetRequest {
    string greeting = 1;
    string name = 2;
}

message GreetResponse{
    string response = 1;
}

```

- Перейдите обратно в каталог `grpc`.
- Выполните следующую команду:

**`$ protoc --go_out=plugins=grpc:. greeter/greeter.proto`**

- Создайте новый каталог с именем `server` и перейдите к нему.
- Создайте файл `greeter.go` со следующим содержимым. Убедитесь, что вы изменили импорт `greeter`, чтобы использовать путь, который вы установили на шаге 3:

```

package main

import (
    "fmt"

```

```

        "github.com/PacktPublishing/
        Go-Programming-Cookbook-Second-Edition/
        chapter7/grpc/greeter"
        "golang.org/x/net/context"
    )

    // Greeter implements the interface
    // generated by protoc
    type Greeter struct {
        Exclaim bool
    }

    // Greet implements grpc Greet
    func (g *Greeter) Greet(ctx context.Context, r
    *greeter.GreetRequest) (*greeter.GreetResponse,
    error) {
        msg := fmt.Sprintf("%s %s", r.GetGreeting(),
        r.GetName())
        if g.Exclaim {
            msg += "!"
        } else {
            msg += "."
        }
        return &greeter.GreetResponse{Response: msg},
        nil
    }

```

- Создайте файл **server.go** со следующим содержимым. Убедитесь, что вы изменили импорт приветствия, чтобы использовать путь, который вы установили на шаге 3:

```

package main

import (
    "fmt"
    "net"

    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter7/grpc/greeter"
    "google.golang.org/grpc"
)

```

```

func main() {
    grpcServer := grpc.NewServer()

    greeter.RegisterGreeterServiceServer(grpcServer,
        &Greeter{Exclaim: true})
    lis, err := net.Listen("tcp", ":4444")
    if err != nil {
        panic(err)
    }
    fmt.Println("Listening on port :4444")
    grpcServer.Serve(lis)
}

```

- Перейдите обратно в каталог `grpc`.
- Создайте новый каталог с именем `client` и перейдите в него.
- Создайте файл `client.go` со следующим содержимым. Убедитесь, что вы изменили импорт приветствия, чтобы использовать путь, который вы установили на шаге 3:

```

package main

import (
    "context"
    "fmt"

    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter7/grpc/greeter"
    "google.golang.org/grpc"
)

func main() {
    conn, err := grpc.Dial(":4444",
    grpc.WithInsecure())
    if err != nil {
        panic(err)
    }
    defer conn.Close()

    client :=
    greeter.NewGreeterServiceClient(conn)

    ctx := context.Background()

```

```

Name:      req := greeter.GreetRequest{Greeting: "Hello",
           "Reader"}
           resp, err := client.Greet(ctx, &req)
           if err != nil {
               panic(err)
           }
           fmt.Println(resp)

           req.Greeting = "Goodbye"
           resp, err = client.Greet(ctx, &req)
           if err != nil {
               panic(err)
           }
           fmt.Println(resp)
       }

```

- Перейдите обратно в каталог `grpc`.
- Запустите `go run ./server`, и вы увидите следующий вывод:

```

$ go run ./server
Listening on port :4444

```

- В отдельном терминале запустите `go run ./client` из каталога `grpc`, и вы увидите следующий вывод:

```

$ go run ./client
response:"Hello Reader!"
response:"Goodbye Reader!"

```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Сервер GRPC настроен на прослушивание порта `4444`. После подключения клиент может отправлять запросы и получать ответы от сервера. Структура запросов, ответов и поддерживаемых методов определяется файлом `.proto`, который мы создали на *шаге 4*. На практике при интеграции с серверами GRPC они должны

предоставлять файл `.proto`, который можно использовать для автоматического создания клиента.

В дополнение к клиенту команда `protoc` генерирует заглушки для сервера, и все, что требуется, — это заполнить детали реализации. Сгенерированный код Go также содержит теги JSON, и те же структуры можно повторно использовать для служб JSON REST. Наш код устанавливает небезопасный клиент. Для безопасной обработки GRPC вам необходимо использовать сертификат SSL.

## Использование twitchtv/twirp для RPC

Платформа `twitchtv/twirp` RPC обладает многими преимуществами GRPC, включая построение моделей с буферами протоколов (<https://developers.google.com/protocol-buffers>) и позволяет обмениваться данными через HTTP 1.1. Он также может взаимодействовать с помощью JSON, поэтому можно использовать команду `curl` для связи со службой `twirp` RPC. Этот рецепт реализует тот же `greeter`, что и предыдущий раздел GRPC. Эта служба принимает приветствие и имя и возвращает предложение `<greeting> <name>!`. Кроме того, сервер может указать, следует ли восклицать `!` или не `..`

В этом рецепте не рассматриваются другие функции `twitchtv/twirp`, а основное внимание уделяется базовому взаимодействию клиент-сервер. Для получения дополнительной информации о том, что поддерживается, посетите их страницу GitHub (<https://github.com/twitchtv/twirp>).

### Подготовка

После выполнения первоначальных шагов по настройке, упомянутых в разделе «Технические требования» в начале этой главы, установите `twirp` <https://twitchtv.github.io/twirp/docs/install.html> и выполните следующие команды:

- `go get -u github.com/golang/protobuf/{proto,protoc-gen-go}`
- `go get github.com/twitchtv/twirp/protoc-gen-twirp`

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter7/twirp` и перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter7/twirp
```

Вы должны увидеть файл с именем `go.mod`, содержащий следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter7/twirp
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter7/twirp` или используйте это как упражнение, чтобы написать собственный код!
- Создайте каталог с именем `rpc/greeter` и перейдите к нему.
- Создайте файл с именем `greeter.proto` со следующим содержимым:

```
    syntax = "proto3";

    package greeter;

    service GreeterService{
        rpc Greet(GreetRequest) returns
        (GreetResponse) {}
    }

    message GreetRequest {
        string greeting = 1;
        string name = 2;
    }

    message GreetResponse{
        string response = 1;
    }
```



- Перейдите обратно в каталог `twirp`.
- Выполните следующую команду:

```
$ protoc --proto_path=$GOPATH/src:. --twirp_out=. --go_out=. ./rpc/greeter/greeter.proto
```

- Создайте новый каталог с именем `server` и перейдите к нему.
- Создайте файл `greeter.go` со следующим содержимым. Убедитесь, что вы изменили импорт `greeter`, чтобы использовать путь, который вы установили на шаге 3:

```
package main

import (
    "context"
    "fmt"

    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter7/twirp/rpc/greeter"
)

// Greeter implements the interface
// generated by protoc
type Greeter struct {
    Exclaim bool
}

// Greet implements twirp Greet
func (g *Greeter) Greet(ctx context.Context, r
*greeter.GreetRequest) (*greeter.GreetResponse, error) {
    msg := fmt.Sprintf("%s %s", r.GetGreeting(),
r.GetName())
    if g.Exclaim {
        msg += "!"
    } else {
        msg += "."
    }
    return &greeter.GreetResponse{Response: msg}, nil
}
```

- Создайте файл `server.go` со следующим содержимым. Убедитесь, что вы изменили импорт `greeter`, чтобы использовать путь,

который вы установили на шаге 3:

```
package main

import (
    "fmt"
    "net/http"

    "github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter7/twirp/rpc/greeter"
)

func main() {
    server := &Greeter{}
    twirpHandler := greeter.NewGreeterServiceServer(server, nil)

    fmt.Println("Listening on port :4444")
    http.ListenAndServe(":4444", twirpHandler)
}
```

- Перейдите обратно в каталог `twirp`.
- Создайте новый каталог с именем `client` и перейдите в него.
- Создайте файл `client.go` со следующим содержимым. Убедитесь, что вы изменили импорт `greeter`, чтобы использовать путь, который вы установили на шаге 3:

```
package main

import (
    "context"
    "fmt"
    "net/http"

    "github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter7/twirp/rpc/greeter"
)

func main() {
    // you can put in a custom client for tighter controls
    // on timeouts etc.
    client :=
    greeter.NewGreeterServiceProtobufClient("http://localhost
```

```

:4444", &http.Client{})

    ctx := context.Background()
    req := greeter.GreetRequest{Greeting: "Hello", Name:
"Reader"}
    resp, err := client.Greet(ctx, &req)
    if err != nil {
        panic(err)
    }
    fmt.Println(resp)

    req.Greeting = "Goodbye"
    resp, err = client.Greet(ctx, &req)
    if err != nil {
        panic(err)
    }
    fmt.Println(resp)
}

```

- Перейдите обратно в каталог `twirp`.
- Запустите `go run ./server`, и вы увидите следующий вывод:

```

$ go run ./server
Listening on port :4444

```

- В отдельном терминале запустите `go run ./client` из каталога `twirp`. Вы должны увидеть следующий вывод:

```

$ go run ./client
response:"Hello Reader."
response:"Goodbye Reader."

```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Мы настроили RPC-сервер `twitchev/twirp` для прослушивания порта `4444`. Как и GRPC, протокол `protoc` можно использовать для создания

клиентов для ряда языков и, например, для создания документации Swagger (<https://swagger.io/>).

Как и GRPC, мы сначала определяем наши модели как файлы `.proto`, генерируем привязки Go и, наконец, реализуем сгенерированный интерфейс. Благодаря использованию файлов `.proto` код относительно переносим между GRPC и `twichtv/twirp`, если вы не полагаетесь на более продвинутые функции любой из платформ.

Кроме того, поскольку сервер `twichtv/twirp` поддерживает HTTP 1.1, мы можем `curl` его следующим образом:

```
$ curl --request "POST" \
  --location
"http://localhost:4444/twirp/greeter.GreeterService/Greet" \
  --header "Content-Type:application/json" \
  --data '{"greeting": "Greetings to", "name":"you"}'

{"response":"Greetings to you."}
```

## 8. Микросервисы для приложений в Go

Из коробки Go — отличный выбор для написания веб-приложений. Встроенные пакеты `net/http` в сочетании с такими пакетами, как `html/template`, позволяют создавать полнофункциональные современные веб-приложения из коробки. Это настолько просто, что поощряет развертывание веб-интерфейсов для управления даже базовыми приложениями, которые долго работают. Хотя стандартная библиотека полнофункциональна, по-прежнему существует множество сторонних веб-пакетов для всего, от маршрутов до фреймворков с полным стекком, включая следующие:

- <https://github.com/urfave/negroni>
- <https://github.com/gin-gonic/gin>
- <https://github.com/labstack/echo>
- <http://www.gorillatoolkit.org/>
- <https://github.com/julienschmidt/httprouter>

Рецепты в этой главе будут сосредоточены на основных задачах, с которыми вы можете столкнуться при работе с обработчиками, при навигации по объектам ответа и запроса и при работе с такими понятиями, как промежуточное ПО.

В этой главе будут рассмотрены следующие рецепты:

- Работа с веб-обработчиками, запросами и экземплярами `ResponseWriter`
- Использование структур и замыканий для обработчиков состояния
- Проверка входных данных для структур Go и пользовательских входных данных
- Рендеринг и согласование контента
- Внедрение и использование промежуточного ПО
- Создание обратного прокси-приложения
- Экспорт GRPC как JSON API

## Технические требования

Чтобы продолжить выполнение всех рецептов в этой главе, настройте свою среду в соответствии со следующими шагами:

- Загрузите и установите Go 1.12.6 или более позднюю версию в своей операционной системе по адресу <https://golang.org/doc/install>.
- Откройте терминал или консольное приложение, создайте и перейдите в каталог проекта, например `~/projects/go-programming-cookbook`. Весь наш код будет запускаться и изменяться из этого каталога.
- Скопируйте последний код в `~/projects/go-programming-cookbook-original` и, при желании, работайте из этого каталога, вместо того, чтобы вводить примеры вручную:

```
$ git clone git@github.com:PacktPublishing/Go-Programming-Cookbook-Second-Edition.git go-programming-cookbook-original
```

- Установите команду `curl` с <https://curl.haxx.se/download.html>.

## Работа с веб-обработчиками, запросами и экземплярами ResponseWriter

Go определяет `HandlerFunc` и интерфейс `Handler` со следующими сигнатурами:

```
// HandlerFunc implements the Handler interface
type HandlerFunc func(http.ResponseWriter, *http.Request)

type Handler interface {
    ServeHTTP(http.ResponseWriter, *http.Request)
}
```

По умолчанию пакет `net/http` широко использует эти типы. Например, маршрут может быть присоединен к интерфейсу `Handler` или `HandlerFunc`. В этом рецепте рассматривается создание интерфейса `Handler`, прослушивание локального порта и выполнение некоторых операций с интерфейсом `http.ResponseWriter` после обработки `http.Request`. Это следует рассматривать как основу для веб-приложений Go и RESTful API.

### Как это сделать...

Следующие шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter8/handlers` и перейдите в этот каталог.

- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter8/handlers
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter8/handlers
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter8/handlers` или используйте это как упражнение для написания собственного кода!
- Создайте файл с именем `get.go` со следующим содержимым:

```
package handlers

import (
    "fmt"
    "net/http"
)

// HelloHandler takes a GET parameter "name" and
responds
// with Hello <name>! in plaintext
func HelloHandler(w http.ResponseWriter, r
*http.Request) {
    w.Header().Set("Content-Type", "text/plain")
    if r.Method != http.MethodGet {
        w.WriteHeader(http.StatusMethodNotAllowed)
        return
    }
    name := r.URL.Query().Get("name")

    w.WriteHeader(http.StatusOK)
    w.Write([]byte(fmt.Sprintf("Hello %s!", name)))
}
```

- Создайте файл с именем `post.go` со следующим содержимым:

```
package handlers

import (
    "encoding/json"
    "net/http"
```

```

    )

    // GreetingResponse is the JSON Response that
    // GreetingHandler returns
    type GreetingResponse struct {
        Payload struct {
            Greeting string `json:"greeting,omitempty"`
            Name string `json:"name,omitempty"`
            Error string `json:"error,omitempty"`
        } `json:"payload"`
        Successful bool `json:"successful"`
    }

    // GreetingHandler returns a GreetingResponse which
    either has
    // errors or a useful payload
    func GreetingHandler(w http.ResponseWriter, r
    *http.Request) {
        w.Header().Set("Content-Type", "application/json")
        if r.Method != http.MethodPost {
            w.WriteHeader(http.StatusMethodNotAllowed)
            return
        }
        var gr GreetingResponse
        if err := r.ParseForm(); err != nil {
            gr.Payload.Error = "bad request"
            if payload, err := json.Marshal(gr); err ==
            nil {
                w.Write(payload)
            } else if err != nil {
                w.WriteHeader(http.StatusInternalServerError)
            }
            name := r.FormValue("name")
            greeting := r.FormValue("greeting")

            w.WriteHeader(http.StatusOK)
            gr.Successful = true
            gr.Payload.Name = name
            gr.Payload.Greeting = greeting
            if payload, err := json.Marshal(gr); err == nil {
                w.Write(payload)
            }
        }
    }

```



- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```
package main

import (
    "fmt"
    "net/http"

    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    $ chapter8/handlers"
)

func main() {
    http.HandleFunc("/name", handlers.HelloHandler)
    http.HandleFunc("/greeting",
handlers.GreetingHandler)
    fmt.Println("Listening on port :3333")
    err := http.ListenAndServe(":3333", nil)
    panic(err)
}
```

- Выполните `go run main.go`.
- Вы также можете запустить следующую команду:

```
$ go build
$ ./example
```

Вы должны увидеть следующий вывод:

```
$ go run main.go
Listening on port :3333
```

- В отдельном терминале выполните следующие команды:

```
$ curl "http://localhost:3333/name?name=Reader" -X GET
$ curl "http://localhost:3333/greeting" -X POST -d
'name=Reader;greeting=Goodbye'
```

Вы должны увидеть следующий вывод:

```
$ curl "http://localhost:3333/name?name=Reader" -X GET
Hello Reader!
```

```
$ curl "http://localhost:3333/greeting" -X POST -d
'name=Reader;greeting=Goodbye'
```

```
{"payload":  
{"greeting":"Goodbye","name":"Reader"},"successful":true}
```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Для этого рецепта мы настроили два обработчика. Первый обработчик ожидает запрос `GET` с параметром `GET` с именем `name`. Когда мы `curl` его, он возвращает простую текстовую строку `Hello <name>!`.

Второй обработчик ожидает метод `POST` с запросами `PostForm`. Это то, что вы получите, если будете использовать стандартную HTML-форму без каких-либо вызовов AJAX. В качестве альтернативы мы могли бы вместо этого проанализировать JSON из тела запроса. Обычно это делается с помощью `json.Decoder`. Я рекомендую попробовать это в качестве упражнения. Наконец, обработчик отправляет ответ в формате JSON и устанавливает все соответствующие заголовки.

Хотя все это было написано явно, существует ряд способов сделать код менее подробным, в том числе следующие:

- Использование <https://github.com/unrolled/render> для обработки ответов
- Использование различных веб-фреймворков, упомянутых в рецепте «Работа с веб-обработчиками, запросами и `ResponseWriters`» этой главы, для анализа аргументов маршрута, ограничения маршрутов определенными HTTP-командами, обработки корректного завершения работы и многого другого

## Использование структур и замыканий для обработчиков состояния

Из-за редких сигнатур функций обработчика HTTP добавление состояния к обработчику может показаться сложным. Например, есть множество способов включить соединение с базой данных. Два подхода к этому — передать состояние через замыкания, что полезно для достижения гибкости в одном обработчике, или с помощью структуры.

Этот рецепт продемонстрирует и то, и другое. Мы будем использовать контроллер структуры для хранения интерфейса хранилища и создания двух маршрутов с одним обработчиком, которые изменяются внешней функцией.

## Как это сделать...

Следующие шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter8/controllers` и перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter8/controllers
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter8/controllers
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter8/controllers` или используйте это как упражнение для написания собственного кода!
- Создайте файл с именем `controller.go` со следующим содержимым:

```
package controllers

// Controller passes state to our handlers
type Controller struct {
    storage Storage
}

// New is a Controller 'constructor'
func New(storage Storage) *Controller {
    return &Controller{
        storage: storage,
    }
}

// Payload is our common response
type Payload struct {
```

```
    Value string `json:"value"`
}
```

- Создайте файл с именем `storage.go` со следующим содержимым:

```
package controllers

// Storage Interface Supports Get and Put
// of a single value
type Storage interface {
    Get() string
    Put(string)
}

// MemStorage implements Storage
type MemStorage struct {
    value string
}

// Get our in-memory value
func (m *MemStorage) Get() string {
    return m.value
}

// Put our in-memory value
func (m *MemStorage) Put(s string) {
    m.value = s
}
```

- Создайте файл с именем `post.go` со следующим содержимым:

```
package controllers

import (
    "encoding/json"
    "net/http"
)

// SetValue modifies the underlying storage of the
controller
// object
func (c *Controller) SetValue(w http.ResponseWriter, r
    *http.Request) {
    if r.Method != http.MethodPost {
        w.WriteHeader(http.StatusMethodNotAllowed)
        return
    }
}
```

```

        if err := r.ParseForm(); err != nil {
            w.WriteHeader(http.StatusInternalServerError)
            return
        }
        value := r.FormValue("value")
        c.storage.Put(value)
        w.WriteHeader(http.StatusOK)
        p := Payload{Value: value}
        if payload, err := json.Marshal(p); err == nil {
            w.Write(payload)
        } else if err != nil {
            w.WriteHeader(http.StatusInternalServerError)
        }
    }
}

```

- Создайте файл с именем `get.go` со следующим содержимым:

```

package controllers

import (
    "encoding/json"
    "net/http"
)

// GetValue is a closure that wraps a HandlerFunc, if
// UseDefault is true value will always be "default"
else it'll
// be whatever is stored in storage
func (c *Controller) GetValue(UseDefault bool)
http.HandlerFunc
{
    return func(w http.ResponseWriter, r
    *http.Request) {
        w.Header().Set("Content-Type",
        "application/json")
        if r.Method != http.MethodGet {
            w.WriteHeader(http.StatusMethodNotAllowed)
            return
        }
        value := "default"
        if !UseDefault {
            value = c.storage.Get()
        }
        w.WriteHeader(http.StatusOK)
        p := Payload{Value: value}
    }
}

```

```

        if payload, err := json.Marshal(p); err == nil
    {
        w.Write(payload)
    }
}

```

- Создайте новый каталог с именем **example** и перейдите к нему.
- Создайте файл с именем **main.go** со следующим содержимым:

```

package main

import (
    "fmt"
    "net/http"

    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter8/controllers"
)

func main() {
    storage := controllers.MemStorage{}
    c := controllers.New(&storage)
    http.HandleFunc("/get", c.GetValue(false))
    http.HandleFunc("/get/default", c.GetValue(true))
    http.HandleFunc("/set", c.SetValue)

    fmt.Println("Listening on port :3333")
    err := http.ListenAndServe(":3333", nil)
    panic(err)
}

```

- Выполните **go run main.go**.
- Вы также можете запустить следующее:

```

$ go build
$ ./example

```

Вы должны увидеть следующий вывод:

```

$ go run main.go
Listening on port :3333

```

- В отдельном терминале выполните следующие команды:

```
$ curl "http://localhost:3333/set" -X POST -d "value=value"
$ curl "http://localhost:3333/get" -X GET
$ curl "http://localhost:3333/get/default" -X GET
```

Вы должны увидеть следующий вывод:

```
$ curl "http://localhost:3333/set" -X POST -d "value=value"
{"value":"value"}
```

```
$ curl "http://localhost:3333/get" -X GET
{"value":"value"}
```

```
$ curl "http://localhost:3333/get/default" -X GET
{"value":"default"}
```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Эти стратегии работают, потому что Go позволяет методам удовлетворять типизированным функциям, таким как `http.HandlerFunc`. Используя структуру, мы можем внедрять в `main.go` различные элементы, которые могут включать соединения с базой данных, ведение журналов и многое другое. В этом рецепте мы вставили интерфейс `Storage`. Все обработчики, подключенные к контроллеру, могут использовать его методы и атрибуты.

Метод `GetValue` не имеет подписи `http.HandlerFunc` и вместо этого возвращает ее. Вот как мы можем использовать замыкание для внедрения состояния. В `main.go` мы определяем два маршрута — один с `UseDefault`, установленным в `false`, а другой с установленным в `true`. Это можно использовать при определении функции, которая охватывает несколько маршрутов, или при использовании структуры, где ваши обработчики кажутся слишком громоздкими.

## Проверка входных данных для структур Go и пользовательских входных данных

Проверка для Интернета может быть проблемой. В этом рецепте будет рассмотрено использование замыканий для поддержки простой имитации функций проверки и обеспечения гибкости в типе проверки, выполняемой при инициализации структуры контроллера, как описано в предыдущем рецепте.

Мы выполним эту проверку структуры, но не будем исследовать, как заполнить структуру. Мы можем предположить, что данные будут заполняться путем анализа полезной нагрузки JSON, заполнения явно из ввода формы или других методов.

## Как это сделать...

Следующие шаги охватывают написание и запуск вашего приложения:

- В своем терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter8/validation` и перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter8/validation
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter8/validation
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter8/validation` или используйте это как упражнение для написания собственного кода!
- Создайте файл с именем `controller.go` со следующим содержимым:

```
package validation

// Controller holds our validation functions
type Controller struct {
    ValidatePayload func(p *Payload) error
}

// New initializes a controller with our
// local validation, it can be overwritten
func New() *Controller {
    return &Controller{
```



```

        ValidatePayload: ValidatePayload,
    }
}

```

- Создайте файл с именем `validate.go` со следующим содержимым:

```

package validation

import "errors"

// Verror is an error that occurs
// during validation, we can
// return this to a user
type Verror struct {
    error
}

// Payload is the value we
// process
type Payload struct {
    Name string `json:"name"`
    Age int `json:"age"`
}

// ValidatePayload is 1 implementation of
// the closure in our controller
func ValidatePayload(p *Payload) error {
    if p.Name == "" {
        return Verror{errors.New("name is required")}
    }

    if p.Age <= 0 || p.Age >= 120 {
        return Verror{errors.New("age is required and
must be a
        value greater than 0 and less than 120")}
    }
    return nil
}

```

- Создайте файл с именем `process.go` со следующим содержимым:

```

package validation

import (
    "encoding/json"
    "fmt"
    "net/http"

```

```

    )

    // Process is a handler that validates a post payload
    func (c *Controller) Process(w http.ResponseWriter, r
    *http.Request) {
        if r.Method != http.MethodPost {
            w.WriteHeader(http.StatusMethodNotAllowed)
            return
        }

        decoder := json.NewDecoder(r.Body)
        defer r.Body.Close()
        var p Payload

        if err := decoder.Decode(&p); err != nil {
            fmt.Println(err)
            w.WriteHeader(http.StatusBadRequest)
            return
        }

        if err := c.ValidatePayload(&p); err != nil {
            switch err.(type) {
            case Verror:
                w.WriteHeader(http.StatusBadRequest)
                // pass the Verror along
                w.Write([]byte(err.Error()))
                return
            default:
                w.WriteHeader(http.StatusInternalServerError)
                return
            }
        }
    }
}

```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "fmt"
    "net/http"

    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/"

```

```

        chapter8/validation"
    )

    func main() {
        c := validation.New()
        http.HandleFunc("/", c.Process)
        fmt.Println("Listening on port :3333")
        err := http.ListenAndServe(":3333", nil)
        panic(err)
    }

```

- Выполните `go run main.go`.
- Вы также можете запустить следующее:

```

$ go build
$ ./example

```

Вы должны увидеть следующий вывод:

```

$ go run main.go
Listening on port :3333

```

- В отдельном терминале выполните следующие команды:

```

$ curl "http://localhost:3333/" -X POST -d '{}'
$ curl "http://localhost:3333/" -X POST -d '{"name":"test"}'
$ curl "http://localhost:3333/" -X POST -d '{"name":"test",
"age": 5}' -v

```

Вы должны увидеть следующий вывод:

```

$ curl "http://localhost:3333/" -X POST -d '{}'
name is required

```

```

$ curl "http://localhost:3333/" -X POST -d '{"name":"test"}'
age is required and must be a value greater than 0 and
less than 120

```

```

$ curl "http://localhost:3333/" -X POST -d '{"name":"test",
"age": 5}' -v

```

**<lots of output, should contain a 200 OK status code>**

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что

все тесты пройдены.

## Как это работает...

Мы обрабатываем проверку, передавая замыкание в нашу структуру контроллера. Для любого ввода, который может потребоваться проверить контроллеру, нам понадобится одно из этих замыканий. Преимущество этого подхода в том, что мы можем имитировать и заменять функции проверки во время выполнения, поэтому тестирование становится намного проще. Кроме того, мы не привязаны к одной сигнатуре функции и можем передавать в наши функции проверки такие вещи, как соединение с базой данных.

Еще одна вещь, которую демонстрирует этот рецепт, — возврат типизированной ошибки с именем `Verror`. Этот тип содержит сообщения об ошибках проверки, которые могут отображаться для пользователей. Одним из недостатков этого подхода является то, что он не обрабатывает сразу несколько сообщений проверки. Это было бы возможно, изменив тип `Verror`, чтобы разрешить большее состояние, например, включив карту, чтобы разместить ряд ошибок проверки, прежде чем он вернется из нашей функции `ValidatePayload`.

## Рендеринг и согласование контента

Веб-обработчики могут возвращать различные типы контента; например, они могут возвращать JSON, обычный текст, изображения и многое другое. Часто при общении с API можно указать и принять тип контента, чтобы уточнить, в каком формате вы будете передавать данные и какие данные вы хотите получить обратно.

В этом рецепте будет рассмотрено использование `unroll/render` и пользовательской функции для согласования типа контента и соответствующего ответа.

## Как это сделать...

Следующие шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter8/negotiate` и перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter8/negotiate
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-
Second-Edition/chapter8/negotiate
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter8/negotiate` или используйте это как упражнение для написания собственного кода!
- Создайте файл с именем `negotiate.go` со следующим содержимым:

```
package negotiate

import (
    "net/http"

    "github.com/unrolled/render"
)

// Negotiator wraps render and does
// some switching on ContentType
type Negotiator struct {
    ContentType string
    *render.Render
}

// GetNegotiator takes a request, and figures
// out the ContentType from the Content-Type header
func GetNegotiator(r *http.Request) *Negotiator {
    contentType := r.Header.Get("Content-Type")

    return &Negotiator{
        ContentType: contentType,
        Render: render.New(),
    }
}
```

- Создайте файл с именем `respond.go` со следующим содержимым:

```
package negotiate

import "io"
import "github.com/unrolled/render"
```

```

// Respond switches on Content Type to determine
// the response
func (n *Negotiator) Respond(w io.Writer, status int,
v
interface{}) {
    switch n.ContentType {
    case render.ContentJSON:
        n.Render.JSON(w, status, v)
    case render.ContentXML:
        n.Render.XML(w, status, v)
    default:
        n.Render.JSON(w, status, v)
    }
}

```

- Создайте файл с именем `handler.go` со следующим содержимым:

```

package negotiate

import (
    "encoding/xml"
    "net/http"
)

// Payload defines it's layout in xml and json
type Payload struct {
    XMLName xml.Name `xml:"payload" json:"- "`
    Status string `xml:"status" json:"status"`
}

// Handler gets a negotiator using the request,
// then renders a Payload
func Handler(w http.ResponseWriter, r *http.Request) {
    n := GetNegotiator(r)

    n.Respond(w, http.StatusOK, &Payload{Status:
        "Successful!"})
}

```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (

```

```

    "fmt"
    "net/http"

    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter8/negotiate"
)

func main() {
    http.HandleFunc("/", negotiate.Handler)
    fmt.Println("Listening on port :3333")
    err := http.ListenAndServe(":3333", nil)
    panic(err)
}

```

- Выполните `go run main.go`.
- Вы также можете запустить следующее:

```

$ go build
$ ./example

```

Вы должны увидеть следующий вывод:

```

$ go run main.go
Listening on port :3333

```

- В отдельном терминале выполните следующие команды:

```

$ curl "http://localhost:3333" -H "Content-Type: text/xml"
$ curl "http://localhost:3333" -H "Content-Type:
application/json"

```

Вы должны увидеть следующий вывод:

```

$ curl "http://localhost:3333" -H "Content-Type: text/xml"
<payload><status>Successful!</status></payload>

$ curl "http://localhost:3333" -H "Content-Type:
application/json"
{"status":"Successful!"}

```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Пакет [github.com/unrolled/render](https://github.com/unrolled/render) делает тяжелую работу по этому рецепту. Существует огромное количество других параметров, которые вы можете ввести, если вам нужно работать с шаблонами HTML и т. д. Этот рецепт можно использовать для автоматического согласования при работе через веб-обработчики, как показано здесь, путем передачи заголовков различных типов контента или непосредственного управления структурой.

Аналогичный шаблон можно применить для принятия заголовков, но имейте в виду, что эти заголовки часто включают несколько значений, и ваш код должен это учитывать.

## Внедрение и использование промежуточного ПО

Промежуточное программное обеспечение для обработчиков в Go — это область, которая широко исследована. Существует множество пакетов для работы с промежуточным ПО. Этот рецепт создаст промежуточное программное обеспечение с нуля и реализует функцию [ApplyMiddleware](#) для объединения нескольких промежуточных программ.

Он также исследует установку значений в объекте контекста запроса и их последующее извлечение с помощью ПО промежуточного слоя. Все это будет сделано с помощью очень простого обработчика, чтобы продемонстрировать, как отделить логику промежуточного программного обеспечения от ваших обработчиков.

## Как это сделать...

Следующие шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter8/middleware` и перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter8/middleware
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:



`module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter8/middleware`

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter8/middleware` или используйте это как упражнение для написания собственного кода!
- Создайте файл с именем `middleware.go` со следующим содержимым:

```
package middleware

import (
    "log"
    "net/http"
    "time"
)

// Middleware is what all middleware functions will
return type Middleware func(http.HandlerFunc)
http.HandlerFunc

// ApplyMiddleware will apply all middleware, the last
// arguments will be the
// outer wrap for context passing purposes
func ApplyMiddleware(h http.HandlerFunc, middleware
...Middleware) http.HandlerFunc {
    applied := h
    for _, m := range middleware {
        applied = m(applied)
    }
    return applied
}

// Logger logs requests, this will use an id passed in
via // SetID()
func Logger(l *log.Logger) Middleware {
    return func(next http.HandlerFunc)
http.HandlerFunc {
    return func(w http.ResponseWriter, r
*http.Request) {
        start := time.Now()
        l.Printf("started request to %s with id
%s", r.URL,
        GetID(r.Context()))
        next(w, r)
    }
}
```

```

        l.Printf("completed request to %s with id
%s in %s", r.URL, GetID(r.Context()),
time.Since(start))
    }
}

```

- Создайте файл с именем `context.go` со следующим содержимым:

```

package middleware

import (
    "context"
    "net/http"
    "strconv"
)

// ContextID is our type to retrieve our context
// objects
type ContextID int

// ID is the only ID we've defined
const ID ContextID = 0

// SetID updates context with the id then
// increments it
func SetID(start int64) Middleware {
    return func(next http.HandlerFunc)
http.HandlerFunc {
        return func(w http.ResponseWriter, r
*http.Request) {
            ctx := context.WithValue(r.Context(), ID,
            strconv.FormatInt(start, 10))
            start++
            r = r.WithContext(ctx)
            next(w, r)
        }
    }
}

// GetID grabs an ID from a context if set
// otherwise it returns an empty string
func GetID(ctx context.Context) string {
    if val, ok := ctx.Value(ID).(string); ok {
        return val
    }
}

```

```
    }
    return ""
}
```

- Создайте файл с именем `handler.go` со следующим содержимым:

```
package middleware

import (
    "net/http"
)

// Handler is very basic
func Handler(w http.ResponseWriter, r *http.Request) {
    w.WriteHeader(http.StatusOK)
    w.Write([]byte("success"))
}
```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```
package main

import (
    "fmt"
    "log"
    "net/http"
    "os"

    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter8/middleware"
)

func main() {
    // We apply from bottom up
    h := middleware.ApplyMiddleware(
        middleware.Handler,
        middleware.Logger(log.New(os.Stdout, "", 0)),
        middleware.SetID(100),
    )
    http.HandleFunc("/", h)
    fmt.Println("Listening on port :3333")
    err := http.ListenAndServe(":3333", nil)
    panic(err)
}
```

- Выполните `go run main.go`.
- Вы также можете запустить следующие команды:

```
$ go build
$ ./example
```

Вы должны увидеть следующий вывод:

```
$ go run main.go
Listening on port :3333
```

- В отдельном терминале несколько раз запустите следующую команду `curl`:

```
$ curl http://localhost:3333
```

Вы должны увидеть следующий вывод:

```
$ curl http://localhost:3333
success
```

```
$ curl http://localhost:3333
success
```

```
$ curl http://localhost:3333
success
```

- В исходном файле `main.go` вы должны увидеть следующее:

```
Listening on port :3333
started request to / with id 100
completed request to / with id 100 in 52.284µs
started request to / with id 101
completed request to / with id 101 in 40.273µs
started request to / with id 102
```

- The `go.mod` file may be updated and the `go.sum` file should now be present in the top-level recipe directory.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

ПО промежуточного слоя можно использовать для выполнения простых операций, таких как ведение журнала, сбор метрик и аналитика. Промежуточное программное обеспечение также можно использовать для

динамического заполнения переменных при каждом запросе. Это можно сделать, например, для сбора X-заголовка из запроса для установки идентификатора или генерации идентификатора, как мы сделали в этом рецепте. Другой стратегией идентификации может быть создание **универсального уникального идентификатора (UUID)** для каждого запроса — это позволяет нам легко сопоставлять сообщения журнала и отслеживать ваш запрос в разных приложениях, если в построении ответа задействовано несколько микрослужб.

При работе со значениями контекста важно учитывать порядок ваших промежуточных программ. Как правило, лучше не делать ПО промежуточного слоя зависимым друг от друга. Например, в этом рецепте, вероятно, было бы лучше сгенерировать UUID в самом промежуточном программном обеспечении ведения журнала. Тем не менее, этот рецепт должен служить руководством по размещению промежуточного программного обеспечения и его инициализации в [main.go](#).

## Создание обратного прокси-приложения

В этом рецепте мы разработаем приложение обратного прокси. Идея заключается в том, что при нажатии <http://localhost:3333> в браузере весь трафик будет перенаправлен на настраиваемый хост, а ответы будут перенаправлены в ваш браузер. Конечным результатом должен быть <https://www.golang.org>, отображаемый в браузере через наше прокси-приложение.

Это можно комбинировать с переадресацией портов и туннелями SSH для безопасного доступа к веб-сайтам через промежуточный сервер. Этот рецепт создаст обратный прокси-сервер с нуля, но эта функциональность также предоставляется пакетом [net/http/httputil](#). Используя этот пакет, входящий запрос может быть изменен с помощью функции `Director(*http.Request)`, а исходящий ответ может быть изменен с помощью `ModifyResponse func(*http.Response) error`. Кроме того, есть поддержка буферизации ответа.

### Как это сделать...

Следующие шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем [~/projects/go-programming-cookbook/chapter8/proxy](#) и перейдите в этот каталог.

- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter8/proxy
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-
Second-Edition/chapter8/proxy
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter8/proxy` или используйте это как упражнение для написания собственного кода!
- Создайте файл с именем `proxy.go` со следующим содержимым:

```
package proxy

import (
    "log"
    "net/http"
)

// Proxy holds our configured client
// and BaseURL to proxy to
type Proxy struct {
    Client *http.Client
    BaseURL string
}

// ServeHTTP means that proxy implements the Handler
interface {
    // It manipulates the request, forwards it to BaseURL,
    then
    // returns the response
    func (p *Proxy) ServeHTTP(w http.ResponseWriter, r
    *http.Request) {
        if err := p.ProcessRequest(r); err != nil {
            log.Printf("error occurred during process
request: %s",
                err.Error())
            w.WriteHeader(http.StatusBadRequest)
            return
        }

        resp, err := p.Client.Do(r)
```

```

        if err != nil {
            log.Printf("error occurred during client
operation:
                %s", err.Error())
            w.WriteHeader(http.StatusInternalServerError)
            return
        }
        defer resp.Body.Close()
        CopyResponse(w, resp)
    }

```

- Создайте файл с именем `process.go` со следующим содержанием:

```

package proxy

import (
    "bytes"
    "net/http"
    "net/url"
)

// ProcessRequest modifies the request in accordance
// with Proxy settings
func (p *Proxy) ProcessRequest(r *http.Request) error
{
    proxyURLRaw := p.BaseURL + r.URL.String()

    proxyURL, err := url.Parse(proxyURLRaw)
    if err != nil {
        return err
    }
    r.URL = proxyURL
    r.Host = proxyURL.Host
    r.RequestURI = ""
    return nil
}

// CopyResponse takes the client response and writes
everything
// to the ResponseWriter in the original handler
func CopyResponse(w http.ResponseWriter, resp
*http.Response) {
    var out bytes.Buffer
    out.ReadFrom(resp.Body)

    for key, values := range resp.Header {

```

```

        for _, value := range values {
            w.Header().Add(key, value)
        }

        w.WriteHeader(resp.StatusCode)
        w.Write(out.Bytes())
    }

```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "fmt"
    "net/http"

    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter8/proxy"
)

func main() {
    p := &proxy.Proxy{
        Client: http.DefaultClient,
        BaseURL: "https://www.golang.org",
    }
    http.Handle("/", p)
    fmt.Println("Listening on port :3333")
    err := http.ListenAndServe(":3333", nil)
    panic(err)
}

```

- Выполните `go run main.go`.
- Вы также можете запустить следующее:

```

$ go build
$ ./example

```

Вы должны увидеть следующий вывод:

```

$ go run main.go
Listening on port :3333

```

- Перейдите в браузере на `localhost:3333/`. Вы должны увидеть отрендеренный веб-сайт <https://golang.org/>!



- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Объекты запроса и ответа Go в значительной степени могут использоваться клиентами и обработчиками. Этот код принимает запрос, полученный структурой `Proxy`, которая удовлетворяет интерфейсу `Handler`. Файл `main.go` использует `Handle` вместо `HandleFunc`, который используется где-то еще. Как только запрос доступен, он модифицируется, чтобы добавить `Proxy.BaseURL` к запросу, который затем отправляет клиент. Наконец, ответ копируется обратно в интерфейс `ResponseWriter`. Сюда входят все заголовки, тело и статус.

Мы также можем добавить некоторые дополнительные функции, такие как базовая `auth` для запросов, управление токенами и многое другое, если это необходимо. Это может быть полезно для управления токенами, когда прокси-сервер управляет сеансами для JavaScript или другого клиентского приложения.

## Экспорт GRPC как JSON API

В рецепте «Понимание клиентов GRPC» из Главы 7 «Веб-клиенты и API» мы написали базовый сервер и клиент GRPC. Этот рецепт расширит эту идею, поместив общие функции RPC в пакет и обернув их как в сервер GRPC, так и в стандартный веб-обработчик. Это может быть полезно, когда ваш API хочет поддерживать оба типа клиентов, но вы не хотите копировать код для общих функций.

## Подготовка

Настройте среду в соответствии со следующими шагами:

- См. шаги, указанные в разделе «Технические требования» в начале этой главы
- Установите GRPC (<https://grpc.io/docs/quickstart/go/>) и выполните следующие команды:
  - `go get -u github.com/golang/protobuf/{proto,protoc-gen-go}`

- `go get -u google.golang.org/grpc`

## Как это сделать...

Следующие шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter8/grpcjson` и перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter8/grpcjson
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter8/grpcjson
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter8/grpcjson` или используйте это как упражнение для написания собственного кода!
- Создайте новый каталог с именем `keyvalue` и перейдите к нему.
- Создайте файл с именем `keyvalue.proto` со следующим содержимым:

```
syntax = "proto3";

package keyvalue;

service KeyValue{
    rpc Set(SetKeyValueRequest) returns
(KeyValueResponse){}
    rpc Get(GetKeyValueRequest) returns
(KeyValueResponse){}
}

message SetKeyValueRequest {
    string key = 1;
    string value = 2;
}

message GetKeyValueRequest{
    string key = 1;
}
```

```
message KeyValueResponse{
    string success = 1;
    string value = 2;
}
```

- Выполните следующую команду:

```
$ protoc --go_out=plugins=grpc:. keyvalue.proto
```

- Перейдите обратно на каталог выше.
- Создайте новый каталог с именем `internal`.
- Создайте файл с именем `internal/keyvalue.go` со следующим содержанием:

```
package internal

import (
    "golang.org/x/net/context"
    "sync"

    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter8/grpcjson/keyvalue"
    "google.golang.org/grpc"
    "google.golang.org/grpc/codes"
)

// KeyValue is a struct that holds a map
type KeyValue struct {
    mutex sync.RWMutex
    m map[string]string
}

// NewKeyValue initializes the KeyValue struct and its
map
func NewKeyValue() *KeyValue {
    return &KeyValue{
        m: make(map[string]string),
    }
}

// Set sets a value to a key, then returns the value
func (k *KeyValue) Set(ctx context.Context, r
    *keyvalue.SetKeyValueRequest)
(*keyvalue.KeyValueResponse,
    error) {
```

```

        k.mutex.Lock()
        k.m[r.GetKey()] = r.GetValue()
        k.mutex.Unlock()
        return &keyvalue.KeyValueResponse{Value:
r.GetValue()}, nil
    }

    // Get gets a value given a key, or say not found if
    // it doesn't exist
    func (k *KeyValue) Get(ctx context.Context, r
    *keyvalue.GetKeyValueRequest)
    (*keyvalue.KeyValueResponse,
    error) {
        k.mutex.RLock()
    defer k.mutex.RUnlock()
        val, ok := k.m[r.GetKey()]
        if !ok {
            return nil, grpc.Errorf(codes.NotFound, "key
not set")
        }
        return &keyvalue.KeyValueResponse{Value: val}, nil
    }

```

- Создайте новый каталог с именем `grpc`.
- Создайте файл с именем `grpc/main.go` со следующим содержимым:

```

package main

import (
    "fmt"
    "net"

    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter8/grpcjson/internal"
    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter8/grpcjson/keyvalue"
    "google.golang.org/grpc"
)

func main() {
    grpcServer := grpc.NewServer()
    keyvalue.RegisterKeyValueServer(grpcServer,
    internal.NewKeyValue())
    lis, err := net.Listen("tcp", ":4444")

```

```

        if err != nil {
            panic(err)
        }
        fmt.Println("Listening on port :4444")
        grpcServer.Serve(lis)
    }

```

- Создайте новый каталог с именем `http`.
- Создайте файл с именем `http/set.go` со следующим содержимым:

```

package main

import (
    "encoding/json"
    "net/http"

    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter8/grpcjson/internal"
    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter8/grpcjson/keyvalue"
    "github.com/apex/log"
)

// Controller holds an internal KeyValueObject
type Controller struct {
    *internal.KeyValue
}

// SetHandler wraps our GRPC Set
func (c *Controller) SetHandler(w http.ResponseWriter,
r
    *http.Request) {
    var kv keyvalue.SetKeyValueRequest

    decoder := json.NewDecoder(r.Body)
    if err := decoder.Decode(&kv); err != nil {
        log.Errorf("failed to decode: %s",
err.Error())
        w.WriteHeader(http.StatusBadRequest)
        return
    }

    gresp, err := c.Set(r.Context(), &kv)
    if err != nil {

```

```

        log.Errorf("failed to set: %s", err.Error())
        w.WriteHeader(http.StatusInternalServerError)
        return
    }

    resp, err := json.Marshal(gresp)
    if err != nil {
        log.Errorf("failed to marshal: %s",
err.Error())
        w.WriteHeader(http.StatusInternalServerError)
        return
    }
    w.WriteHeader(http.StatusOK)
    w.Write(resp)
}

```

- Создайте файл с именем `http/get.go` со следующим содержимым:

```

package main

import (
    "encoding/json"
    "net/http"

    "google.golang.org/grpc"
    "google.golang.org/grpc/codes"

    "github.com/PacktPublishing/
Go-Programming-Cookbook-Second-Edition/
chapter8/grpcjson/keyvalue"
    "github.com/apex/log"
)

// GetHandler wraps our RPC Get call
func (c *Controller) GetHandler(w http.ResponseWriter,
r
    *http.Request) {
    key := r.URL.Query().Get("key")
    kv := keyvalue.GetKeyValueRequest{Key: key}

    gresp, err := c.Get(r.Context(), &kv)
    if err != nil {
        if grpc.Code(err) == codes.NotFound {
            w.WriteHeader(http.StatusNotFound)
            return
        }
    }
}

```

```

        log.Errorf("failed to get: %s", err.Error())
        w.WriteHeader(http.StatusInternalServerError)
        return
    }

    w.WriteHeader(http.StatusOK)
    resp, err := json.Marshal(gresp)
    if err != nil {
        log.Errorf("failed to marshal: %s",
err.Error())
        w.WriteHeader(http.StatusInternalServerError)
        return
    }
    w.Write(resp)
}

```

- Создайте файл с именем `http/main.go` со следующим содержимым:

```

package main

import (
    "fmt"
    "net/http"

    "github.com/PacktPublishing/
Go-Programming-Cookbook-Second-Edition/
chapter8/grpcjson/internal"
)

func main() {
    c := Controller{KeyValue: internal.NewKeyValue()}
    http.HandleFunc("/set", c.SetHandler)
    http.HandleFunc("/get", c.GetHandler)

    fmt.Println("Listening on port :3333")
    err := http.ListenAndServe(":3333", nil)
    panic(err)
}

```

- Запустите команду `go run ./http`. Вы должны увидеть следующий вывод:

```

$ go run ./http
Listening on port :3333

```

- В отдельном терминале выполните следующие команды:

```
$ curl "http://localhost:3333/set" -d '{"key":"test",
"value":"123"}' -v
$ curl "http://localhost:3333/get?key=badtest" -v
$ curl "http://localhost:3333/get?key=test" -v
```

Вы должны увидеть следующий вывод:

```
$ curl "http://localhost:3333/set" -d '{"key":"test",
"value":"123"}' -v
{"value":"123"}
```

```
$ curl "http://localhost:3333/get?key=badtest" -v
<should return a 404>
```

```
$ curl "http://localhost:3333/get?key=test" -v
{"value":"123"}
```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Хотя в этом рецепте отсутствует клиент, вы можете воспроизвести шаги рецепта «Понимание клиентов GRPC» из [Главы 7](#), «Веб-клиенты и API», и вы должны увидеть результаты, идентичные тем, что мы видим с нашими завитками. Каталоги `http` и `grpc` используют один и тот же внутренний пакет. Мы должны быть осторожны в этом пакете, чтобы вернуть соответствующие коды ошибок GRPC и правильно сопоставить эти коды ошибок с нашим ответом HTTP. В этом случае мы используем `codes.NotFound`, которые мы сопоставляем с `http.StatusNotFound`. Если вам нужно обработать несколько ошибок, оператор `switch` может иметь больше смысла, чем оператор `if...else`.

Еще одна вещь, которую вы можете заметить, это то, что сигнатуры GRPC обычно очень непротиворечивы. Они принимают запрос и возвращают необязательный ответ и ошибку. Можно создать общий обработчик, `shim`, если ваши вызовы GRPC достаточно повторяются, а также кажется, что он хорошо подходит для генерации кода; вы можете в конечном итоге увидеть что-то подобное с таким пакетом, как `goadesign/goa`.



## 9. Тестирование Go-кода

Эта глава будет отличаться от предыдущих глав; в этой главе основное внимание будет уделено тестированию и методологиям тестирования. Go предоставляет отличную поддержку тестирования из коробки. Однако это может быть трудно понять разработчикам, пришедшим из более динамичных языков, где исправление обезьяны и насмешка относительно просты.

Go-тестирование поощряет определенную структуру вашего кода. В частности, тестирование и имитация интерфейсов очень просты и хорошо поддерживаются. Некоторые типы кода могут быть более сложными для тестирования. Например, может быть сложно тестировать код, использующий глобальные переменные уровня пакета, места, которые не были абстрагированы в интерфейсы, и структуры, содержащие неэкспортируемые переменные или методы. В этой главе мы поделимся некоторыми рецептами тестирования кода Go.

В этой главе мы рассмотрим следующие рецепты:

- Мокинг с использованием стандартной библиотеки
- Использование пакета **Mockgen** для имитации интерфейсов
- Использование табличных тестов для улучшения покрытия
- Использование сторонних инструментов тестирования
- Поведенческое тестирование с использованием Go

### Технические требования

Чтобы продолжить выполнение всех рецептов в этой главе, настройте свою среду в соответствии со следующими шагами:

- Загрузите и установите Go 1.12.6 или более позднюю версию в своей операционной системе по адресу <https://golang.org/doc/install>.
- Откройте терминал или консольное приложение, создайте и перейдите в каталог проекта, например `~/projects/go-programming-cookbook`. Весь наш код будет запускаться и изменяться из этого каталога.

- Скопируйте последний код в `~/projects/go-programming-cookbook-original` и, при желании, работайте из этого каталога, вместо того, чтобы вводить примеры вручную:

```
$ git clone git@github.com:PacktPublishing/Go-Programming-Cookbook-Second-Edition.git go-programming-cookbook-original
```

## Мокинг с использованием стандартной библиотеки

В Go мокинг обычно означает реализацию интерфейса с тестовой версией, которая позволяет вам контролировать поведение во время выполнения из тестов. Это может также относиться к фиктивным функциям и методам, для которых мы рассмотрим еще один прием в этом рецепте. Этот трюк использует функции `Patch` и `Restore`, определенные на <https://play.golang.org/p/oLF1XnRX3C>.

В общем, код лучше составлять таким образом, чтобы можно было часто использовать интерфейсы и чтобы код был небольшим, пригодным для тестирования фрагментами. Код, который содержит множество условий ветвления или глубоко вложенную логику, может быть сложно тестировать, а тесты, как правило, становятся более хрупкими в конце. Это связано с тем, что разработчику нужно будет отслеживать больше фиктивных объектов, исправлений, возвращаемых значений и состояний в своих тестах.

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter9/mocking` и перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter9/mocking
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter9/mockin
```

- Создайте файл с именем `mock.go` со следующим содержимым:

```
package mocking

// DoStuffer is a simple interface
type DoStuffer interface {
    DoStuff(input string) error
}
```

- Создайте файл с именем `patch.go` со следующим содержимым:

```
package mocking

import "reflect"

// Restorer holds a function that can be used
// to restore some previous state.
type Restorer func()

// Restore restores some previous state.
func (r Restorer) Restore() {
    r()
}

// Patch sets the value pointed to by the given
destination to
// the given value, and returns a function to
restore it to its
// original value. The value must be assignable to
the element
//type of the destination.
func Patch(dest, value interface{}) Restorer {
    destv := reflect.ValueOf(dest).Elem()
    oldv := reflect.New(destv.Type()).Elem()
    oldv.Set(destv)
    valuev := reflect.ValueOf(value)
    if !valuev.IsValid() {
        // This isn't quite right when the
```

```

destination type is
    // not nilable, but it's better than the
complex
    // alternative.
    valuev = reflect.Zero(destv.Type())
}
destv.Set(valuev)
return func() {
    destv.Set(oldv)
}
}

```

- Создайте файл с именем `exec.go` со следующим содержимым:

```

package mocking
import "errors"
var ThrowError = func() error {
    return errors.New("always fails")
}

func DoSomeStuff(d DoStuffer) error {

    if err := d.DoStuff("test"); err != nil {
        return err
    }

    if err := ThrowError(); err != nil {
        return err
    }

    return nil
}

```

- Создайте файл с именем `mock_test.go` со следующим содержимым:

```

package mocking
type MockDoStuffer struct {
    // closure to assist with mocking
    MockDoStuff func(input string) error
}
func (m *MockDoStuffer) DoStuff(input string)
error {
    if m.MockDoStuff != nil {

```

```

        return m.MockDoStuff(input)
    }
    // if we don't mock, return a common case
    return nil
}

```

- Создайте файл с именем `exec_test.go` со следующим содержимым:

```

package mocking
import (
    "errors"
    "testing"
)

func TestDoSomeStuff(t *testing.T) {
    tests := []struct {
        name      string
        DoStuff    error
        ThrowError error
        wantErr    bool
    }{
        {"base-case", nil, nil, false},
        {"DoStuff error", errors.New("failed"),
nil, true},
        {"ThrowError error", nil,
errors.New("failed"), true},
    }
    for _, tt := range tests {
        t.Run(tt.name, func(t *testing.T) {
            // An example of mocking an interface
            // with our mock struct
            d := MockDoStuffer{}
            d.MockDoStuff = func(string) error {
                return tt.DoStuff }

            // mocking a function that is declared
as a variable
            // will not work for func A(),
            // must be var A = func()
            defer Patch(&ThrowError, func() error {
return
                tt.ThrowError }).Restore()

```

```

        if err := DoSomeStuff(&d); (err != nil)
    != tt.wantErr
    {
        t.Errorf("DoSomeStuff() error = %v,
        wantErr %v", err, tt.wantErr)
    }
    })
}

```

- Заполните тесты для оставшихся функций, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены:

```

$go test
PASS
ok github.com/PacktPublishing/Go-Programming-Cookbook-
Second-
Edition/chapter9/mocking 0.006s

```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.

## Как это работает...

Этот рецепт демонстрирует, как имитировать интерфейсы, а также функции, объявленные как переменные. Есть также определенные библиотеки, которые могут имитировать это исправление/восстановление непосредственно в объявленных функциях, но они обходят большую часть безопасности типов Go для достижения этой цели. Если вам нужно пропатчить функции из внешнего пакета, вы можете использовать следующий прием:

```

// Whatever package you wanna patch
import "github.com/package"

// This is patchable using the method described in this
recipe
var packageDoSomething = package.DoSomething

```

В этом рецепте мы начнем с настройки нашего теста и использования табличных тестов. Об этой методике написано много литературы,

например <https://github.com/golang/go/wiki/TableDrivenTests>, и я рекомендую изучить ее подробнее. Как только наши тесты настроены, мы выбираем выходные данные для наших фиктивных функций. Чтобы имитировать наш интерфейс, наши имитируемые объекты определяют замыкания, которые можно переписать во время выполнения. Техника patch/restore применяется для изменения нашей глобальной функции и ее восстановления после каждого цикла. Это благодаря `t.Run`, который устанавливает новую функцию для каждого цикла теста.

## Использование пакета Mockgen для имитации интерфейсов

В предыдущем примере использовались наши настраиваемые фиктивные объекты. Когда вы работаете с большим количеством интерфейсов, их написание может стать громоздким и подверженным ошибкам. Это место, где генерация кода имеет большой смысл. К счастью, есть пакет под названием [github.com/golang/mock/gomock](https://github.com/golang/mock/gomock), который обеспечивает создание фиктивных объектов и дает нам очень полезную библиотеку для использования в сочетании с тестированием интерфейса.

В этом рецепте будут рассмотрены некоторые функциональные возможности `gomock` и рассмотрены компромиссы в отношении того, где, когда и как работать с фиктивными объектами и генерировать их.

### Подготовка

Настройте среду в соответствии с этими шагами:

- См. раздел «Технические требования» в начале этой главы.
- Запустите команду `go get github.com/golang/mock/mockgen`.

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter9/mockgen` и перейдите в этот каталог.

- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter9/mockgen
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter9/mockgen
```

- Создайте файл с именем `interface.go` со следующим содержимым:

```
package mockgen

// GetSetter implements get a set of a
// key value pair
type GetSetter interface {
    Set(key, val string) error
    Get(key string) (string, error)
}
```

- Создайте каталог с именем `internal`.
- Запустите команду `mockgen -destination internal/mocks.go -package internal github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter9/mockgen GetSetter`. Это создаст файл с именем `internal/mocks.go`.
- Создайте файл с именем `exec.go` со следующим содержимым:

```
package mockgen

// Controller is a struct demonstrating
// one way to initialize interfaces
type Controller struct {
    GetSetter
}

// GetThenSet checks if a value is set. If not
// it sets it.
func (c *Controller) GetThenSet(key, value string)
error {
    val, err := c.Get(key)
```



```

        if err != nil {
            return err
        }

        if val != value {
            return c.Set(key, value)
        }
        return nil
    }

```

- Создайте файл с именем `interface_test.go` со следующим содержимым:

```

package mockgen

import (
    "errors"
    "testing"

    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter9/mockgen/internal"
    "github.com/golang/mock/gomock"
)

func TestExample(t *testing.T) {
    ctrl := gomock.NewController(t)
    defer ctrl.Finish()

    mockGetSetter :=
internal.NewMockGetSetter(ctrl)

    var k string
    mockGetSetter.EXPECT().Get("we can put
anything
here!").Do(func(key string) {
        k = key
    }).Return("", nil)

    customError := errors.New("failed this time")

    mockGetSetter.EXPECT().Get(gomock.Any()).Return("",

```

```

        customError)

        if _, err := mockGetSetter.Get("we can put
anything here!"); err != nil {
            t.Errorf("got %#v; want %#v", err, nil)
        }
        if k != "we can put anything here!" {
            t.Errorf("bad key")
        }

        if _, err := mockGetSetter.Get("key"); err ==
nil {
            t.Errorf("got %#v; want %#v", err,
customError)
        }
    }
}

```

- Создайте файл с именем `exec_test.go` со следующим содержимым:

```

package mockgen

import (
    "errors"
    "testing"

    "github.com/PacktPublishing/
Go-Programming-Cookbook-Second-Edition/
chapter9/mockgen/internal"
    "github.com/golang/mock/gomock"
)

func TestController_Set(t *testing.T) {
    tests := []struct {
        name string
        getReturnVal string
        getReturnErr error
        setReturnErr error
        wantErr bool
    }{
        {"get error", "value",
errors.New("failed"), nil,

```

```

        true},
        {"value match", "value", nil, nil, false},
        {"no errors", "not set", nil, nil, false},
        {"set error", "not set", nil,
errors.New("failed"),
        true},
    }
    for _, tt := range tests {
        t.Run(tt.name, func(t *testing.T) {
            ctrl := gomock.NewController(t)
            defer ctrl.Finish()

            mockGetSetter :=
internal.NewMockGetSetter(ctrl)

mockGetSetter.EXPECT().Get("key").AnyTimes()
                .Return(tt.getReturnVal,
tt.getReturnErr)
            mockGetSetter.EXPECT().Set("key",

gomock.Any()).AnyTimes().Return(tt.setReturnErr)

            c := &Controller{
                GetSetter: mockGetSetter,
            }
            if err := c.GetThenSet("key",
"value"); (err !=
nil) != tt.wantErr {
                t.Errorf("Controller.Set() error =
%v, wantErr
                %v", err, tt.wantErr)
            }
        })
    }
}

```

- Заполните тесты для оставшихся функций, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.
- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.

## Как это работает...

Сгенерированные фиктивные объекты позволяют тестам указать, какие аргументы ожидаются, сколько раз функция будет вызываться и что возвращать. Они также позволяют нам устанавливать дополнительные артефакты. Например, мы могли бы писать в канал напрямую, если исходная функция имела аналогичный рабочий процесс. Файл `interface_test.go` демонстрирует несколько примеров использования фиктивных объектов при вызове их в строке. Как правило, тесты будут больше похожи на `exec_test.go`, где нам нужно будет перехватывать вызовы функций интерфейса, выполняемые нашим реальным кодом, и изменять их поведение во время тестирования.

Файл `exec_test.go` также демонстрирует, как вы можете использовать фиктивные объекты в тестовой среде, управляемой таблицами. Функция `Any()` означает, что фиктивная функция может быть вызвана ноль или более раз, что отлично подходит для случаев, когда код завершается раньше.

Последний трюк, продемонстрированный в этом рецепте, заключается в том, чтобы вставлять фиктивные объекты в `internal` пакет. Это полезно, когда вам нужно имитировать функции, объявленные в пакетах за пределами вашего собственного. Это позволяет определить эти методы в файле `non_test.go`, но они не будут видны пользователям ваших библиотек, поскольку они не могут импортироваться из внутреннего пакета. Как правило, проще просто вставить фиктивные объекты в файлы `_test.go`, используя то же имя пакета, что и тесты, которые вы сейчас пишете.

## Использование табличных тестов для улучшения покрытия

Этот рецепт продемонстрирует процесс написания табличного теста, сбора тестового покрытия и его улучшения. Он также будет использовать пакет [github.com/cweill/gotests](https://github.com/cweill/gotests) для создания тестов. Если вы загружали тестовый код для других глав, он должен показаться вам очень знакомым. Используя комбинацию этого рецепта

и двух предыдущих, вы должны быть в состоянии достичь 100% покрытия тестами во всех случаях с некоторой работой.

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter9/coverage` и перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter9/coverage
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter9/coverage
```

- Создайте файл с именем `coverage.go` со следующим содержимым:

```
package main

import "errors"

// Coverage is a simple function with some
branching conditions
func Coverage(condition bool) error {
    if condition {
        return errors.New("condition was set")
    }
    return nil
}
```

- Запустите команду `gotests -all -w`.
- Это создаст файл с именем `cover_test.go` со следующим содержимым:

```
package main

import "testing"
```

```

func TestCoverage(t *testing.T) {
    type args struct {
        condition bool
    }
    tests := []struct {
        name string
        args args
        wantErr bool
    }{
        // TODO: Add test cases.
    }
    for _, tt := range tests {
        t.Run(tt.name, func(t *testing.T) {
            if err := Coverage(tt.args.condition);
(err != nil)
                != tt.wantErr {
                    t.Errorf("Coverage() error = %v,
wantErr %v",
                        err, tt.wantErr)
                }
            })
        }
    }
}

```

- Заполните раздел `TODO` следующим образом:

```

{"no condition", args{true}, true},

```

- Запустите команду `go test -cover`, и вы увидите следующий вывод:

```

$ go test -cover
PASS
coverage: 66.7% of statements
ok github.com/PacktPublishing/Go-Programming-Cookbook-
Second-
Edition/chapter9/coverage 0.007s

```

- Add the following item to the `TODO` section:

```

{"condition", args{false}, false},

```

- Run the `go test -cover` command, and you will see the following output:

```
$ go test -cover
PASS
coverage: 100.0% of statements
ok github.com/PacktPublishing/Go-Programming-Cookbook-Second-
Edition/chapter9/coverage 0.007s
```

- Выполните следующие команды:

```
$ go test -coverprofile=cover.out
$ go tool cover -html=cover.out -o coverage.html
```

- Open the `coverage.html` file in a browser to see a graphical coverage report.
- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.

## Как это работает...

Команда `go test -cover` входит в базовую установку Go. Его можно использовать для сбора отчета о покрытии вашего приложения Go. Кроме того, он имеет возможность выводить метрики покрытия и отчет о покрытии в формате HTML. Этот инструмент часто оборачивают другими инструментами, о которых пойдет речь в следующем рецепте. Эти стили тестов на основе таблиц описаны на <https://github.com/golang/go/wiki/TableDrivenTests> и являются отличным способом создания чистых тестов, которые могут обрабатывать многие случаи без написания большого количества дополнительного кода.

Этот рецепт начинается с автоматической генерации тестового кода, а затем заполнения тестовых случаев по мере необходимости, чтобы обеспечить большее покрытие. Единственный раз, когда это особенно сложно, — это когда у вас есть непеременные функции или методы, которые вызываются. Например, может быть сложно заставить `gob.Encode()` возвращать ошибку, чтобы увеличить тестовое покрытие. Также может показаться странным использовать метод, описанный в Mocking, используя рецепт стандартной библиотеки этой главы, и использовать `var gobEncode = gob.Encode`, чтобы разрешить исправление. По этой причине может быть трудно отстаивать 100-процентное покрытие тестами и вместо этого приводить доводы в пользу того, чтобы сосредоточиться на всестороннем тестировании

внешнего интерфейса, то есть на тестировании многих вариантов ввода и вывода, а в некоторых случаях, как мы увидим в рецепте «Поведенческое тестирование с использованием Go» из этой главы, фаззинг может оказаться полезным.

## Использование сторонних инструментов тестирования

Существует ряд полезных инструментов для тестирования Go: инструменты, упрощающие получение представления о покрытии кода на уровне каждой функции, инструменты для реализации утверждений для сокращения количества строк кода для тестирования и средства выполнения тестов. Этот рецепт охватывает пакеты [github.com/axw/gocov](https://github.com/axw/gocov) и [github.com/smartystreets/goconvey](https://github.com/smartystreets/goconvey), чтобы продемонстрировать некоторые из этих функций. Существует ряд других известных тестовых фреймворков в зависимости от ваших потребностей. Пакет [github.com/smartystreets/goconvey](https://github.com/smartystreets/goconvey) поддерживает оба утверждения и является средством запуска тестов. До Go 1.7 это был самый чистый способ маркировать подтесты.

### Подготовка

Настройте среду в соответствии с этими шагами:

- См. раздел «*Технические требования*» в начале этой главы.
- Запустите команду `go get github.com/axw/gocov/gocov`.
- Запустите команду `go get github.com/smartystreets/goconvey`.

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter9/tools` и перейдите в этот каталог.
- Выполните следующую команду:



```
$ go mod init github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter9/tools
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter9/tools
```

- Создайте файл с именем `funcs.go` со следующим содержимым:

```
package tools

import (
    "fmt"
)

func example() error {
    fmt.Println("in example")
    return nil
}

var example2 = func() int {
    fmt.Println("in example2")
    return 10
}
```

- Создайте файл с именем `structs.go` со следующим содержимым:

```
package tools

import (
    "errors"
    "fmt"
)

type c struct {
    Branch bool
}

func (c *c) example3() error {
    fmt.Println("in example3")
    if c.Branch {
        fmt.Println("branching code!")
    }
}
```

```

        return errors.New("bad branch")
    }

    return nil
}

```

- Создайте файл с именем `funcs_test.go` со следующим содержимым:

```

package tools

import (
    "testing"

    . "github.com/smartystreets/goconvey/convey"
)

func Test_example(t *testing.T) {
    tests := []struct {
        name string
    }{
        {"base-case"},
    }
    for _, tt := range tests {
        Convey(tt.name, t, func() {
            res := example()
            So(res, ShouldBeNil)
        })
    }
}

func Test_example2(t *testing.T) {
    tests := []struct {
        name string
    }{
        {"base-case"},
    }
    for _, tt := range tests {
        Convey(tt.name, t, func() {
            res := example2()
            So(res, ShouldBeGreaterThanOrEqualTo,
1)
        })
    }
}

```

```
    }
}
```

- Создайте файл с именем `structs_test.go` со следующим содержимым:

```
package tools

import (
    "testing"

    . "github.com/smartystreets/goconvey/convey"
)

func Test_c_example3(t *testing.T) {
    type fields struct {
        Branch bool
    }
    tests := []struct {
        name string
        fields fields
        wantErr bool
    }{
        {"no branch", fields{false}, false},
        {"branch", fields{true}, true},
    }
    for _, tt := range tests {
        Convey(tt.name, t, func() {
            c := &c{
                Branch: tt.fields.Branch,
            }
            So((c.example3() != nil), ShouldEqual,
tt.wantErr)
        })
    }
}
```

- Запустите команду `gocov test | gocov report`, и вы увидите следующий вывод:

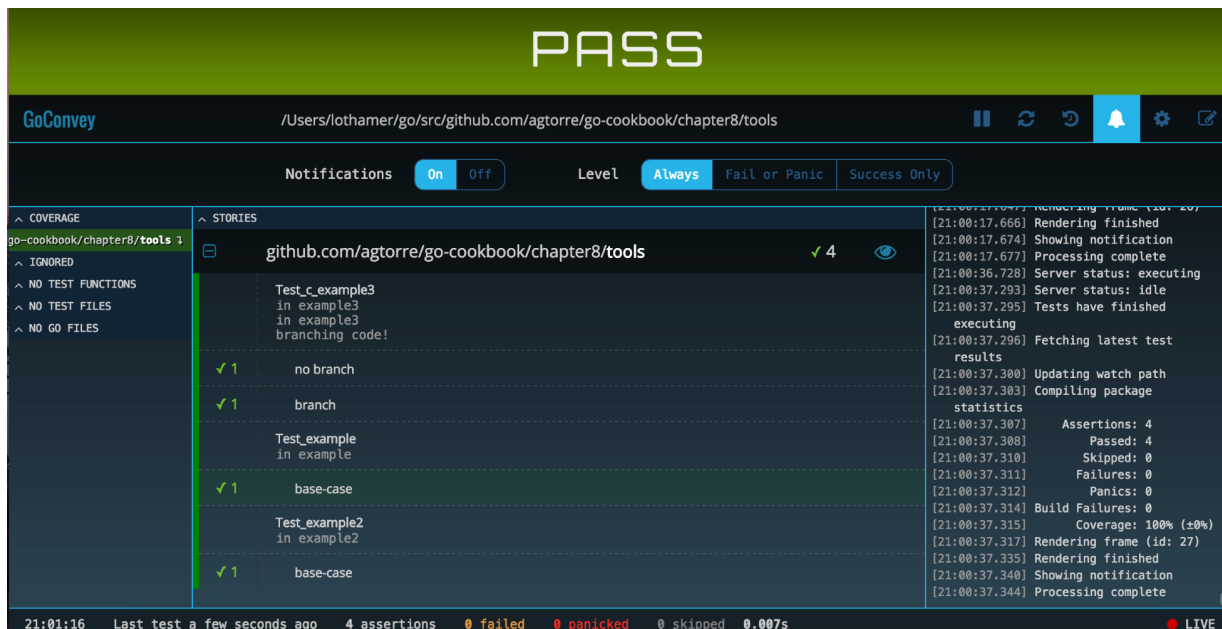
```
$ gocov test | gocov report
ok github.com/PacktPublishing/Go-Programming-Cookbook-Second-
Edition/chapter9/tools 0.006s
```

coverage: 100.0% of statements

```
github.com/PacktPublishing/Go-Programming-Cookbook-Second-
Edition/chapter9/tools/struct.go
c.example3 100.00% (5/5)
github.com/PacktPublishing/Go-Programming-Cookbook-Second-
Edition/chapter9/tools/funcs.go example
100.00% (2/2)
github.com/PacktPublishing/Go-Programming-Cookbook-Second-
Edition/chapter9/tools/funcs.go @12:16
100.00% (2/2)
github.com/PacktPublishing/Go-Programming-Cookbook-Second-
Edition/chapter9/tools -----
100.00% (9/9)
```

**Total Coverage: 100.00% (9/9)**

- Запустите команду `goconvey`, и она откроет браузер, который должен выглядеть так:



- Убедитесь, что все тесты пройдены.
- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.

## Как это работает...

Этот рецепт демонстрирует, как подключить команду `goconvey` к вашим тестам. Ключевое слово `Convey` в основном заменяет `t.Run` и добавляет дополнительные метки в веб-интерфейс `goconvey`, но ведет себя немного по-другому. Если у вас есть вложенные блоки `Convey`, они всегда повторно выполняются по порядку, как показано ниже:

```
Convey("Outer loop", t, func(){
    a := 1
    Convey("Inner loop", t, func() {
        a = 2
    })
    Convey("Inner loop2", t, func(){
        fmt.Println(a)
    })
})
```

Предыдущий код с использованием команды `goconvey` напечатает `1`. Если бы вместо этого мы использовали встроенную команду `t.Run`, она напечатала бы `2`. Другими словами, тесты Go `t.Run` выполняются последовательно и никогда не повторяются. Такое поведение может быть полезно для размещения кода установки во внешних блоках `Convey`, но важно помнить об этом различии, если вам приходится работать с обоими.

При использовании утверждений `Convey` есть отметки об успехах в веб-интерфейсе и в дополнительной статистике. Он также может уменьшить размер проверок до одной строки и даже создать собственные утверждения.

Если вы оставите веб-интерфейс `goconvey` включенным и включите уведомления, при сохранении кода тесты будут запускаться автоматически, и вы будете получать уведомления о любом увеличении или уменьшении охвата, а также о сбое сборки.

Все три инструмента утверждения, средство запуска тестов и веб-интерфейс можно использовать независимо или вместе.

Инструмент `gocov` может быть полезен при работе над более широким тестовым покрытием. Он может быстро определить функции, которым не хватает охвата, и поможет вам глубже погрузиться в отчет о

покрытии. Кроме того, `gocov` можно использовать для создания альтернативного HTML-отчета, который поставляется с кодом Go с помощью пакета [github.com/matm/gocov-html](https://github.com/matm/gocov-html).

## Поведенческое тестирование с использованием Go

Поведенческое тестирование или интеграционное тестирование — хороший метод реализации сквозного тестирования черного ящика. Одной из популярных платформ для этого типа тестирования является Cucumber (<https://cucumber.io/>), которая использует язык Gherkin для описания шагов теста на английском языке, а затем реализует эти шаги в коде. В Go также есть библиотека Cucumber ([github.com/DAI-DOG/godog](https://github.com/DAI-DOG/godog)). Этот рецепт будет использовать пакет `godog` для написания поведенческих тестов.

### Подготовка

Настройте среду в соответствии с этими шагами:

- См. раздел «*Технические требования*» в начале этой главы.
- Запустите команду `go get github.com/DAI-DOG/godog/cmd/godog`.

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter9/bdd` и перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter9/bdd
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter9/bdd
```

- Создайте файл с именем `handler.go` со следующим содержимым:

```
package bdd

import (
    "encoding/json"
    "fmt"
    "net/http"
)

// HandlerRequest will be json decoded
// into by Handler
type HandlerRequest struct {
    Name string `json:"name"`
}

// Handler takes a request and renders a response
func Handler(w http.ResponseWriter, r
*http.Request) {
    w.Header().Set("Content-Type", "text/plain;
charset=utf-8")
    if r.Method != http.MethodPost {
        w.WriteHeader(http.StatusMethodNotAllowed)
        return
    }

    dec := json.NewDecoder(r.Body)
    var req HandlerRequest
    if err := dec.Decode(&req); err != nil {
        w.WriteHeader(http.StatusBadRequest)
        return
    }

    w.WriteHeader(http.StatusOK)
    w.Write([]byte(fmt.Sprintf("BDD testing %s",
req.Name)))
}
```

- Создайте новый каталог с именем `functions` и файл с именем `features/handler.go` со следующим содержимым:

```
Feature: Bad Method
Scenario: Good request
```

Given we create a HandlerRequest payload with:

```
| reader |
| coder  |
| other  |
```

And we POST the HandlerRequest to /hello

Then the response code should be 200

And the response body should be:

```
| BDD testing reader |
| BDD testing coder  |
| BDD testing other  |
```

- Запустите команду `godog`, и вы увидите следующий вывод:

```
$ godog
```

```
.
1 scenarios (1 undefined)
4 steps (4 undefined)
89.062µs
.
```

- Это должно дать вам основу для реализации тестов, которые мы написали в нашем файле функций; скопируйте их в `handler_test.go` и выполните первые два шага:

```
package bdd

import (
    "bytes"
    "encoding/json"
    "fmt"
    "net/http/httptest"

    "github.com/DATA-DOG/godog"
    "github.com/DATA-DOG/godog/gherkin"
)

var payloads []HandlerRequest
var resps []*httptest.ResponseRecorder

func weCreateAHandlerRequestPayloadWith(arg1
*gherkin.DataTable) error {
    for _, row := range arg1.Rows {
        h := HandlerRequest{
            Name: row.Cells[0].Value,
```



```

    }
    payloads = append(payloads, h)
  }
  return nil
}

func wePOSTTheHandlerRequestToHello() error {
  for _, p := range payloads {
    v, err := json.Marshal(p)
    if err != nil {
      return err
    }
    w := httptest.NewRecorder()
    r := httptest.NewRequest("POST", "/hello",
      bytes.NewBuffer(v))

    Handler(w, r)
    resps = append(resps, w)
  }
  return nil
}

```

- Запустите команду **godog**, и вы увидите следующий вывод:

```

$ godog
.
1 scenarios (1 pending)
4 steps (2 passed, 1 pending, 1 skipped)
.

```

- Заполните оставшиеся два шага:

```

func theResponseCodeShouldBe(arg1 int) error {
  for _, r := range resps {
    if got, want := r.Code, arg1; got != want
{
      return fmt.Errorf("got: %d; want %d",
got, want)
    }
  }
  return nil
}

func theResponseBodyShouldBe(arg1

```

```

*gherkin.DataTable) error {
    for c, row := range arg1.Rows {
        b := bytes.Buffer{}
        b.ReadFrom(resps[c].Body)
        if got, want := b.String(),
row.Cells[0].Value;
        got != want
        {
            return fmt.Errorf("got: %s; want %s",
got, want)
        }
    }
    return nil
}

func FeatureContext(s *godog.Suite) {
    s.Step(`^we create a HandlerRequest payload
with:$`,
        weCreateAHandlerRequestPayloadWith)
    s.Step(`^we POST the HandlerRequest to
/hello$`,
        wePOSTTheHandlerRequestToHello)
    s.Step(`^the response code should be (d+)$`,
        theResponseCodeShouldBe)
    s.Step(`^the response body should be:$`,
        theResponseBodyShouldBe)
}

```

- Запустите команду **godog**, и вы увидите следующий вывод:

```

$ godog
.
1 scenarios (1 passed)
4 steps (4 passed)
552.605µs
.

```

## Как это работает...

Фреймворки Cucumber отлично подходят для парного программирования, сквозного тестирования и любых видов тестирования, которые лучше всего сопровождаются письменными инструкциями и понятны для нетехнических людей. Как только шаг

реализован, обычно его можно повторно использовать везде, где это необходимо. Если вы хотите протестировать интеграцию между службами, можно написать тесты для использования реальных HTTP-клиентов, если вы сначала убедитесь, что ваша среда настроена для приема HTTP-соединений.

В реализации **datadog поведенческо-ориентированной разработки (behavior-driven development — BDD)**, отсутствуют некоторые функции, которые вы могли бы ожидать, если бы вы использовали другие среды Cucumber, включая отсутствие примеров, передачу контекста между функциями и ряд других ключевых слов. Тем не менее, это хорошее начало, и, используя несколько трюков в этом рецепте, таких как глобальные переменные для отслеживания состояния (и гарантируя, что вы очищаете эти глобальные переменные между сценариями), можно создать довольно надежный набор тестов. Тестовый пакет **datadog** также использует сторонний тест-раннер, поэтому его невозможно собрать вместе с такими пакетами, как **gocov** или **go test -cover**.

# 10. Параллелизм и конкурентность

Рецепты в этой главе охватывают рабочие пулы, группы ожидания для асинхронных операций и использование пакета `context`. Параллелизм и конкурентность — одни из наиболее рекламируемых и продвигаемых функций языка Go. В этой главе будет предложен ряд полезных шаблонов, которые помогут вам начать работу и помогут понять эти функции.

Go предоставляет примитивы, делающие возможными параллельные приложения. Горутины позволяют любой функции стать асинхронной и параллельной. Каналы позволяют приложению устанавливать связь с горутинами. Одно из известных высказываний в Go: *«Не общайтесь, делаясь памятью; вместо этого делитесь памятью, общаясь»*, и взято из <https://blog.golang.org/share-memory-by-communicating>.

В этой главе мы рассмотрим следующие рецепты:

- Использование каналов и оператора `select`
- Выполнение асинхронных операций с `sync.WaitGroup`
- Использование атомарных операций и мьютекса
- Использование пакета `context`
- Выполнение управления состоянием для каналов
- Использование шаблона проектирования пула рабочих процессов
- Использование воркеров для создания пайплайнов

## Технические требования

Чтобы продолжить выполнение всех рецептов в этой главе, настройте свою среду в соответствии со следующими шагами:

- Загрузите и установите Go 1.12.6 или более позднюю версию в своей операционной системе по адресу <https://golang.org/doc/install>.
- Откройте терминал или консольное приложение, создайте и перейдите в каталог проекта, например `~/projects/go-`

`programming-cookbook`. Весь наш код будет запускаться и изменяться из этого каталога.

- Скопируйте последний код в `~/projects/go-programming-cookbook-original` и, при желании, работайте из этого каталога, вместо того, чтобы вводить примеры вручную:

```
$ git clone git@github.com:PacktPublishing/Go-Programming-Cookbook-Second-Edition.git go-programming-cookbook-original
```

## Использование каналов и оператора `select`

Каналы Go в сочетании с горутинами являются первоклассными гражданами для асинхронного общения. Каналы становятся особенно мощными, когда мы используем операторы `select`. Эти операторы позволяют горутине разумно обрабатывать запросы из нескольких каналов одновременно.

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter10/channels` и перейдите в него.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter10/channels
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующий код:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter10/channels
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter10/channels` или используйте это как возможность написать свой собственный код!
- Создайте файл с именем `sender.go` со следующим содержимым:

```

package channels

import "time"

// Sender sends "tick" on ch until done is
// written to, then it sends "sender done."
// and exits
func Sender(ch chan string, done chan bool) {
    t := time.Tick(100 * time.Millisecond)
    for {
        select {
        case <-done:
            ch <- "sender done."
            return
        case <-t:
            ch <- "tick"
        }
    }
}

```

- Создайте файл с именем `printer.go` со следующим содержимым:

```

package channels

import (
    "context"
    "fmt"
    "time"
)

// Printer will print anything sent on the ch chan
// and will print tock every 200 milliseconds
// this will repeat forever until a context is
// Done, i.e. timed out or cancelled
func Printer(ctx context.Context, ch chan string)
{
    t := time.Tick(200 * time.Millisecond)
    for {
        select {
        case <-ctx.Done():
            fmt.Println("printer done.")
            return
        case res := <-ch:

```

```

        fmt.Println(res)
    case <-t:
        fmt.Println("tock")
    }
}
}

```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "context"
    "time"

    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter10/channels"
)

func main() {
    ch := make(chan string)
    done := make(chan bool)

    ctx := context.Background()
    ctx, cancel := context.WithCancel(ctx)
    defer cancel()

    go channels.Printer(ctx, ch)
    go channels.Sender(ch, done)

    time.Sleep(2 * time.Second)
    done <- true
    cancel()
    //sleep a bit extra so channels can clean up
    time.Sleep(3 * time.Second)
}

```

- Выполните `go run main.go`.
- Вы также можете запустить следующие команды:

```

$ go build
$ ./example

```

Теперь вы должны увидеть следующий вывод, но порядок печати может отличаться:

```
$ go run main.go
tick
tock
tick
tick
tock
tick
tick
tock
tick
.
.
.
sender done.
printer done.
```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Этот рецепт демонстрирует два способа запуска рабочего процесса, который либо читает, либо записывает в канал, и потенциально может делать и то, и другое. Рабочий процесс завершится, когда будет произведена запись в канал `done` или когда `context` будет отменен посредством вызова функции отмены или по таймауту. Рецепт «*Использование пакета context*» более подробно описывает пакет `context`.

Пакет `main` используется для соединения отдельных функций; благодаря этому можно настроить несколько пар, если каналы не используются совместно. В дополнение к этому, на одном и том же канале может прослушиваться несколько горутин, как мы рассмотрим в рецепте «*Использование шаблона проектирования пула рабочих процессов*».



Наконец, из-за асинхронной природы горутин может быть сложно установить условия очистки и завершения; например, распространенной ошибкой является выполнение следующего:

```
select{
    case <-time.Tick(200 * time.Millisecond):
        //this resets whenever any other 'lane' is chosen
}
```

Поставив `Tick` в операторе `select`, можно предотвратить возникновение этого случая. Также нет простого способа приоритизировать трафик в операторе `select`.

## Выполнение асинхронных операций с `sync.WaitGroup`

Иногда полезно выполнить ряд операций асинхронно, а затем дождаться их завершения, прежде чем двигаться дальше. Например, если операция требует извлечения информации из нескольких API и ее агрегирования, может быть полезно выполнять эти клиентские запросы асинхронно. В этом рецепте рассматривается использование `sync.WaitGroup` для параллельной организации независимых задач.

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter10/waitgroup` и перейдите в него.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter10/waitgroup
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter10/waitgroup
```

- Скопируйте тесты из [~/projects/go-programming-cookbook-original/chapter10/waitgroup](#) или используйте это как возможность написать свой собственный код!
- Создайте файл с именем `tasks.go` со следующим содержимым:

```
package waitgroup

import (
    "fmt"
    "log"
    "net/http"
    "strings"
    "time"
)

// GetURL gets a url, and logs the time it took
func GetURL(url string) (*http.Response, error) {
    start := time.Now()
    log.Printf("getting %s", url)
    resp, err := http.Get(url)
    log.Printf("completed getting %s in %s", url,
        time.Since(start))
    return resp, err
}

// CrawlError is our custom error type
// for aggregating errors
type CrawlError struct {
    Errors []string
}

// Add adds another error
func (c *CrawlError) Add(err error) {
    c.Errors = append(c.Errors, err.Error())
}

// Error implements the error interface
func (c *CrawlError) Error() string {
    return fmt.Sprintf("All Errors: %s",
strings.Join(c.Errors,
    ", "))
}
```

```
// Present can be used to determine if
// we should return this
func (c *CrawlError) Present() bool {
    return len(c.Errors) != 0
}
```

- Создайте файл с именем `process.go` со следующим содержимым:

```
package waitgroup

import (
    "log"
    "sync"
    "time"
)

// Crawl collects responses from a list of urls
// that are passed in. It waits for all requests
// to complete before returning.
func Crawl(sites []string) ([]int, error) {
    start := time.Now()
    log.Printf("starting crawling")
    wg := &sync.WaitGroup{}

    var resps []int
    cerr := &CrawlError{}
    for _, v := range sites {
        wg.Add(1)
        go func(v string) {
            defer wg.Done()
            resp, err := GetURL(v)
            if err != nil {
                cerr.Add(err)
                return
            }
            resps = append(resps, resp.StatusCode)
        }(v)
    }
    wg.Wait()
    // we encountered a crawl error
    if cerr.Present() {
        return resps, cerr
    }
}
```

```

    }
    log.Printf("completed crawling in %s",
time.Since(start))
    return resps, nil
}

```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "fmt"

    "github.com/PacktPublishing/
Go-Programming-Cookbook-Second-Edition/
chapter10/waitgroup"
)

func main() {
    sites := []string{
        "https://golang.org",
        "https://godoc.org",
        "https://www.google.com/search?q=golang",
    }

    resps, err := waitgroup.Crawl(sites)
    if err != nil {
        panic(err)
    }
    fmt.Println("Resps received:", resps)
}

```

- Выполните `go run main.go`.
- Вы также можете запустить следующие команды:

```

$ go build
$ ./example

```

Вы должны увидеть следующий вывод:

```

$ go run main.go
2017/04/05 19:45:07 starting crawling
2017/04/05 19:45:07 getting

```

```

https://www.google.com/search?
q=golang
2017/04/05 19:45:07 getting https://golang.org
2017/04/05 19:45:07 getting https://godoc.org
2017/04/05 19:45:07 completed getting https://golang.org
in
178.22407ms
2017/04/05 19:45:07 completed getting https://godoc.org
in
181.400873ms
2017/04/05 19:45:07 completed getting
https://www.google.com/search?q=golang in 238.019327ms
2017/04/05 19:45:07 completed crawling in 238.191791ms
Resps received: [200 200 200]

```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Этот рецепт показывает вам, как использовать группы ожидания в качестве механизма синхронизации при ожидании работы. По сути, `waitgroup.Wait()` будет ждать, пока его внутренний счетчик не достигнет `0`. Метод `waitgroup.Add(int)` увеличит счетчик на введенное значение, а `waitgroup.Done()` уменьшит счетчик на `1`. Потому что из этого необходимо асинхронно `Wait()`, в то время как различные горуты помечают группу ожидания как `Done()`.

В этом рецепте мы увеличиваем значение перед отправкой каждого HTTP-запроса, а затем вызываем метод `defer wg.Done()`, чтобы мы могли уменьшить значение всякий раз, когда горутина завершает работу. Затем мы ждем завершения всех горутин, прежде чем вернуть наши агрегированные результаты.

На практике лучше использовать каналы для передачи ошибок и ответов.

Выполняя такие асинхронные операции, вы должны учитывать безопасность потоков для таких вещей, как изменение общей карты.

Если вы помните об этом, `waitgroups` — полезная функция для ожидания любой асинхронной операции.

## Использование атомарных операций и мьютекса

В таком языке, как Go, где вы можете встроить асинхронные операции и параллелизм, становится важным учитывать такие вещи, как безопасность потоков. Например, опасно одновременно обращаться к карте из нескольких горутин. Go предоставляет ряд помощников в пакетах `sync` и `sync/atomic`, чтобы убедиться, что определенные события происходят только один раз, или что горутин могут сериализоваться при операции.

Этот рецепт продемонстрирует использование этих пакетов для безопасного изменения карты с помощью различных горутин и сохранения глобального порядкового значения, к которому могут безопасно обращаться многочисленные горутин. Он также продемонстрирует метод `Once.Do`, который можно использовать для обеспечения того, чтобы приложение Go делало что-либо только один раз, например чтение файла конфигурации или инициализацию переменной.

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter10/atomic` и перейдите в него.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter10/atomic
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter10/atomic
```

- Скопируйте тесты из [~/projects/go-programming-cookbook-original/chapter10/atomic](#) или используйте это как возможность написать свой собственный код!
- Создайте файл с именем `map.go` со следующим содержимым:

```
package atomic

import (
    "errors"
    "sync"
)

// SafeMap uses a mutex to allow
// getting and setting in a thread-safe way
type SafeMap struct {
    m map[string]string
    mu *sync.RWMutex
}

// NewSafeMap creates a SafeMap
func NewSafeMap() SafeMap {
    return SafeMap{m: make(map[string]string), mu:
        &sync.RWMutex{}}
}

// Set uses a write lock and sets the value given
// a key
func (t *SafeMap) Set(key, value string) {
    t.mu.Lock()
    defer t.mu.Unlock()

    t.m[key] = value
}

// Get uses a RW lock and gets the value if it
exists,
// otherwise an error is returned
func (t *SafeMap) Get(key string) (string, error)
{
    t.mu.RLock()
    defer t.mu.RUnlock()
```

```

        if v, ok := t.m[key]; ok {
            return v, nil
        }

        return "", errors.New("key not found")
    }

```

- Создайте файл с именем `ordinal.go` со следующим содержимым:

```

package atomic

import (
    "sync"
    "sync/atomic"
)

// Ordinal holds a global a value
// and can only be initialized once
type Ordinal struct {
    ordinal uint64
    once *sync.Once
}

// NewOrdinal returns ordinal with once
// setup
func NewOrdinal() *Ordinal {
    return &Ordinal{once: &sync.Once{}}
}

// Init sets the ordinal value
// can only be done once
func (o *Ordinal) Init(val uint64) {
    o.once.Do(func() {
        atomic.StoreUint64(&o.ordinal, val)
    })
}

// GetOrdinal will return the current
// ordinal
func (o *Ordinal) GetOrdinal() uint64 {
    return atomic.LoadUint64(&o.ordinal)
}

```



```
// Increment will increment the current
// ordinal
func (o *Ordinal) Increment() {
    atomic.AddUint64(&o.ordinal, 1)
}
```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```
package main

import (
    "fmt"
    "sync"

    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter10/atomic"
)

func main() {
    o := atomic.NewOrdinal()
    m := atomic.NewSafeMap()
    o.Init(1123)
    fmt.Println("initial ordinal is:",
o.GetOrdinal())
    wg := sync.WaitGroup{}
    for i := 0; i < 10; i++ {
        wg.Add(1)
        go func(i int) {
            defer wg.Done()
            m.Set(fmt.Sprintf(i), "success")
            o.Increment()
        }(i)
    }

    wg.Wait()
    for i := 0; i < 10; i++ {
        v, err := m.Get(fmt.Sprintf(i))
        if err != nil || v != "success" {
            panic(err)
        }
    }
}
```

```

        fmt.Println("final ordinal is:",
o.GetOrdinal())
        fmt.Println("all keys found and marked as:
'success'")
    }

```

- Выполните `go run main.go`.
- Вы также можете запустить следующие команды:

```

$ go build
$ ./example

```

Вы также можете запустить следующие команды:

```

$ go run main.go
initial ordinal is: 1123
final ordinal is: 1133
all keys found and marked as: 'success'

```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Для нашего рецепта карты мы использовали мьютекс `ReadWrite`. Идея этого мьютекса заключается в том, что любое количество ридеров может получить блокировку чтения, но только один райтер может получить блокировку записи. Кроме того, писатель не может не может получить блокировку, когда она есть у кого-то другого (ридера или райтера). Это полезно, потому что чтение выполняется очень быстро и не блокирует по сравнению со стандартным мьютексом. Всякий раз, когда мы хотим установить данные, мы используем объект `Lock()`, а всякий раз, когда мы хотим прочитать данные, мы используем `RLock()`. Крайне важно, чтобы вы использовали `Unlock()` или `RUnlock()` в конечном итоге, чтобы не заблокировать приложение. Отложенный объект `Unlock()` может быть полезен, но может быть медленнее, чем вызов `Unlock()` вручную.

Этот шаблон может быть недостаточно гибким, если вы хотите сгруппировать дополнительные действия с заблокированным значением. Например, в некоторых случаях вы можете захотеть заблокировать, выполнить некоторую дополнительную обработку, и только после того, как вы это сделаете, вы разблокируете. Это важно учитывать при разработке дизайна.

Пакет `sync/atomic` используется `Ordinal` для получения и установки значений. Существуют также операции атомарного сравнения, такие как `atomic.CompareAndSwapUint64()`, которые чрезвычайно полезны. Этот рецепт позволяет вызывать `Init` для объекта `Ordinal` только один раз; в противном случае его можно только увеличивать и делать это атомарно.

Мы зацикливаем и создаем 10 горутин (синхронизируя с `sync.Waitgroup`) и показываем, что порядковый номер правильно увеличился в 10 раз и что каждый ключ в нашей карте был установлен соответствующим образом.

## Использование пакета context

Несколько рецептов в этой книге используют пакет `context`. Этот рецепт исследует основы создания контекстов и управления ими. Хороший справочник для понимания контекста — <https://blog.golang.org/context>. С момента написания этого сообщения в блоге контекст переместился из `net/context` в пакет с именем `context`. Это по-прежнему иногда вызывает проблемы при взаимодействии со сторонними библиотеками, такими как GRPC.

Этот рецепт исследует настройку и получение значений для контекстов, отмены и таймаутов.

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter10/context` и перейдите к нему.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter10/context
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter10/context
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter10/context` или используйте это как упражнение для написания собственного кода!
- Создайте файл с именем `values.go` со следующим содержимым:

```
package context

import "context"

type key string

const (
    timeoutKey key = "TimeoutKey"
    deadlineKey key = "DeadlineKey"
)

// Setup sets some values
func Setup(ctx context.Context) context.Context {

    ctx = context.WithValue(ctx, timeoutKey,
        "timeout exceeded")
    ctx = context.WithValue(ctx, deadlineKey,
        "deadline exceeded")

    return ctx
}

// GetValue grabs a value given a key and
// returns a string representation of the
// value
func GetValue(ctx context.Context, k key) string {

    if val, ok := ctx.Value(k).(string); ok {
        return val
    }
}
```

```

    }
    return ""
}

```

- Создайте файл с именем `exec.go` со следующим содержимым:

```

package context

import (
    "context"
    "fmt"
    "math/rand"
    "time"
)

// Exec sets two random timers and prints
// a different context value for whichever
// fires first
func Exec() {
    // a base context
    ctx := context.Background()
    ctx = Setup(ctx)

    rand.Seed(time.Now().UnixNano())

    timeoutCtx, cancel := context.WithTimeout(ctx,
        (time.Duration(rand.Intn(2)) *
time.Millisecond))
    defer cancel()

    deadlineCtx, cancel :=
context.WithDeadline(ctx,
    time.Now().Add(time.Duration(rand.Intn(2))
*time.Millisecond))
    defer cancel()

    for {
        select {
            case <-timeoutCtx.Done():
                fmt.Println(GetValue(ctx, timeoutKey))
                return
            case <-deadlineCtx.Done():

```

```

        fmt.Println(GetValue(ctx,
deadlineKey))
    }
    return
}
}

```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import "github.com/PacktPublishing/
Go-Programming-Cookbook-Second-
Edition/
chapter10/context"

func main() {
    context.Exec()
}

```

- Выполните `go run main.go`.
- Вы также можете запустить следующие команды:

```

$ go build
$ ./example

```

Вы также можете запустить следующие команды:

```

$ go run main.go
timeout exceeded
OR
$ go run main.go
deadline exceeded

```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

При работе со значениями контекста полезно создать новый тип для представления ключа. В этом случае мы создали тип `key`, а затем объявили некоторые соответствующие `const` значения для представления всех наших возможных ключей.

В этом случае мы инициализируем все наши пары ключ/значение одновременно с помощью функции `Setup()`. При изменении контекстов функции обычно принимают аргумент `context` и возвращают значение `context`. Итак, подпись часто выглядит так:

```
func ModifyContext(ctx context.Context) context.Context
```

Иногда эти методы также возвращают ошибку или функцию `cancel()`, например, в случаях `context.WithCancel`, `context.WithTimeout` и `context.WithDeadline`. Все дочерние контексты наследуют атрибуты родителя.

В этом рецепте мы создали два дочерних контекста, один с дедлайном, а другой с таймаутом. Мы устанавливаем для них таймаут, чтобы они были случайными диапазонами, а затем прекращаем работу, когда любой из них получен. Наконец, мы извлекли значение с заданным ключом и напечатали его.

## Выполнение управления состоянием для каналов

Каналы в Go могут быть любого типа. Канал структур позволяет передавать множество состояний с помощью одного сообщения. Этот рецепт исследует использование каналов для передачи сложных структур запросов и возврата их результатов в сложных структурах ответов.

В следующем рецепте «*Использование шаблона проектирования пула рабочих операций*» значение этого становится еще более очевидным, поскольку вы можете создавать рабочие процессы общего назначения, способные выполнять различные задачи.

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter10/state` и перейдите в него.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter10/state
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter10/state
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter10/state` или используйте это как возможность написать свой собственный код!
- Создайте файл с именем `state.go` со следующим содержимым:

```
package state

type op string

const (
    // Add values
    Add op = "add"
    // Subtract values
    Subtract = "sub"
    // Multiply values
    Multiply = "mult"
    // Divide values
    Divide = "div"
)

// WorkRequest perform an op
// on two values
type WorkRequest struct {
    Operation op
    Value1 int64
    Value2 int64
}

// WorkResponse returns the result
```



```
// and any errors
type WorkResponse struct {
    Wr *WorkRequest
    Result int64
    Err error
}
```

- Создайте файл с именем `processor.go` со следующим содержимым:

```
package state

import "context"

// Processor routes work to Process
func Processor(ctx context.Context, in chan
*WorkRequest, out
chan *WorkResponse) {
    for {
        select {
            case <-ctx.Done():
                return
            case wr := <-in:
                out <- Process(wr)
        }
    }
}
```

- Создайте файл с именем `process.go` со следующим содержимым:

```
package state

import "errors"

// Process switches on operation type
// Then does work
func Process(wr *WorkRequest) *WorkResponse {
    resp := WorkResponse{Wr: wr}

    switch wr.Operation {
        case Add:
            resp.Result = wr.Value1 + wr.Value2
        case Subtract:
            resp.Result = wr.Value1 - wr.Value2
    }
}
```

```

        case Multiply:
            resp.Result = wr.Value1 * wr.Value2
        case Divide:
            if wr.Value2 == 0 {
                resp.Err = errors.New("divide by
0")
                break
            }
            resp.Result = wr.Value1 / wr.Value2
        default:
            resp.Err = errors.New("unsupported
operation")
    }
    return &resp
}

```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "context"
    "fmt"

    "github.com/PacktPublishing/
Go-Programming-Cookbook-Second-Edition/
chapter10/state"
)

func main() {
    in := make(chan *state.WorkRequest, 10)
    out := make(chan *state.WorkResponse, 10)
    ctx := context.Background()
    ctx, cancel := context.WithCancel(ctx)
    defer cancel()

    go state.Processor(ctx, in, out)

    req := state.WorkRequest{state.Add, 3, 4}
    in <- &req

    req2 := state.WorkRequest{state.Subtract, 5,

```

```

2}
    in <- &req2

    req3 := state.WorkRequest{state.Multiply, 9,
9}
    in <- &req3

    req4 := state.WorkRequest{state.Divide, 8, 2}
    in <- &req4

    req5 := state.WorkRequest{state.Divide, 8, 0}
    in <- &req5

    for i := 0; i < 5; i++ {
        resp := <-out
        fmt.Printf("Request: %v; Result: %v,
Error: %vn",
            resp.Wr, resp.Result, resp.Err)
    }
}

```

- Выполните `go run main.go`.
- Вы также можете запустить следующие команды:

```

$ go build
$ ./example

```

Вы также можете запустить следующие команды:

```

$ go run main.go
Request: &{add 3 4}; Result: 7, Error: <nil>
Request: &{sub 5 2}; Result: 3, Error: <nil>
Request: &{mult 9 9}; Result: 81, Error: <nil>
Request: &{div 8 2}; Result: 4, Error: <nil>
Request: &{div 8 0}; Result: 0, Error: divide by 0

```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Функция `Processor()` в этом рецепте — это функция, которая зацикливается до тех пор, пока ее контекст не будет отменен либо посредством явных вызовов отмены, либо через таймаут. Он отправляет всю работу в `Process()`, который может обрабатывать различные функции при различных операциях. Также было бы возможно, чтобы каждый из этих случаев отправлял другую функцию для еще более модульного кода.

В конечном итоге ответ возвращается в канал ответа, и мы повторяем цикл и печатаем все результаты в самом конце. Мы также демонстрируем случай ошибки в примере с `divide by 0`.

## Использование шаблона проектирования пула рабочих процессов

Шаблон проектирования рабочего пула — это тот, где вы отправляете долго работающие горютины в качестве рабочих. Эти обработчики могут выполнять разнообразную работу либо с использованием нескольких каналов, либо с помощью структуры запроса с отслеживанием состояния, которая указывает тип, как описано в предыдущем рецепте. Этот рецепт создаст воркеры с отслеживанием состояния и продемонстрирует, как координировать и запускать несколько воркеров, которые одновременно обрабатывают запросы на одном и том же канале. Эти воркеры будут `crypto` воркерами, как в приложении веб-аутентификации. Их целью будет хеширование строк открытого текста с помощью пакета `bcrypt` и сравнение текстового пароля с хешем.

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter10/pool` и перейдите в него.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter10/pool
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter10/pool
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter10/pool` или используйте это как возможность написать свой собственный код!
- Создайте файл с именем `worker.go` со следующим содержимым:

```
package pool

import (
    "context"
    "fmt"
)

// Dispatch creates numWorker workers, returns a
cancel // function channels for adding work and
responses,
// cancel must be called
func Dispatch(numWorker int) (context.CancelFunc,
chan WorkRequest, chan WorkResponse) {
    ctx := context.Background()
    ctx, cancel := context.WithCancel(ctx)
    in := make(chan WorkRequest, 10)
    out := make(chan WorkResponse, 10)

    for i := 0; i < numWorker; i++ {
        go Worker(ctx, i, in, out)
    }
    return cancel, in, out
}

// Worker loops forever and is part of the worker
pool
func Worker(ctx context.Context, id int, in chan
```

```

WorkRequest,
    out chan WorkResponse) {
    for {
        select {
            case <-ctx.Done():
                return
            case wr := <-in:
                fmt.Printf("worker id: %d,
performing %s
                                workn", id, wr.Op)
                                out <- Process(wr)
        }
    }
}

```

- Создайте файл с именем `work.go` со следующим содержимым:

```

package pool

import "errors"

type op string

const (
    // Hash is the bcrypt work type
    Hash op = "encrypt"
    // Compare is bcrypt compare work
    Compare = "decrypt"
)

// WorkRequest is a worker req
type WorkRequest struct {
    Op op
    Text []byte
    Compare []byte // optional
}

// WorkResponse is a worker resp
type WorkResponse struct {
    Wr WorkRequest
    Result []byte
    Matched bool
    Err error
}

```

```

    }

    // Process dispatches work to the worker pool
channel
func Process(wr WorkRequest) WorkResponse {
    switch wr.Op {
    case Hash:
        return hashWork(wr)
    case Compare:
        return compareWork(wr)
    default:
        return WorkResponse{Err:
errors.New("unsupported
        operation")}
    }
}

```

- Создайте файл с именем `crypto.go` со следующим содержимым:

```

package pool

import "golang.org/x/crypto/bcrypt"

func hashWork(wr WorkRequest) WorkResponse {
    val, err :=
bcrypt.GenerateFromPassword(wr.Text,
    bcrypt.DefaultCost)
    return WorkResponse{
        Result: val,
        Err: err,
        Wr: wr,
    }
}

func compareWork(wr WorkRequest) WorkResponse {
    var matched bool
    err :=
bcrypt.CompareHashAndPassword(wr.Compare, wr.Text)
    if err == nil {
        matched = true
    }
    return WorkResponse{
        Matched: matched,
    }
}

```

```

        Err: err,
        Wr: wr,
    }
}

```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "fmt"

    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter10/pool"
)

func main() {
    cancel, in, out := pool.Dispatch(10)
    defer cancel()

    for i := 0; i < 10; i++ {
        in <- pool.WorkRequest{Op: pool.Hash,
Text:    []byte(fmt.Sprintf("messages %d", i))}
    }

    for i := 0; i < 10; i++ {
        res := <-out
        if res.Err != nil {
            panic(res.Err)
        }
        in <- pool.WorkRequest{Op: pool.Compare,
Text:    res.Wr.Text, Compare: res.Result}
    }

    for i := 0; i < 10; i++ {
        res := <-out
        if res.Err != nil {
            panic(res.Err)
        }
    }
}

```



```

        fmt.Printf("string: \"%s\"; matched: %vn",
            string(res.Wr.Text), res.Matched)
    }
}

```

- Выполните `go run main.go`.
- Вы также можете запустить следующие команды:

```

$ go build
$ ./example

```

Вы также можете запустить следующие команды:

```

$ go run main.go
worker id: 9, performing encrypt work
worker id: 5, performing encrypt work
worker id: 2, performing encrypt work
worker id: 8, performing encrypt work
worker id: 6, performing encrypt work
worker id: 1, performing encrypt work
worker id: 0, performing encrypt work
worker id: 4, performing encrypt work
worker id: 3, performing encrypt work
worker id: 7, performing encrypt work
worker id: 2, performing decrypt work
worker id: 6, performing decrypt work
worker id: 8, performing decrypt work
worker id: 1, performing decrypt work
worker id: 0, performing decrypt work
worker id: 9, performing decrypt work
worker id: 3, performing decrypt work
worker id: 4, performing decrypt work
worker id: 7, performing decrypt work
worker id: 5, performing decrypt work
string: "messages 9"; matched: true
string: "messages 3"; matched: true
string: "messages 4"; matched: true
string: "messages 0"; matched: true
string: "messages 1"; matched: true
string: "messages 8"; matched: true
string: "messages 5"; matched: true
string: "messages 7"; matched: true
string: "messages 2"; matched: true
string: "messages 6"; matched: true

```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

В этом рецепте используется метод `Dispatch()` для создания нескольких рабочих процессов на одном входном канале, выходном канале и тех, которые подключены к одной функции `cancel()`. Это можно использовать, если вы хотите сделать разные пулы для разных целей. Например, вы можете создать 10 `crypto` и 20 `compare` рабочих процессов, используя отдельные пулы. В этом рецепте мы используем один пул, отправляем хеш-запросы рабочим процессам, получаем ответы, а затем отправляем запросы `compare` в тот же пул. Из-за этого рабочий, выполняющий работу, каждый раз будет другим, но все они способны выполнять любой тип работы.

Преимущество этого подхода заключается в том, что оба запроса допускают параллелизм, а также могут контролировать максимальный параллелизм. Ограничение максимального количества горутин также может быть важно для ограничения памяти. Я выбрал `crypto` для этого рецепта, потому что `crypto` — хороший пример кода, который может перегрузить ваш процессор или память, если вы запускаете новую Горути для каждого нового запроса; например, в веб-сервисе.

## Использование воркеров для создания пайплайнов

Этот рецепт демонстрирует создание групп рабочих пулов и их соединение для формирования конвейера. В этом рецепте мы связываем вместе два пула, но шаблон можно использовать для гораздо более сложных операций, подобных промежуточному программному обеспечению.

Пулы рабочих процессов могут быть полезны для обеспечения относительной простоты рабочих процессов, а также для дальнейшего контроля параллелизма. Например, может быть полезно сериализовать

ведение журнала при параллельном выполнении других операций. Также может быть полезно иметь меньший пул для более дорогостоящих операций, чтобы не перегружать ресурсы компьютера.

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter10/pipeline` и перейдите к нему.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter10/pipeline
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter10/pipeline
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter10/pipeline` или используйте это как возможность написать свой собственный код!
- Создайте файл с именем `worker.go` со следующим содержимым:

```
package pipeline

import "context"

// Worker have one role
// that is determined when
// Work is called
type Worker struct {
    in chan string
    out chan string
}

// Job is a job a worker can do
type Job string

const (
```

```

        // Print echo's all input to
        // stdout
        Print Job = "print"
        // Encode base64 encodes input
        Encode Job = "encode"
    )

    // Work is how to dispatch a worker, they are
assigned
    // a job here
    func (w *Worker) Work(ctx context.Context, j Job)
{
    switch j {
    case Print:
        w.Print(ctx)
    case Encode:
        w.Encode(ctx)
    default:
        return
    }
}

```

- Создайте файл с именем `print.go` со следующим содержимым:

```

package pipeline

import (
    "context"
    "fmt"
)

// Print prints w.in and repalys it
// on w.out
func (w *Worker) Print(ctx context.Context) {
    for {
        select {
        case <-ctx.Done():
            return
        case val := <-w.in:
            fmt.Println(val)
            w.out <- val
        }
    }
}

```

```
    }
}
```

- Создайте файл с именем `encode.go` со следующим содержимым:

```
package pipeline

import (
    "context"
    "encoding/base64"
    "fmt"
)

// Encode takes plain text as int
// and returns "string => <base64 string encoding>
// as out
func (w *Worker) Encode(ctx context.Context) {
    for {
        select {
            case <-ctx.Done():
                return
            case val := <-w.in:
                w.out <- fmt.Sprintf("%s => %s",
val,
base64.StdEncoding.EncodeToString([]byte(val)))
        }
    }
}
```

- Создайте файл с именем `pipeline.go` со следующим содержимым:

```
package pipeline

import "context"

// NewPipeline initializes the workers and
// connects them, it returns the input of the
pipeline
// and the final output
func NewPipeline(ctx context.Context, numEncoders,
numPrinters
int) (chan string, chan string) {
```

```

inEncode := make(chan string, numEncoders)
inPrint := make(chan string, numPrinters)
outPrint := make(chan string, numPrinters)
for i := 0; i < numEncoders; i++ {
    w := Worker{
        in: inEncode,
        out: inPrint,
    }
    go w.Work(ctx, Encode)
}

for i := 0; i < numPrinters; i++ {
    w := Worker{
        in: inPrint,
        out: outPrint,
    }
    go w.Work(ctx, Print)
}
return inEncode, outPrint
}

```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "context"
    "fmt"

    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter10/pipeline"
)

func main() {
    ctx := context.Background()
    ctx, cancel := context.WithCancel(ctx)
    defer cancel()

    in, out := pipeline.NewPipeline(ctx, 10, 2)

    go func() {

```

```

        for i := 0; i < 20; i++ {
            in <- fmt.Sprintf("Message", i)
        }
    }()

    for i := 0; i < 20; i++ {
        <-out
    }
}

```

- Выполните `go run main.go`.
- Вы также можете запустить следующие команды:

```

$ go build
$ ./example

```

Вы также можете запустить следующие команды:

```

$ go run main.go
Message3 => TWVzc2FnZTM=
Message7 => TWVzc2FnZTc=
Message8 => TWVzc2FnZTg=
Message9 => TWVzc2FnZTk=
Message5 => TWVzc2FnZTU=
Message11 => TWVzc2FnZTEw
Message10 => TWVzc2FnZTEw
Message4 => TWVzc2FnZTQ=
Message12 => TWVzc2FnZTEy
Message6 => TWVzc2FnZTY=
Message14 => TWVzc2FnZTE0
Message13 => TWVzc2FnZTEz
Message0 => TWVzc2FnZTA=
Message15 => TWVzc2FnZTE1
Message1 => TWVzc2FnZTE=
Message17 => TWVzc2FnZTE3
Message16 => TWVzc2FnZTE2
Message19 => TWVzc2FnZTE5
Message18 => TWVzc2FnZTE4
Message2 => TWVzc2FnZTI=

```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь,

что все тесты пройдены.

## Как это работает...

Пакет `main` создает конвейер, состоящий из 10 энкодеров и 2 принтеров. Он ставит в очередь 20 строк на входном канале и ожидает 20 ответов на выходном канале. Если сообщения достигают выходного канала, это означает, что они успешно прошли весь конвейер.

Функция `NewPipeline` используется для подключения пулов. Это гарантирует, что каналы создаются с правильно буферизованными размерами и что выходные каналы некоторых пулов подключены к соответствующим входным каналам других пулов. Также возможно разветвить конвейер, используя массив входных каналов и массив выходных каналов для каждого рабочего процесса, несколько именованных каналов или карты каналов. Это позволило бы на каждом этапе отправлять сообщения в регистратор.



# 11. Распределенные системы

Иногда параллелизма на уровне приложения недостаточно, и вещи, кажущиеся простыми в разработке, могут усложниться во время развертывания. Распределенные системы создают ряд проблем, которых нет при разработке на одной машине. Эти приложения усложнили такие вещи, как мониторинг, написание приложений, требующих строгих гарантий согласованности, и обнаружение сервисов. Кроме того, вы всегда должны помнить об единых точках отказа, таких как база данных, иначе ваши распределенные приложения могут выйти из строя, когда этот единственный компонент выйдет из строя.

В этой главе будут рассмотрены методы управления распределенными данными, оркестровки, контейнеризации, метрик и мониторинга. Они станут частью вашего набора инструментов для написания и поддержки микросервисов и больших распределенных приложений.

В этой главе мы рассмотрим следующие рецепты:

- Использование службы обнаружения с Consul
- Реализация базового консенсуса с использованием Raft
- Использование контейнеризации с Docker
- Стратегии оркестрации и развертывания
- Приложения для мониторинга
- Сбор метрик

## Технические требования

Чтобы следовать всем рецептам этой главы, настройте свою среду в соответствии со следующими шагами:

- Загрузите и установите Go 1.12.6 или более позднюю версию в своей операционной системе по адресу <https://golang.org/doc/install>.
- Install Consul from <https://www.consul.io/intro/getting-started/install.html>.
- Откройте терминал или консольное приложение, создайте и перейдите в каталог проекта, например `~/projects/go-programming-cookbook`. Весь наш код будет запускаться и изменяться из этого каталога.

- Скопируйте последний код в `~/projects/go-programming-cookbook-original` и (необязательно) работайте из этого каталога, а не вводите примеры вручную:

```
$ git clone git@github.com:PacktPublishing/Go-Programming-Cookbook-Second-Edition.git go-programming-cookbook-original
```

## Использование службы обнаружения с Consul

При использовании микросервисного подхода к приложениям вы получаете множество серверов, прослушивающих различные IP-адреса, домены и порты. Эти IP-адреса будут различаться в зависимости от среды (промежуточная или производственная), и может быть сложно сохранить их статическими для настройки между службами. Вы также хотите знать, когда компьютер или служба отключены или недоступны из-за сетевого раздела. Сетевой раздел возникает, когда две части сети не могут связаться друг с другом. Например, если происходит сбой коммутатора между двумя центрами обработки данных, службы в одном центре обработки данных не могут получить доступ к службам в другом центре обработки данных. Consul — это инструмент, предоставляющий множество функций, но здесь мы рассмотрим регистрацию сервисов в Consul и запрос к ним из других наших сервисов.

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter11/discovery` и перейдите к нему.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter11/discovery
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

`module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter11/discovery`

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter11/discovery` или используйте это как возможность написать свой собственный код!
- Создайте файл `client.go` со следующим содержимым:

```
package discovery

import "github.com/hashicorp/consul/api"

// Client exposes api methods we care
// about
type Client interface {
    Register(tags []string) error
    Service(service, tag string)
    ([]*api.ServiceEntry,
     *api.QueryMeta, error)
}

type client struct {
    client *api.Client
    address string
    name string
    port int
}

//NewClient initializes a consul client
func NewClient(config *api.Config, address, name
string, port
int) (Client, error) {
    c, err := api.NewClient(config)
    if err != nil {
        return nil, err
    }
    cli := &client{
        client: c,
        name: name,
        address: address,
        port: port,
    }
}
```

```
        return cli, nil
    }
}
```

- Создайте файл с именем `operations.go` со следующим содержимым:

```
package discovery

import "github.com/hashicorp/consul/api"

// Register adds our service to consul
func (c *client) Register(tags []string) error {
    reg := &api.AgentServiceRegistration{
        ID: c.name,
        Name: c.name,
        Port: c.port,
        Address: c.address,
        Tags: tags,
    }
    return c.client.Agent().ServiceRegister(reg)
}

// Service return a service
func (c *client) Service(service, tag string)
([]*api.ServiceEntry, *api.QueryMeta, error) {
    return c.client.Health().Service(service, tag,
false,
    nil)
}
```

- Создайте файл с именем `exec.go` со следующим содержимым:

```
package discovery

import "fmt"

// Exec creates a consul entry then queries it
func Exec(cli Client) error {
    if err := cli.Register([]string{"Go", "Awesome"}); err
!= nil {
        return err
    }

    entries, _, err := cli.Service("discovery", "Go")
}
```

```

    if err != nil {
        return err
    }
    for _, entry := range entries {
        fmt.Printf("%#v\n", entry.Service)
    }

    return nil
}

```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter11/discovery"

func main() {
    if err := discovery.Exec(); err != nil {
        panic(err)
    }
}

```

- Запустите Consul в отдельном Терминале с помощью команды `consul agent -dev -node=localhost`.
- Запустите команду `go run main.go`.
- Вы также можете запустить следующие команды:

```

$ go build
$ ./example

```

Вы должны увидеть следующий вывод:

```

$ go run main.go
&api.AgentService{ID:"discovery", Service:"discovery",
Tags:
[]string{"Go", "Awesome"}, Port:8080,
Address:"localhost",
EnableTagOverride:false, CreateIndex:0x23,
ModifyIndex:0x23}

```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Consul предоставляет надежную библиотеку Go API. Это может показаться пугающим, когда вы начинаете в первый раз, но этот рецепт показывает, как вы можете подойти к его обертыванию. Дальнейшая настройка Consul выходит за рамки этого рецепта; здесь показаны основы регистрации службы и запроса других служб при наличии ключа и тега.

Это можно было бы использовать для регистрации новых микрослужб во время запуска, запроса всех зависимых служб и отмены регистрации при завершении работы. Вы также можете кэшировать эту информацию, чтобы не нажимать Consul для каждого запроса, но этот рецепт предоставляет основные инструменты, которые вы можете расширить. Агент Consul также делает эти повторные запросы быстрыми и эффективными (<https://www.consul.io/intro/getting-started/agent.html>).

## Реализация базового консенсуса с использованием Raft

Raft — алгоритм консенсуса. Это позволяет распределенным системам сохранять общее и управляемое состояние (<https://raft.github.io/>). Настройка системы Raft сложна во многих отношениях — во-первых, вам нужен консенсус, чтобы выборы состоялись и прошли успешно. Это может быть сложно запустить, когда вы работаете с несколькими узлами, и вам может быть трудно начать работу. Базовый кластер можно запустить на одном узле/лидере. Однако, если вам нужна избыточность, необходимо как минимум три узла, чтобы предотвратить потерю данных в случае отказа одного узла. Эта концепция известна как кворум, при которой вы должны поддерживать  $(n/2)+1$  доступных узлов, чтобы обеспечить возможность фиксации новых журналов в кластере Raft. По сути, если вы можете поддерживать кворум, кластер остается работоспособным и пригодным для использования.

Этот рецепт реализует базовый кластер Raft в памяти, создает конечный автомат, который может переходить между определенными разрешенными состояниями, и подключает распределенный конечный автомат к веб-обработчику, который может инициировать переход. Это может быть полезно при реализации базового интерфейса конечного автомата, который требуется для Raft, или при тестировании. Этот рецепт использует <https://github.com/hashicorp/raft> для базовой реализации Raft.

## Как это сделать...

Следующие шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter11/consensus` и перейдите к нему.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter11/consensus
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter11/consensus
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter11/consensus` или используйте это как возможность написать свой собственный код!
- Создайте файл с именем `state.go` со следующим содержимым:

```
package consensus

type state string

const (
    first state = "first"
    second = "second"
    third = "third"
)

var allowedState map[state][]state
```

```

func init() {
    // setup valid states
    allowedState = make(map[state][]state)
    allowedState[first] = []state{second, third}
    allowedState[second] = []state{third}
    allowedState[third] = []state{first}
}

// CanTransition checks if a new state is valid
func (s *state) CanTransition(next state) bool {
    for _, n := range allowedState[*s] {
        if n == next {
            return true
        }
    }
    return false
}

// Transition will move a state to the next
// state if able
func (s *state) Transition(next state) {
    if s.CanTransition(next) {
        *s = next
    }
}

```

- Создайте файл с именем `raftset.go` со следующим содержимым:

```

package consensus

import (
    "fmt"

    "github.com/hashicorp/raft"
)

// keep a map of rafts for later
var rafts map[raft.ServerAddress]*raft.Raft

func init() {
    rafts = make(map[raft.ServerAddress]*raft.Raft)
}

```



```

// raftSet stores all the setup material we need
type raftSet struct {
    Config *raft.Config
    Store *raft.InmemStore
    SnapShotStore raft.SnapshotStore
    FSM *FSM
    Transport raft.LoopbackTransport
    Configuration raft.Configuration
}

// generate n raft sets to bootstrap the raft cluster
func getRaftSet(num int) []*raftSet {
    rs := make([]*raftSet, num)
    servers := make([]raft.Server, num)
    for i := 0; i < num; i++ {
        addr := raft.ServerAddress(fmt.Sprintf(i))
        _, transport := raft.NewInmemTransport(addr)
        servers[i] = raft.Server{
            Suffrage: raft.Voter,
            ID: raft.ServerID(addr),
            Address: addr,
        }
        config := raft.DefaultConfig()
        config.LocalID = raft.ServerID(addr)

        rs[i] = &raftSet{
            Config: config,
            Store: raft.NewInmemStore(),
            SnapShotStore: raft.NewInmemSnapshotStore(),
            FSM: NewFSM(),
            Transport: transport,
        }
    }

    // configuration needs to be consistent between
    // services and so we need the full serverlist in this
    // case
    for _, r := range rs {
        r.Configuration = raft.Configuration{Servers:
servers}
    }
}

```

```
    return rs
}
```

- Создайте файл с именем `config.go` со следующим содержимым:

```
package consensus

import (
    "github.com/hashicorp/raft"
)

// Config creates num in-memory raft
// nodes and connects them
func Config(num int) {

    // create n "raft-sets" consisting of
    // everything needed to represent a node
    rs := getRaftSet(num)

    //connect all of the transports
    for _, r1 := range rs {
        for _, r2 := range rs {
            r1.Transport.Connect(r2.Transport.LocalAddr(),
r2.Transport)
        }
    }

    // for each node, bootstrap then connect
    for _, r := range rs {
        if err := raft.BootstrapCluster(r.Config, r.Store,
r.Store, r.SnapShotStore, r.Transport, r.Configuration);
err != nil {
            panic(err)
        }
        raft, err := raft.NewRaft(r.Config, r.FSM, r.Store,
r.Store, r.SnapShotStore, r.Transport)
        if err != nil {
            panic(err)
        }
        rafts[r.Transport.LocalAddr()] = raft
    }
}
```

- Создайте файл с именем `fsm.go` со следующим содержимым:

```

package consensus

import (
    "io"

    "github.com/hashicorp/raft"
)

// FSM implements the raft FSM interface
// and holds a state
type FSM struct {
    state state
}

// NewFSM creates a new FSM with
// start state of "first"
func NewFSM() *FSM {
    return &FSM{state: first}
}

// Apply updates our FSM
func (f *FSM) Apply(r *raft.Log) interface{} {
    f.state.Transition(state(r.Data))
    return string(f.state)
}

// Snapshot needed to satisfy the raft FSM
interface {
    func (f *FSM) Snapshot() (raft.FSMSnapshot, error)

    return nil, nil
}

// Restore needed to satisfy the raft FSM interface
func (f *FSM) Restore(io.ReadCloser) error {
    return nil
}

```

- Создайте файл с именем **handler.go** со следующим содержимым:

```

package consensus

import (
    "net/http"

```

```

    "time"
)

// Handler grabs the get param ?next= and tries
// to transition to the state contained there
func Handler(w http.ResponseWriter, r *http.Request) {
    r.ParseForm()
    state := r.FormValue("next")
    for address, raft := range rafts {
        if address != raft.Leader() {
            continue
        }

        result := raft.Apply([]byte(state), 1*time.Second)
        if result.Error() != nil {
            w.WriteHeader(http.StatusBadRequest)
            return
        }
        newState, ok := result.Response().(string)
        if !ok {
            w.WriteHeader(http.StatusInternalServerError)
            return
        }

        if newState != state {
            w.WriteHeader(http.StatusBadRequest)
            w.Write([]byte("invalid transition"))
            return
        }
        w.WriteHeader(http.StatusOK)
        w.Write([]byte(newState))
        return
    }
}

```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "net/http"

    "github.com/PacktPublishing/

```

```

        Go-Programming-Cookbook-Second-Edition/
        chapter11/consensus"
    )

    func main() {
        consensus.Config(3)

        http.HandleFunc("/", consensus.Handler)
        err := http.ListenAndServe(":3333", nil)
        panic(err)
    }

```

- Запустите команду `go run main.go`. Alternatively, Вы также можете запустить следующие команды:

```

$ go build
$ ./example

```

Вы также можете запустить следующие команды:

```

$ go run main.go
2019/05/04 21:06:46 [INFO] raft: Initial configuration
(index=1): [{Suffrage:Voter ID:0 Address:0}
{Suffrage:Voter ID:1 Address:1} {Suffrage:Voter ID:2
Address:2}]
2019/05/04 21:06:46 [INFO] raft: Initial configuration
(index=1): [{Suffrage:Voter ID:0 Address:0}
{Suffrage:Voter ID:1 Address:1} {Suffrage:Voter ID:2
Address:2}]
2019/05/04 21:06:46 [INFO] raft: Node at 0 [Follower]
entering Follower state (Leader: "")
2019/05/04 21:06:46 [INFO] raft: Node at 1 [Follower]
entering Follower state (Leader: "")
2019/05/04 21:06:46 [INFO] raft: Initial configuration
(index=1): [{Suffrage:Voter ID:0 Address:0}
{Suffrage:Voter ID:1 Address:1} {Suffrage:Voter ID:2
Address:2}]
2019/05/04 21:06:46 [INFO] raft: Node at 2 [Follower]
entering Follower state (Leader: "")
2019/05/04 21:06:47 [WARN] raft: Heartbeat timeout from
"" reached, starting election
2019/05/04 21:06:47 [INFO] raft: Node at 0 [Candidate]
entering Candidate state in term 2
2019/05/04 21:06:47 [DEBUG] raft: Votes needed: 2

```

```

2019/05/04 21:06:47 [DEBUG] raft: Vote granted from 0 in
term 2. Tally: 1
2019/05/04 21:06:47 [DEBUG] raft: Vote granted from 1 in
term 2. Tally: 2
2019/05/04 21:06:47 [INFO] raft: Election won. Tally: 2
2019/05/04 21:06:47 [INFO] raft: Node at 0 [Leader]
entering Leader state
2019/05/04 21:06:47 [INFO] raft: Added peer 1, starting
replication
2019/05/04 21:06:47 [INFO] raft: Added peer 2, starting
replication
2019/05/04 21:06:47 [INFO] raft: pipelining replication
to peer {Voter 1 1}
2019/05/04 21:06:47 [INFO] raft: pipelining replication
to peer {Voter 2 2}

```

- В отдельном терминале выполните следующую команду:

```
$ curl "http://localhost:3333/?next=second"
second
```

```
$ curl "http://localhost:3333/?next=third"
third
```

```
$ curl "http://localhost:3333/?next=second"
invalid transition
```

```
$ curl "http://localhost:3333/?next=first"
first
```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Когда приложение запускается, мы инициализируем несколько объектов `Raft`. Каждый из них имеет свой адрес и транспорт. Функция `InmemTransport{}` также предоставляет метод для подключения других транспортов и называется `Connect()`. Как только эти соединения установлены, кластер `Raft` проводит выборы. При общении в кластере

Raft клиенты должны общаться с лидером. В нашем случае один обработчик может общаться со всеми узлами, поэтому обработчик отвечает за вызов объекта `Apply()` лидера Raft. Это, в свою очередь, запускает `apply()` на всех остальных узлах.

Функция `InmemTransport{}` упрощает процесс выбора и начальной загрузки, позволяя всему находиться в памяти. На практике это не очень полезно, за исключением тестирования и проверки концепций, поскольку горутины могут свободно обращаться к общей памяти. Более ориентированная на производство реализация будет использовать что-то вроде HTTP-транспорта, чтобы экземпляры службы могли обмениваться данными между машинами. Для этого может потребоваться дополнительный учет или обнаружение службы, поскольку экземпляры службы должны прослушивать и обслуживать, а также иметь возможность обнаруживать и устанавливать соединения друг с другом.

## Использование контейнеризации с Docker

Docker — это контейнерная технология для упаковки и доставки приложений. Другие преимущества включают переносимость, поскольку контейнер будет работать одинаково независимо от ОС хоста. Он предоставляет множество преимуществ виртуальной машины, но в более легком контейнере. Можно ограничить потребление ресурсов отдельными контейнерами и изолировать вашу среду. Может быть чрезвычайно полезно иметь общую среду для ваших приложений локально и когда вы отправляете свой код в производство. Docker написан на Go и имеет открытый исходный код, поэтому воспользоваться преимуществами клиента и библиотек просто. Этот рецепт настроит контейнер Docker для базового приложения Go, сохранит некоторую информацию о версии контейнера и продемонстрирует обращение к обработчику из конечной точки Docker.

### Подготовка

Настройте среду в соответствии с этими шагами:

- Обратитесь к разделу «*Технические требования*» в этой главе, чтобы узнать, как настроить вашу среду.
- Установите Docker с <https://docs.docker.com/install>. Это также будет включать Docker Compose.

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter11/docker` и перейдите к нему.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter11/docker
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter11/docker
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter11/docker` или используйте это как возможность написать свой собственный код!
- Создайте файл с именем `dockerfile` со следующим содержимым:

```
FROM alpine
```

```
ADD ./example/example /example
```

```
EXPOSE 8000
```

```
ENTRYPOINT /example
```

- Создайте файл с именем `setup.sh` со следующим содержимым:

```
#!/usr/bin/env bash
```

```
pushd example
```

```
env GOOS=linux go build -ldflags "-X
```

```
main.version=1.0 -X
```

```
main.builddate=$(date +%s)"
```

```
popd
```



```
docker build . -t example
docker run -d -p 8000:8000 example
```

- Создайте файл с именем `version.go` со следующим содержимым:

```
package docker

import (
    "encoding/json"
    "net/http"
    "time"
)

// VersionInfo holds artifacts passed in
// at build time
type VersionInfo struct {
    Version string
    BuildDate time.Time
    Uptime time.Duration
}

// VersionHandler writes the latest version info
func VersionHandler(v *VersionInfo)
http.HandlerFunc {
    t := time.Now()
    return func(w http.ResponseWriter, r
*http.Request) {
        v.Uptime = time.Since(t)
        vers, err := json.Marshal(v)
        if err != nil {
            w.WriteHeader
                (http.StatusInternalServerError)
            return
        }
        w.WriteHeader(http.StatusOK)
        w.Write(vers)
    }
}
```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```
package main
```

```

import (
    "fmt"
    "net/http"
    "strconv"
    "time"

    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter11/docker"
)

// these are set at build time
var (
    version string
    builddate string
)

var versioninfo docker.VersionInfo

func init() {
    // parse buildtime variables
    versioninfo.Version = version
    i, err := strconv.ParseInt(builddate, 10,
64)
        if err != nil {
            panic(err)
        }
        tm := time.Unix(i, 0)
        versioninfo.BuildDate = tm
    }

    func main() {
        http.HandleFunc("/version",
        docker.VersionHandler(&versioninfo))
        fmt.Printf("version %s listening on :8000\n",
        versioninfo.Version)
        panic(http.ListenAndServe(":8000", nil))
    }
}

```

- Вернитесь в исходный каталог.
- Выполните следующую команду:

```
$ bash setup.sh
```

Вы также можете запустить следующие команды:

```
$ bash setup.sh
~/go/src/github.com/PacktPublishing/Go-Programming-
Cookbook-
Second-Edition/chapter11/docker/example
~/go/src/github.com/PacktPublishing/Go-Programming-
Cookbook-
Second-Edition/chapter11/docker
~/go/src/github.com/PacktPublishing/Go-Programming-
Cookbook-
Second-Edition/chapter11/docker
Sending build context to Docker daemon 6.031 MB
Step 1/4 : FROM alpine
---> 4a415e366388
Step 2/4 : ADD ./example/example /example
---> de34c3c5451e
Removing intermediate container bdc9c4f4381
Step 3/4 : EXPOSE 8000
---> Running in 188f450d4e7b
---> 35d1a2652b43
Removing intermediate container 188f450d4e7b
Step 4/4 : ENTRYPOINT /example
---> Running in cf0af4f48c3a
---> 3d737fc4e6e2
Removing intermediate container cf0af4f48c3a
Successfully built 3d737fc4e6e2
b390ef429fbd6e7ff87058dc82e15c3e7a8b2e
69a601892700d1d434e9e8e43b
```

- Выполните следующие команды:

```
$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
b390ef429fbd example "/bin/sh -c /example" 22 seconds ago
Up 23
seconds 0.0.0.0:8000->8000/tcp optimistic_wescoff
```

```
$ curl localhost:8000/version
{"Version":"1.0","BuildDate":"2017-04-
30T21:55:56Z","Uptime":4813211264}
```

```
$docker kill optimistic_wescoff # grab from first output
optimistic_wescoff
```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Этот рецепт создал сценарий, который компилирует двоичный файл Go для архитектуры Linux и устанавливает различные частные переменные в `main.go`. Эти переменные используются для возврата информации о версии конечной точки версии. После компиляции двоичного файла создается контейнер Docker, содержащий двоичный файл. Это позволяет нам использовать очень маленькие образы контейнеров, поскольку среда выполнения Go является автономной в двоичном формате. Затем мы запускаем контейнер, открывая порт, на котором контейнер прослушивает HTTP-трафик. Наконец, мы `curl` порт на локальном хосте и видим, что возвращается информация о нашей версии.

## Стратегии оркестрации и развертывания

Docker значительно упрощает оркестрацию и развертывание. В этом рецепте мы настроим соединение с MongoDB, а затем вставим документ и запросим все это из контейнеров Docker. Этот рецепт настроит ту же среду, что и рецепт «*Использование NoSQL с MongoDB и tgo*» из [Главы 6](#) «*Все о базах данных и хранилище*», но запустит приложение и среду внутри контейнеров и будет использовать Docker Compose для оркестровки и подключения к ним.

Позже это можно использовать в сочетании с Docker Swarm, интегрированным инструментом Docker, который позволяет вам управлять кластером, создавать и развертывать узлы, которые можно легко увеличивать или уменьшать, а также управлять балансировкой нагрузки (<https://docs.docker.com/engine/swarm/>). Еще одним хорошим примером оркестрации контейнеров является Kubernetes (<https://kuberne>

[tes.io/](https://kubernetes.io/)), фреймворк для оркестровки контейнеров, написанный Google с использованием языка программирования Go.

## Как это сделать...

Следующие шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter11/orchestrate` и перейдите к нему.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter11/orchestrate
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter11/orchestrate
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter11/orchestrate` или используйте это как возможность написать свой собственный код!
- Создайте файл с именем `dockerfile` со следующим содержимым:

```
FROM golang:1.12.4-alpine3.9
```

```
ENV GOPATH /code/  
ADD . /code/src/github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter11/docker  
WORKDIR /code/src/github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter11/docker/example  
RUN GO111MODULE=on GOPROXY=off go build -mod=vendor
```

```
ENTRYPOINT /code/src/github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter11/docker/example/example
```

- Создайте файл с именем `docker-compose.yml` со следующим содержимым:

```

version: '2'
services:
  app:
    build: .
    mongodb:
      image: "mongo:latest"

```

- Создайте файл с именем **config.go** со следующим содержимым:

```

package mongodb

import (
    "context"
    "fmt"
    "time"

    "github.com/mongodb/mongo-go-driver/mongo"
    "go.mongodb.org/mongo-driver/mongo/options"
)

// Setup initializes a mongo client
func Setup(ctx context.Context, address string)
(*mongo.Client, error) {
    ctx, cancel := context.WithTimeout(ctx, 1*time.Second)
    defer cancel()

    fmt.Println(address)
    client, err :=
mongo.NewClient(options.Client().ApplyURI(address))
    if err != nil {
        return nil, err
    }

    if err := client.Connect(ctx); err != nil {
        return nil, err
    }
    return client, nil
}

```

- Создайте файл с именем **exec.go** со следующим содержимым:

```

package mongodb

```

```

import (
    "context"
    "fmt"

    "github.com/mongodb/mongo-go-driver/bson"
)

// State is our data model
type State struct {
    Name string `bson:"name"`
    Population int `bson:"pop"`
}

// Exec creates then queries an Example
func Exec(address string) error {
    ctx := context.Background()
    db, err := Setup(ctx, address)
    if err != nil {
        return err
    }

    conn := db.Database("gocookbook").Collection("example")

    vals := []interface{}{&State{"Washington", 7062000},
        &State{"Oregon", 3970000}}

    // we can inserts many rows at once
    if _, err := conn.InsertMany(ctx, vals); err != nil {
        return err
    }

    var s State
    if err := conn.FindOne(ctx, bson.M{"name":
        "Washington"}).Decode(&s); err != nil {
        return err
    }

    if err := conn.Drop(ctx); err != nil {
        return err
    }

    fmt.Printf("State: %#v\n", s)
}

```

```
    return nil
}
```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```
package main
```

```
import mongodb "github.com/PacktPublishing/Go-
Programming-Cookbook-Second-
Edition/chapter11/orchestrate"
```

```
func main() {
    if err := mongodb.Exec("mongodb://mongodb:27017"); err
    != nil {
        panic(err)
    }
}
```

- Вернитесь в исходный каталог.
- Запустите команду `go mod vendor`.
- Запустите команду `docker-compose up -d`.
- Run the `docker logs orchestrate_app_1` command. You should now see the following output:

```
$ docker logs orchestrate_app_1
State: docker.State{Name:"Washington",
Population:7062000}
```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Эта конфигурация хороша для локальной разработки. После запуска команды `docker-compose up` локальный каталог перестраивается, Docker устанавливает соединение с экземпляром MongoDB, используя последнюю версию, и начинает работать с ним. В этом рецепте для управления зависимостями используется `go mod vendor`. В результате



мы отключаем `go mod cache` и сообщаем команде `go build` использовать созданный нами каталог `vendor`.

Это может стать хорошей отправной точкой при запуске приложений, требующих подключения к внешним службам; все рецепты в [Главе 6 «Все о базах данных и хранилищах»](#) могут использовать этот подход, а не создавать локальный экземпляр базы данных. Для производства вы, вероятно, не захотите запускать хранилище данных за контейнером Docker, но у вас также обычно будут статические имена хостов для настройки.

## Приложения для мониторинга

Существует множество способов мониторинга приложений Go. Один из самых простых способов — настроить Prometheus, приложение для мониторинга, написанное на Go (<https://prometheus.io>). Это приложение, которое опрашивает конечную точку на основе вашего файла конфигурации и собирает много информации о вашем приложении, включая количество горутин, использование памяти и многое другое. Это приложение будет использовать методы из предыдущего рецепта для настройки среды Docker для размещения Prometheus и подключения к нему.

### Как это сделать...

Следующие шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter11/monitoring` и перейдите в него.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter11/monitoring
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter11/monitoring
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter11/monitoring` или используйте это как возможность написать свой собственный код!
- Создайте файл с именем `dockerfile` со следующим содержимым:

```
FROM golang:1.12.4-alpine3.9

ENV GOPATH /code/
ADD . /code/src/github.com/agtorre/go-
cookbook/chapter11/monitoring
WORKDIR /code/src/github.com/agtorre/go-
cookbook/chapter11/monitoring
RUN GO111MODULE=on GOPROXY=off go build -mod=vendor

ENTRYPOINT /code/src/github.com/agtorre/go-
cookbook/chapter11/monitoring/monitoring
```

- Создайте файл с именем `docker-compose.yml` со следующим содержимым:

```
version: '2'
services:
  app:
    build: .
  prometheus:
    ports:
      - 9090:9090
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    image: "prom/prometheus"
```

- Создайте файл с именем `main.go` со следующим содержимым:

```
package main

import (
    "net/http"

    "github.com/prometheus/client_golang/prometheus/promhttp"
)

func main() {
    http.Handle("/metrics", promhttp.Handler())
}
```

```
        panic(http.ListenAndServe(":80", nil))
    }
}
```

- Создайте файл с именем `prometheus.yml` со следующим содержимым:

```
global:
  scrape_interval: 15s # By default, scrape targets
every 15
  seconds.

# A scrape configuration containing exactly one
endpoint to
  scrape:
    # Here it's Prometheus itself.
  scrape_configs:
    # The job name is added as a label `job=
<job_name>` to any
    timeseries scraped from this config.
    - job_name: 'app'

# Override the global default and scrape targets
from this job
  every 5 seconds.
  scrape_interval: 5s

static_configs:
  - targets: ['app:80']
```

- Запустите команду `go mod vendor`.
- Запустите команду `docker-compose up`. Теперь вы должны увидеть следующий вывод:

```
$ docker-compose up
Starting monitoring_prometheus_1 ... done
Starting monitoring_app_1 ... done
Attaching to monitoring_app_1, monitoring_prometheus_1
prometheus_1 | time="2019-05-05T03:10:25Z" level=info
msg="Starting prometheus (version=1.6.1, branch=master,
revision=4666df502c0e239ed4aa1d80abbbfb54f61b23c3)"
source="main.go:88"
prometheus_1 | time="2019-05-05T03:10:25Z" level=info
msg="Build context (go=go1.8.1, user=root@7e45fa0366a7,
date=20170419-14:32:22)" source="main.go:89"
```

```
prometheus_1 | time="2019-05-05T03:10:25Z" level=info
msg="Loading configuration file
/etc/prometheus/prometheus.yml" source="main.go:251"
prometheus_1 | time="2019-05-05T03:10:25Z" level=info
msg="Loading series map and head chunks..."
source="storage.go:421"
prometheus_1 | time="2019-05-05T03:10:25Z" level=warning
msg="Persistence layer appears dirty."
source="persistence.go:846"
prometheus_1 | time="2019-05-05T03:10:25Z" level=warning
msg="Starting crash recovery. Prometheus is inoperational
until complete." source="crashrecovery.go:40"
prometheus_1 | time="2019-05-05T03:10:25Z" level=warning
msg="To avoid crash recovery in the future, shut down
Prometheus with SIGTERM or a HTTP POST to /-/quit."
source="crashrecovery.go:41"
prometheus_1 | time="2019-05-05T03:10:25Z" level=info
msg="Scanning files." source="crashrecovery.go:55"
prometheus_1 | time="2019-05-05T03:10:25Z" level=info
msg="File scan complete. 43 series found."
source="crashrecovery.go:83"
prometheus_1 | time="2019-05-05T03:10:25Z" level=info
msg="Checking for series without series file."
source="crashrecovery.go:85"
prometheus_1 | time="2019-05-05T03:10:25Z" level=info
msg="Check for series without series file complete."
source="crashrecovery.go:131"
prometheus_1 | time="2019-05-05T03:10:25Z" level=info
msg="Cleaning up archive indexes."
source="crashrecovery.go:411"
prometheus_1 | time="2019-05-05T03:10:25Z" level=info
msg="Clean-up of archive indexes complete."
source="crashrecovery.go:504"
prometheus_1 | time="2019-05-05T03:10:25Z" level=info
msg="Rebuilding label indexes."
source="crashrecovery.go:512"
prometheus_1 | time="2019-05-05T03:10:25Z" level=info
msg="Indexing metrics in memory."
source="crashrecovery.go:513"
prometheus_1 | time="2019-05-05T03:10:25Z" level=info
msg="Indexing archived metrics."
source="crashrecovery.go:521"
prometheus_1 | time="2019-05-05T03:10:25Z" level=info
```

```

msg="All requests for rebuilding the label indexes
queued. (Actual processing may lag behind.)"
source="crashrecovery.go:540"
prometheus_1 | time="2019-05-05T03:10:25Z" level=warning
msg="Crash recovery complete."
source="crashrecovery.go:153"
prometheus_1 | time="2019-05-05T03:10:25Z" level=info
msg="43 series loaded." source="storage.go:432"
prometheus_1 | time="2019-05-05T03:10:25Z" level=info
msg="Starting target manager..."
source="targetmanager.go:61"
prometheus_1 | time="2019-05-05T03:10:25Z" level=info
msg="Listening on :9090" source="web.go:259"

```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Перейдите в браузере по адресу <http://localhost:9090/>. Вы должны увидеть множество показателей, связанных с вашим приложением!

## Как это работает...

Этот рецепт создает простой обработчик в Go, который экспортирует статистику о запущенном приложении в Prometheus с помощью Prometheus Go клиента. Мы подключаем наше приложение к серверу Prometheus, работающему в докере, и обрабатываем сетевое подключение и запуск с помощью Docker Compose. Параметры частоты сбора данных, порт, который обслуживает приложение, и имя приложения указаны в файле `Prometheus.yml`. После запуска обоих контейнеров сервер Prometheus начинает сбор и мониторинг приложения на указанном порту. Он также предоставляет веб-интерфейс, который мы посещаем в браузере, чтобы увидеть больше информации о нашем приложении.

Обработчик клиента Prometheus возвращает различные статистические данные о вашем приложении на сервер Prometheus. Это позволяет указать несколько серверов Prometheus на приложение без необходимости перенастраивать или развертывать приложение. Большинство этих статистических данных являются общими и полезными для таких вещей, как обнаружение утечек памяти. Многие другие решения требуют от вас периодической отправки информации

на сервер. Следующий рецепт, «Сбор метрик», продемонстрирует, как отправлять пользовательские метрики на сервер Prometheus.

## Сбор метрик

В дополнение к общей информации о вашем приложении может быть полезно выдавать метрики, специфичные для приложения. Например, мы можем захотеть собрать данные о времени или отслеживать, сколько раз происходит событие.

Этот рецепт будет использовать пакет [github.com/rcrowley/go-metrics](https://github.com/rcrowley/go-metrics) для сбора метрик и предоставления их через конечную точку. Существуют различные инструменты экспорта, которые вы можете использовать для экспорта метрик в такие места, как Prometheus и InfluxDB, которые также написаны на Go.

## Подготовка

Настройте среду в соответствии с этими шагами:

- Обратитесь к разделу «Технические требования» в этой главе, чтобы узнать, как настроить вашу среду.
- Запустите команду `go get github.com/rcrowley/go-metrics`.

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter11/metrics` и перейдите в него.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter11/metrics
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter11/metrics
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter11/metrics` или используйте это как возможность написать свой собственный код!
- Создайте файл с именем `handler.go` со следующим содержимым:

```
package metrics

import (
    "net/http"
    "time"

    metrics "github.com/rcrowley/go-metrics"
)

// CounterHandler will update a counter each time
it's called
func CounterHandler(w http.ResponseWriter, r
*http.Request) {
    c :=
metrics.GetOrRegisterCounter("counterhandler.counter",
    nil)
    c.Inc(1)

    w.WriteHeader(http.StatusOK)
    w.Write([]byte("success"))
}

// TimerHandler records the duration required to
complete
func TimerHandler(w http.ResponseWriter, r
*http.Request) {
    currt := time.Now()
    t :=
metrics.GetOrRegisterTimer("timerhandler.timer", nil)

    w.WriteHeader(http.StatusOK)
    w.Write([]byte("success"))
    t.UpdateSince(currt)
}
```

- Создайте файл с именем `report.go` со следующим содержимым:

```

package metrics

import (
    "net/http"

    gometrics "github.com/rcrowley/go-metrics"
)

// ReportHandler will emit the current metrics in
json format
func ReportHandler(w http.ResponseWriter, r
*http.Request) {

    w.WriteHeader(http.StatusOK)

    t := gometrics.GetOrRegisterTimer(
        "reporhandler.writemetrics", nil)
    t.Time(func() {

gometrics.WriteJSONOnce(gometrics.DefaultRegistry, w)
    })
}

```

- Создайте новый каталог с именем **example** и перейдите к нему.
- Создайте файл с именем **main.go**:

```

package main

import (
    "net/http"

    "github.com/PacktPublishing/
Go-Programming-Cookbook-Second-Edition/
chapter11/metrics"
)

func main() {
    // handler to populate metrics
    http.HandleFunc("/counter",
metrics.CounterHandler)
    http.HandleFunc("/timer", metrics.TimerHandler)
    http.HandleFunc("/report",
metrics.ReportHandler)
    fmt.Println("listening on :8080")
}

```



```
        panic(http.ListenAndServe(":8080", nil))
    }
}
```

- Выполните `go run main.go`. Alternatively, Вы также можете запустить следующую команду:

```
$ go build
$ ./example
```

Вы также можете запустить следующие команды:

```
$ go run main.go
listening on :8080
```

- Запустите следующие команды из отдельной оболочки:

```
$ curl localhost:8080/counter
success
```

```
$ curl localhost:8080/timer
success
```

```
$ curl localhost:8080/report
{"counterhandler.counter":{"count":1},
"reporthandler.writemetrics":
{"15m.rate":0,"1m.rate":0,"5m.rate":0,"75%":0,"95%":0,"99%":0,"99.9%":0,"count":0,"max":0,"mean":0,"mean.rate":0,"median":0,"min":0,"stddev":0},"timerhandler.timer":
{"15m.rate":0.0011080303990206543,"1m.rate":0.015991117074135343,"5m.rate":0.0033057092356765017,"75%":60485,"95%":60485,"99%":60485,"99.9%":60485,"count":1,"max":60485,"mean":60485,"mean.rate":1.1334543719787356,"median":60485,"min":60485,"stddev":0}}
```

- Попробуйте нажать все конечные точки еще несколько раз, чтобы посмотреть, как они изменяются.
- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

`gometrics` хранит все ваши показатели в реестре. После настройки вы можете использовать любой из параметров генерации метрик, например `counter` или `timer`, и он сохранит это обновление в реестре. Существует несколько экспортеров, которые экспортируют метрики в сторонние инструменты. В нашем случае мы настроили обработчик, который выдает все метрики в формате JSON.

Мы настроили три обработчика: один увеличивает счетчик, другой записывает время до выхода из обработчика и третий печатает отчет (одновременно увеличивая дополнительный счетчик). Функции `GetOrRegister` полезны для атомарного получения или создания генератора метрик, если он в настоящее время не существует потокобезопасным способом. Кроме того, вы можете зарегистрировать все один раз заранее.

# 12. Реактивное программирование и потоки данных

В этой главе мы обсудим шаблоны проектирования реактивного программирования в Go. Реактивное программирование — это концепция программирования, которая фокусируется на потоках данных и распространении изменений. Такие технологии, как Kafka, позволяют быстро создавать или потреблять поток данных. В результате эти технологии естественным образом подходят друг другу. В рецепте «Подключение Kafka к Goflow» мы рассмотрим объединение очереди сообщений `kafka` с `goflow`, чтобы показать практический пример использования этих технологий. В этой главе также будут рассмотрены различные способы подключения к Kafka и использования его для обработки сообщений. Наконец, в этой главе будет показано, как создать базовый сервер `graphql` в Go.

В этой главе мы рассмотрим следующие рецепты:

- Использование Goflow для программирования потока данных
- Использование Kafka с Sarama
- Использование асинхронных продюсеров с Kafka
- Подключение Kafka к Goflow
- Пишем сервер GraphQL на Go

## Технические требования

Чтобы продолжить выполнение всех рецептов в этой главе, настройте свою среду в соответствии со следующими шагами:

- Загрузите и установите Go 1.12.6 или более позднюю версию в своей операционной системе с <https://golang.org/doc/install>.
- Откройте терминал или консольное приложение, создайте и перейдите в каталог проекта, например `~/projects/go-programming-cookbook`. Весь код будет запускаться и изменяться из этого каталога.

- Скопируйте последний код в `~/projects/go-programming-cookbook-original` и, при желании, работайте из этого каталога, вместо того, чтобы вводить примеры вручную:

```
$ git clone git@github.com:PacktPublishing/Go-Programming-Cookbook-Second-Edition.git go-programming-cookbook-original
```

## Использование Goflow для программирования потока данных

Пакет `github.com/trustmaster/goflow` полезен для создания приложений на основе потока данных. Он пытается абстрагировать концепции, чтобы вы могли писать компоненты и соединять их вместе с помощью пользовательской сети. Этот рецепт воссоздает приложение, описанное в [Главе 9](#), «Тестирование кода Go», но сделает это с использованием пакета `goflow`.

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter12/goflow` и перейдите в этот каталог.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter12/goflow
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter12/goflow
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter12/goflow` или используйте это как упражнение для написания собственного кода!
- Создайте файл с именем `components.go` со следующим содержимым:

```

package goflow

import (
    "encoding/base64"
    "fmt"
)

// Encoder base64 encodes all input
type Encoder struct {
    Val <-chan string
    Res chan<- string
}

// Process does the encoding then pushes the result onto
Res
func (e *Encoder) Process() {
    for val := range e.Val {
        encoded :=
base64.StdEncoding.EncodeToString([]byte(val))
        e.Res <- fmt.Sprintf("%s => %s", val, encoded)
    }
}

// Printer is a component for printing to stdout
type Printer struct {
    Line <-chan string
}

// Process Prints the current line received
func (p *Printer) Process() {
    for line := range p.Line {
        fmt.Println(line)
    }
}

```

- Создайте файл с именем `network.go` со следующим содержимым:

```

package goflow

import (
    "github.com/trustmaster/goflow"
)

```

```
// NewEncodingApp wires together the components
func NewEncodingApp() *goflow.Graph {
    e := goflow.NewGraph()

    // define component types
    e.Add("encoder", new(Encoder))
    e.Add("printer", new(Printer))

    // connect the components using channels
    e.Connect("encoder", "Res", "printer", "Line")

    // map the in channel to Val, which is
    // tied to OnVal function
    e.MapInPort("In", "encoder", "Val")

    return e
}
```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```
package main

import (
    "fmt"

    "github.com/PacktPublishing/
        Go-Programming-Cookbook-Second-
        Edition/chapter12/goflow"
    flow "github.com/trustmaster/goflow"
)

func main() {

    net := goflow.NewEncodingApp()

    in := make(chan string)
    net.SetInPort("In", in)

    wait := flow.Run(net)

    for i := 0; i < 20; i++ {
        in <- fmt.Sprintf("Message", i)
    }
}
```

```

    }

    close(in)
    <-wait
}

```

- Выполните `go run main.go`.
- Вы также можете запустить следующие команды:

```

$ go build
$ ./example

```

Вы также можете запустить следующие команды:

```

$ go run main.go
Message6 => TWVzc2FnZTY=
Message5 => TWVzc2FnZTU=
Message1 => TWVzc2FnZTE=
Message0 => TWVzc2FnZTA=
Message4 => TWVzc2FnZTQ=
Message8 => TWVzc2FnZTg=
Message2 => TWVzc2FnZTI=
Message3 => TWVzc2FnZTM=
Message7 => TWVzc2FnZTc=
Message10 => TWVzc2FnZTEw
Message9 => TWVzc2FnZTk=
Message12 => TWVzc2FnZTEy
Message11 => TWVzc2FnZTEz
Message14 => TWVzc2FnZTE0
Message13 => TWVzc2FnZTEz
Message16 => TWVzc2FnZTE2
Message15 => TWVzc2FnZTE1
Message18 => TWVzc2FnZTE4
Message17 => TWVzc2FnZTE3
Message19 => TWVzc2FnZTE5

```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите команду `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Пакет [github.com/trustmaster/goflow](https://github.com/trustmaster/goflow) работает, определяя [network/graph](#), регистрируя некоторые компоненты, а затем связывая их вместе. Это может показаться немного подверженным ошибкам, поскольку компоненты описываются с помощью строк, но обычно это приводит к сбою на ранней стадии выполнения, пока приложение не настроено и не работает правильно.

В этом рецепте мы устанавливаем два компонента, один из которых кодирует входящую строку с помощью Base64, а другой печатает все, что ему передается. Мы подключаем его к входному каналу, который инициализирован в [main.go](#), и все, что передается на этот канал, будет проходить через наш конвейер.

Большой упор в этом подходе делается на игнорирование внутренней сути того, что происходит. Мы относимся ко всему как к подключенному черному ящику и позволяем [goflow](#) делать все остальное. Вы можете видеть в этом рецепте, насколько мал код для выполнения этого конвейера задач и что у нас меньше ручек для управления числом рабочих процессов, среди прочего.

## Использование Kafka с Sarama

Kafka — популярная распределенная очередь сообщений с множеством расширенных функций для построения распределенных систем. Этот рецепт покажет, как писать в тему Kafka с помощью синхронного продюсера и как использовать ту же тему с помощью потребителя раздела. В этом рецепте не рассматриваются различные конфигурации Kafka, так как это гораздо более широкая тема, выходящая за рамки данной книги, но я предлагаю начать с <https://kafka.apache.org/intro>.

## Подготовка

Настройте среду в соответствии с этими шагами:

- Обратитесь к разделу «*Технические требования*» в этой главе, чтобы узнать, как настроить вашу среду.



- Установите Kafka, выполнив шаги, указанные на странице [https://www.tutorialspoint.com/apache\\_kafka/apache\\_kafka\\_installation\\_steps.htm](https://www.tutorialspoint.com/apache_kafka/apache_kafka_installation_steps.htm).
- Кроме того, вы также можете получить доступ к <https://github.com/spotify/docker-kafka>.

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter12/synckafka` и перейдите в этот каталог.
- Запустите эту команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter12/synckafka
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter12/synckafka
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter12/synckafka` или используйте это как упражнение для написания собственного кода!
- Убедитесь, что Kafka запущена и работает на `localhost:9092`.
- Создайте файл с именем `main.go` в каталоге с именем `consumer` со следующим содержимым:

```
package main

import (
    "log"

    sarama "github.com/Shopify/sarama"
)

func main() {
    consumer, err :=
        sarama.NewConsumer([]string{"localhost:9092"},
```

```

nil)
    if err != nil {
        panic(err)
    }
    defer consumer.Close()

    partitionConsumer, err :=
    consumer.ConsumePartition("example", 0,
        sarama.OffsetNewest)
    if err != nil {
        panic(err)
    }
    defer partitionConsumer.Close()

    for {
        msg := <-partitionConsumer.Messages()
        log.Printf("Consumed message: \"%s\" at
offset: %d\n",
            msg.Value, msg.Offset)
    }
}

```

- Создайте файл с именем **main.go** в каталоге с именем **producer** со следующим содержимым:

```

package main

import (
    "fmt"
    "log"

    sarama "github.com/Shopify/sarama"
)

func sendMessage(producer sarama.SyncProducer,
value string) {
    msg := &sarama.ProducerMessage{Topic:
"example", Value:
    sarama.StringEncoder(value)}
    partition, offset, err :=
producer.SendMessage(msg)
}

```

```

        if err != nil {
            log.Printf("FAILED to send message: %s\n",
err)
            return
        }
        log.Printf("> message sent to partition %d at
offset %d\n",
partition, offset)
    }

    func main() {
        producer, err :=
sarama.NewSyncProducer([]string{"localhost:9092"}, nil)
        if err != nil {
            panic(err)
        }
        defer producer.Close()

        for i := 0; i < 10; i++ {
            sendMessage(producer, fmt.Sprintf("Message
%d", i))
        }
    }

```

- Перейдите вверх по каталогу.
- Запустите `go run ./consumer`.
- В отдельном терминале из того же каталога запустите `go run ./producer`.
- В `producer` терминале вы должны увидеть следующее:

```

$ go run ./producer
2017/05/07 11:50:38 > message sent to partition 0 at
offset 0
2017/05/07 11:50:38 > message sent to partition 0 at
offset 1
2017/05/07 11:50:38 > message sent to partition 0 at
offset 2
2017/05/07 11:50:38 > message sent to partition 0 at
offset 3
2017/05/07 11:50:38 > message sent to partition 0 at
offset 4

```

```

2017/05/07 11:50:38 > message sent to partition 0 at
offset 5
2017/05/07 11:50:38 > message sent to partition 0 at
offset 6
2017/05/07 11:50:38 > message sent to partition 0 at
offset 7
2017/05/07 11:50:38 > message sent to partition 0 at
offset 8
2017/05/07 11:50:38 > message sent to partition 0 at
offset 9

```

В `consumer` терминале вы должны увидеть следующее::

```

$ go run ./consumer
2017/05/07 11:50:38 Consumed message: "Message 0" at
offset: 0
2017/05/07 11:50:38 Consumed message: "Message 1" at
offset: 1
2017/05/07 11:50:38 Consumed message: "Message 2" at
offset: 2
2017/05/07 11:50:38 Consumed message: "Message 3" at
offset: 3
2017/05/07 11:50:38 Consumed message: "Message 4" at
offset: 4
2017/05/07 11:50:38 Consumed message: "Message 5" at
offset: 5
2017/05/07 11:50:38 Consumed message: "Message 6" at
offset: 6
2017/05/07 11:50:38 Consumed message: "Message 7" at
offset: 7
2017/05/07 11:50:38 Consumed message: "Message 8" at
offset: 8
2017/05/07 11:50:38 Consumed message: "Message 9" at
offset: 9

```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Этот рецепт демонстрирует передачу простых сообщений через Kafka. Более сложные методы должны использовать формат сериализации, такой как `json`, `gob`, `protobuf` или другие. Продюсер может отправить сообщение Kafka синхронно через `sendMessage`. Это плохо работает в случаях, когда кластер Kafka не работает, и может привести к зависанию процесса в этих случаях. Это важно учитывать для таких приложений, как веб-обработчики, поскольку это может привести к тайм-аутам и жестким зависимостям от кластера Kafka.

Предполагая, что очереди сообщений правильны, наш потребитель будет наблюдать за потоком Kafka и что-то делать с результатами. Предыдущие рецепты в этой главе могут использовать этот поток для выполнения дополнительной обработки.

## Использование асинхронных продюсеров с Kafka

Часто полезно не ждать, пока продюсер Kafka завершит работу, прежде чем переходить к следующей задаче. В таких случаях вы можете использовать асинхронного продюсера. Эти продюсеры принимают сообщения `Sarama` на канале и имеют методы для возврата канала успеха/ошибки, которые можно проверить отдельно.

В этом рецепте мы создадим подпрограмму Go, которая будет обрабатывать сообщения об успехе и неудаче, в то время как мы разрешаем обработчику ставить сообщения в очередь для отправки независимо от результата.

### Подготовка

Обратитесь к разделу «Подготовка» рецепта «Использование Kafka с `Sarama`».

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В своем терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter12/asynckafka` и перейдите в этот каталог.

- Запустите эту команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter12/asynckafka
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter12/asynckafka
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter12/asynckafka` или используйте это как упражнение для написания собственного кода!
- Убедитесь, что Kafka запущена и работает на `localhost:9092`.
- Скопируйте каталог `consumer` из предыдущего рецепта.
- Создайте каталог с именем `producer` и перейдите к нему.
- Создайте файл с именем `producer.go` со следующим содержимым:

```
package main

import (
    "log"

    sarama "github.com/Shopify/sarama"
)

// Process response grabs results and errors from
a producer
// asynchronously
func ProcessResponse(producer
sarama.AsyncProducer) {
    for {
        select {
            case result := <-producer.Successes():
                log.Printf("> message: \"%s\" sent to
partition
                                %d at offset %d\n", result.Value,
                                result.Partition, result.Offset)
            case err := <-producer.Errors():
                log.Println("Failed to produce
```

```

message", err)
    }
}

```

- Создайте файл с именем `handler.go` со следующим содержимым:

```

package main

import (
    "net/http"

    sarama "github.com/Shopify/sarama"
)

// KafkaController allows us to attach a producer
// to our handlers
type KafkaController struct {
    producer sarama.AsyncProducer
}

// Handler grabs a message from a GET parama and
// send it to the kafka queue asynchronously
func (c *KafkaController) Handler(w
http.ResponseWriter, r
*http.Request) {
    if err := r.ParseForm(); err != nil {
        w.WriteHeader(http.StatusBadRequest)
        return
    }

    msg := r.FormValue("msg")
    if msg == "" {
        w.WriteHeader(http.StatusBadRequest)
        w.Write([]byte("msg must be set"))
        return
    }

    c.producer.Input() <-
&sarama.ProducerMessage{Topic:
    "example", Key: nil, Value:
    sarama.StringEncoder(msg)}
    w.WriteHeader(http.StatusOK)
}

```

- Создайте файл с именем `main.go` со следующим содержимым:

```
package main

import (
    "fmt"
    "net/http"

    sarama "github.com/Shopify/sarama"
)

func main() {
    config := sarama.NewConfig()
    config.Producer.Return.Successes = true
    config.Producer.Return.Errors = true
    producer, err :=

sarama.NewAsyncProducer([]string{"localhost:9092"},
config)

    if err != nil {
        panic(err)
    }
    defer producer.AsyncClose()

    go ProcessResponse(producer)

    c := KafkaController{producer}
    http.HandleFunc("/", c.Handler)
    fmt.Println("Listening on port :3333")
    panic(http.ListenAndServe(":3333", nil))
}
```

- Перейдите вверх по каталогу.
- Запустите `go run ./consumer`.
- В отдельном терминале из того же каталога запустите `go run ./producer`.
- В третьем терминале выполните следующие команды:

```
$ curl "http://localhost:3333/?msg=this"
$ curl "http://localhost:3333/?msg=is"
$ curl "http://localhost:3333/?msg=an"
$ curl "http://localhost:3333/?msg=example"
```



В терминале `producer` вы должны увидеть следующее:

```
$ go run ./producer
Listening on port :3333
2017/05/07 13:52:54 > message: "this" sent to partition 0
at offset 0
2017/05/07 13:53:25 > message: "is" sent to partition 0
at offset 1
2017/05/07 13:53:27 > message: "an" sent to partition 0
at offset 2
2017/05/07 13:53:29 > message: "example" sent to
partition 0 at offset 3
```

- В терминале `consumer` вы должны увидеть следующее:

```
$ go run ./consumer
2017/05/07 13:52:54 Consumed message: "this" at offset: 0
2017/05/07 13:53:25 Consumed message: "is" at offset: 1
2017/05/07 13:53:27 Consumed message: "an" at offset: 2
2017/05/07 13:53:29 Consumed message: "example" at
offset: 3
```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Все наши модификации в этой главе были сделаны продюсером. На этот раз мы создали отдельную процедуру Go для обработки успехов и ошибок. Если их оставить необработанными, ваше приложение заблокируется. Затем мы привязываем нашего производителя к обработчику и посылаем ему сообщения всякий раз, когда сообщение получено, через вызов `GET` к обработчику.

Обработчик немедленно вернет успех после отправки сообщения, независимо от его ответа. Если это неприемлемо, следует использовать синхронный подход. В нашем случае мы согласны с последующей обработкой успехов и ошибок отдельно.

Наконец, мы `curl` нашу конечную точку с несколькими разными сообщениями, и вы можете видеть, как они перетекают из обработчика туда, где они в конечном итоге печатаются потребителем Kafka, который мы написали в предыдущем разделе.

## Подключение Kafka к Goflow

Этот рецепт объединит потребителя Kafka с конвейером Goflow. Когда наш потребитель получает сообщения от Kafka, он запускает для них `string.ToUpper()`, а затем выводит результаты. Они естественным образом сочетаются друг с другом, поскольку Goflow предназначен для работы с входящим потоком, что и предоставляет нам Kafka.

### Подготовка

Обратитесь к разделу «Подготовка» рецепта «Использование Kafka с Sarama».

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter12/kafkaflow` и перейдите в этот каталог.
- Запустите эту команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter12/kafkaflow
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter12/kafkaflow
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter12/kafkaflow` или используйте это как упражнение для написания собственного кода!
- Убедитесь, что Kafka запущена и работает на `localhost:9092`.

- Создайте файл с именем `components.go` со следующим содержимым:

```
package kafkaflow

import (
    "fmt"
    "strings"

    flow "github.com/trustmaster/goflow"
)

// Upper upper cases the incoming
// stream
type Upper struct {
    Val <-chan string
    Res chan<- string
}

// Process loops over the input values and writes the
upper
// case string version of them to Res
func (e *Upper) Process() {
    for val := range e.Val {
        e.Res <- strings.ToUpper(val)
    }
}

// Printer is a component for printing to stdout
type Printer struct {
    flow.Component
    Line <-chan string
}

// Process Prints the current line received
func (p *Printer) Process() {
    for line := range p.Line {
        fmt.Println(line)
    }
}
```

- Создайте файл с именем `network.go` со следующим содержимым:

```

package kafkaflow

import "github.com/trustmaster/goflow"

// NewUpperApp wires together the components
func NewUpperApp() *goflow.Graph {
    u := goflow.NewGraph()

    u.Add("upper", new(Upper))
    u.Add("printer", new(Printer))

    u.Connect("upper", "Res", "printer", "Line")
    u.MapInPort("In", "upper", "Val")

    return u
}

```

- Создайте файл с именем `main.go` в каталоге с именем `consumer` со следующим содержимым:

```

package main

import (
    "github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter12/kafkaflow"
    sarama "github.com/Shopify/sarama"
    flow "github.com/trustmaster/goflow"
)

func main() {
    consumer, err :=
sarama.NewConsumer([]string{"localhost:9092"}, nil)
    if err != nil {
        panic(err)
    }
    defer consumer.Close()

    partitionConsumer, err :=
consumer.ConsumePartition("example", 0,
sarama.OffsetNewest)
    if err != nil {
        panic(err)
    }
}

```

```

defer partitionConsumer.Close()

net := kafkaflow.NewUpperApp()

in := make(chan string)
net.SetInPort("In", in)

wait := flow.Run(net)
defer func() {
    close(in)
    <-wait
}()

for {
    msg := <-partitionConsumer.Messages()
    in <- string(msg.Value)
}

}

```

- Скопируйте каталог `producer` из рецепта «*Использование Kafka с Sarama*».
- Run `go run ./consumer`.
- В отдельном терминале из того же каталога запустите `go run ./producer`.
- В терминале `producer` вы должны увидеть следующее:

```

$ go run ./producer
2017/05/07 18:24:12 > message "Message 0" sent to
partition 0 at offset 0
2017/05/07 18:24:12 > message "Message 1" sent to
partition 0 at offset 1
2017/05/07 18:24:12 > message "Message 2" sent to
partition 0 at offset 2
2017/05/07 18:24:12 > message "Message 3" sent to
partition 0 at offset 3
2017/05/07 18:24:12 > message "Message 4" sent to
partition 0 at offset 4
2017/05/07 18:24:12 > message "Message 5" sent to
partition 0 at offset 5
2017/05/07 18:24:12 > message "Message 6" sent to
partition 0 at offset 6
2017/05/07 18:24:12 > message "Message 7" sent to

```

```
partition 0 at offset 7
2017/05/07 18:24:12 > message "Message 8" sent to
partition 0 at offset 8
2017/05/07 18:24:12 > message "Message 9" sent to
partition 0 at offset 9
```

В терминале `consumer` вы должны увидеть следующее:

```
$ go run ./consumer
MESSAGE 0
MESSAGE 1
MESSAGE 2
MESSAGE 3
MESSAGE 4
MESSAGE 5
MESSAGE 6
MESSAGE 7
MESSAGE 8
MESSAGE 9
```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Этот рецепт сочетает в себе идеи из предыдущих рецептов в этой главе. Как и в предыдущих рецептах, мы настраиваем потребителя и производителя Kafka. В этом рецепте используется синхронный производитель из рецепта «*Использование Kafka с Sarama*», но вместо этого можно было бы использовать и асинхронного производителя. Как только сообщение получено, мы помещаем его в очередь на входном канале, как мы это делали в рецепте «*Goflow для программирования потока данных*». Мы модифицируем компоненты из этого рецепта, чтобы преобразовать нашу входящую строку в верхний регистр, а не кодировать ее в Base64. Мы повторно используем компоненты печати, и итоговая конфигурация сети аналогична.

Конечным результатом является то, что все сообщения, полученные через потребителя Kafka, транспортируются в наш рабочий конвейер на основе потока для обработки. Это позволяет нам сделать наши компоненты конвейера модульными и многоразовыми, и мы можем использовать один и тот же компонент несколько раз в разных конфигурациях. Точно так же мы будем получать трафик от любого производителя, который пишет в Kafka, поэтому мы можем объединять производителей в один поток данных.

## Пишем сервер GraphQL на Go

GraphQL — это альтернатива REST, созданная Facebook (<http://graphql.org/>). Эта технология позволяет серверу реализовать и опубликовать схему, после чего клиенты могут запрашивать необходимую им информацию, вместо того чтобы понимать и использовать различные конечные точки API.

Для этого рецепта мы создадим схему `GraphQL`, представляющую колоду игральных карт. Мы выложим одну карту ресурса, которую можно отфильтровать по масти и достоинству. Кроме того, эта схема может возвращать все карты в колоде, если аргументы не указаны.

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter12/graphql` и перейдите в этот каталог.
- Запустите эту команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter12/graphql
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter12/graphql
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter12/graphql` или используйте это как

упражнение для написания собственного кода!

- Создайте и перейдите в каталог `cards`.
- Создайте файл с именем `card.go` со следующим содержимым:

```
package cards

// Card represents a standard playing
// card
type Card struct {
    Value string
    Suit string
}

var cards []Card

func init() {
    cards = []Card{
        {"A", "Spades"}, {"2", "Spades"}, {"3",
"Spades"},
        {"4", "Spades"}, {"5", "Spades"}, {"6",
"Spades"},
        {"7", "Spades"}, {"8", "Spades"}, {"9",
"Spades"},
        {"10", "Spades"}, {"J", "Spades"}, {"Q",
"Spades"},
        {"K", "Spades"},
        {"A", "Hearts"}, {"2", "Hearts"}, {"3",
"Hearts"},
        {"4", "Hearts"}, {"5", "Hearts"}, {"6",
"Hearts"},
        {"7", "Hearts"}, {"8", "Hearts"}, {"9",
"Hearts"},
        {"10", "Hearts"}, {"J", "Hearts"}, {"Q",
"Hearts"},
        {"K", "Hearts"},
        {"A", "Clubs"}, {"2", "Clubs"}, {"3",
"Clubs"},
        {"4", "Clubs"}, {"5", "Clubs"}, {"6",
"Clubs"},
        {"7", "Clubs"}, {"8", "Clubs"}, {"9",
"Clubs"},
        {"10", "Clubs"}, {"J", "Clubs"}, {"Q",
```



```

"Clubs"},
    {"K", "Clubs"},
    {"A", "Diamonds"}, {"2", "Diamonds"},
{"3",
    "Diamonds"},
    {"4", "Diamonds"}, {"5", "Diamonds"},
{"6",
    "Diamonds"},
    {"7", "Diamonds"}, {"8", "Diamonds"},
{"9",
    "Diamonds"},
    {"10", "Diamonds"}, {"J", "Diamonds"},
{"Q",
    "Diamonds"},
    {"K", "Diamonds"},
}
}

```

- Создайте файл с именем `type.go` со следующим содержимым:

```
package cards

import "github.com/graphql-go/graphql"

// CardType returns our card graphql object
func CardType() *graphql.Object {
    cardType :=
graphql.NewObject(graphql.ObjectConfig{
        Name: "Card",
        Description: "A Playing Card",
        Fields: graphql.Fields{
            "value": &graphql.Field{
                Type: graphql.String,
                Description: "Ace through King",
                Resolve: func(p
graphql.ResolveParams)
                    (interface{}, error) {
                        if card, ok := p.Source.
(Card); ok {
                                return card.Value, nil
                            }
                            return nil, nil
                        },
```

```

    },
    "suit": &graphql.Field{
        Type: graphql.String,
        Description: "Hearts, Diamonds,
Clubs, Spades",
        Resolve: func(p
graphql.ResolveParams)
(interface{}, error) {
    if card, ok := p.Source.
(Card); ok {
        return card.Suit, nil
    }
    return nil, nil
},
    },
    },
})
return cardType
}

```

- Создайте файл с именем `resolve.go` со следующим содержимым:

```

package cards

import (
    "strings"

    "github.com/graphql-go/graphql"
)

// Resolve handles filtering cards
// by suit and value
func Resolve(p graphql.ResolveParams)
(interface{}, error) {
    finalCards := []Card{}
    suit, suitOK := p.Args["suit"].(string)
    suit = strings.ToLower(suit)

    value, valueOK := p.Args["value"].(string)
    value = strings.ToLower(value)

    for _, card := range cards {
        if suitOK && suit !=

```

```

strings.ToLower(card.Suit) {
    continue
}
if valueOK && value !=
strings.ToLower(card.Value) {
    continue
}

    finalCards = append(finalCards, card)
}
return finalCards, nil
}

```

- Создайте файл с именем `schema.go` со следующим содержимым:

```

package cards

import "github.com/graphql-go/graphql"

// Setup prepares and returns our card
// schema
func Setup() (graphql.Schema, error) {
    cardType := CardType()

    // Schema
    fields := graphql.Fields{
        "cards": &graphql.Field{
            Type: graphql.NewList(cardType),
            Args: graphql.FieldConfigArgument{
                "suit": &graphql.ArgumentConfig{
                    Description: "Filter cards by
card suit
spades)",
                    Type: graphql.String,
                },
                "value": &graphql.ArgumentConfig{
                    Description: "Filter cards by
card
value (A-K)",
                    Type: graphql.String,
                },
            },
        },
    },
}

```

```

        Resolve: Resolve,
    },
}

    rootQuery := graphql.ObjectConfig{Name:
"RootQuery",
    Fields: fields}
    schemaConfig := graphql.SchemaConfig{Query:
graphql.NewObject(rootQuery)}
    schema, err := graphql.NewSchema(schemaConfig)

    return schema, err
}

```

- Вернитесь в каталог `graphql`.
- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "encoding/json"
    "fmt"
    "log"

    "github.com/PacktPublishing/
Go-Programming-Cookbook-Second-Edition/
chapter12/graphql/cards"
    "github.com/graphql-go/graphql"
)

func main() {
    // grab our schema
    schema, err := cards.Setup()
    if err != nil {
        panic(err)
    }

    // Query
    query := `
{
    cards(value: "A"){
        value
    }
}
`
}

```

```

        suit
    }
}

params := graphql.Params{Schema: schema,
RequestString:
    query}
r := graphql.Do(params)
if len(r.Errors) > 0 {
    log.Fatalf("failed to execute graphql
operation,
        errors: %+v", r.Errors)
}
rJSON, err := json.MarshalIndent(r, "", " ")
if err != nil {
    panic(err)
}
fmt.Printf("%s \n", rJSON)
}

```

- Выполните `go run main.go`.
- Вы также можете запустить следующую команду:

```

$ go build
$ ./example

```

Вы должны увидеть следующий вывод:

```

$ go run main.go
{
  "data": {
    "cards": [
      {
        "suit": "Spades",
        "value": "A"
      },
      {
        "suit": "Hearts",
        "value": "A"
      },
      {
        "suit": "Clubs",
        "value": "A"
      },
    ]
  }
}

```

```
{
  "suit": "Diamonds",
  "value": "A"
}
]
```

- Протестируйте некоторые дополнительные запросы, например следующие:
  - `cards(suit: "Spades")`
  - `cards(value: "3", suit:"Diamonds")`
- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Файл `cards.go` определяет объект `card` и инициализирует базовую колоду в глобальной переменной с именем `cards`. Это состояние также может храниться в долговременном хранилище, например в базе данных. Затем мы определяем `CardType` в `types.go`, что позволяет `graphql` преобразовывать объекты карты в ответы. Затем мы переходим к `resolve.go`, где определяем, как фильтровать карты по значению и типу. Эта функция `Resolve` будет использоваться окончательной схемой, которая определена в `schema.go`.

Например, вы должны изменить функцию `Resolve` в этом рецепте, чтобы получить данные из базы данных. Наконец, мы загружаем схему и запускаем к ней запрос. Это небольшая модификация для подключения нашей схемы к конечной точке REST, но для краткости этот рецепт просто запускает жестко заданный запрос. Для получения дополнительной информации о запросах `GraphQL` посетите <http://graphql.org/learn/queries/>.

# 13. Бессерверное программирование

В этой главе основное внимание будет уделено бессерверным архитектурам и их использованию с языком Go. В бессерверных архитектурах разработчик не управляет внутренним сервером. Сюда входят такие сервисы, как Amazon Lambda, Google App Engine и Firebase. Эти сервисы позволяют быстро разворачивать приложения и хранить данные в Интернете.

Все рецепты в этой главе касаются сторонних сервисов, выставляющих счета за использование; убедитесь, что вы убираете, когда закончите их использовать. В противном случае считайте эти рецепты стартовыми для запуска больших приложений на этих платформах.

В этой главе мы рассмотрим следующие рецепты:

- Go программирование на Lambda с Apex
- Бессерверное ведение журналов и метрик Apex
- Google App Engine с Go
- Работа с Firebase с помощью [firebase.google.com/go](https://firebase.google.com/go)

## Go программирование на Lambda с Apex

Apex — это инструмент для создания, разворачивания и управления функциями AWS Lambda. Раньше он предоставлял Go `shim` для управления функциями Lambda в коде, но теперь это делается с помощью собственной библиотеки AWS (<https://github.com/aws/aws-lambda-go>). В этом рецепте рассматривается создание функций Go Lambda и их разворачивание с помощью Apex.

### Подготовка

Настройте среду в соответствии с этими шагами:

- Загрузите и установите Go 1.12.6 или более позднюю версию в своей операционной системе с <https://golang.org/doc/install>.
- Установите Apex с <http://apex.run/#installation>.
- Откройте терминал или консольное приложение, создайте и перейдите в каталог проекта, например `~/projects/go-programming-cookbook`. Весь код, который мы рассмотрим в этом рецепте, будет запускаться и модифицироваться из этого каталога.
- Скопируйте последний код в `~/projects/go-programming-cookbook-original`. Здесь у вас есть возможность работать из этого каталога, а не вводить примеры вручную:

```
$ git clone git@github.com:PacktPublishing/Go-Programming-Cookbook-Second-Edition.git go-programming-cookbook-original
```

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter13/lambda` и перейдите в него.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter13/lambda
```

Вы должны увидеть файл с именем `go.mod`, содержащий следующее содержимое:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter13/lambda
```

- Создайте учетную запись Amazon и роль IAM, которая может редактировать функции Lambda, что можно сделать на странице <https://aws.amazon.com/lambda/>.
- Создайте файл с именем `~/.aws/credentials` со следующим содержимым, скопировав свои учетные данные из того, что вы настроили в консоли Amazon:

```
[default]
aws_access_key_id = xxxxxxxx
```



```
aws_secret_access_key = xxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

- Создайте переменную среды для хранения нужного региона:

```
export AWS_REGION=us-west-2
```

- Запустите команду `apex init` и следуйте инструкциям на экране:

```
$ apex init
```

```
Enter the name of your project. It should be machine-
friendly, as this is used to prefix your functions in
Lambda.
```

```
Project name: go-cookbook
```

```
Enter an optional description of your project.
```

```
Project description: Demonstrating Apex with the Go
Cookbook
```

```
[+] creating IAM go-cookbook_lambda_function role
[+] creating IAM go-cookbook_lambda_logs policy
[+] attaching policy to lambda_function role.
[+] creating ./project.json
[+] creating ./functions
```

```
Setup complete, deploy those functions!
```

```
$ apex deploy
```

- Удалите каталог `lambda/functions/hello`.
- Создайте новый файл `lambda/functions/greeter1/main.go` со следующим содержимым:

```
package main
```

```
import (
    "context"
    "fmt"

    "github.com/aws/aws-lambda-go/lambda"
)
```

```
// Message is the input to the function and
// includes a Name
type Message struct {
    Name string `json:"name"`
}

// Response is sent back and contains a greeting
// string
type Response struct {
    Greeting string `json:"greeting"`
}

// HandleRequest will be called when the lambda function
// is invoked
// it takes a Message and returns a Response that
// contains a greeting
func HandleRequest(ctx context.Context, m Message)
(Response, error) {
    return Response{Greeting: fmt.Sprintf("Hello, %s",
m.Name)}, nil
}

func main() {
    lambda.Start(HandleRequest)
}
```

- Создайте новый файл `lambda/functions/greeter/main.go` со следующим содержимым:

```
package main

import (
    "context"
    "fmt"

    "github.com/aws/aws-lambda-go/lambda"

    // Message is the input to the function and
    // includes a FirstName and LastName
    type Message struct {
        FirstName string `json:"first_name"`
        LastName string `json:"last_name"`
    }
}
```

```

}

// Response is sent back and contains a greeting
// string
type Response struct {
    Greeting string `json:"greeting"`
}

// HandleRequest will be called when the lambda function
// is invoked
// it takes a Message and returns a Response that
// contains a greeting
// this greeting contains the first and last name
// specified
func HandleRequest(ctx context.Context, m Message)
(Response, error) {
    return Response{Greeting: fmt.Sprintf("Hello, %s %s",
m.FirstName, m.LastName)}, nil
}

func main() {
    lambda.Start(HandleRequest)
}

```

- Разверните их:

```

$ apex deploy
• creating function env= function=greeter2
• creating function env= function=greeter1
• created alias current env= function=greeter2 version=4
• function created env= function=greeter2 name=go-
cookbook_greeter2 version=1
• created alias current env= function=greeter1 version=5
• function created env= function=greeter1 name=go-
cookbook_greeter1 version=1

```

- Вызовите недавно развернутые функции:

```

$ echo '{"name": "Reader"}' | apex invoke greeter1
{"greeting": "Hello, Reader"}

```

```

$ echo '{"first_name": "Go", "last_name": "Coders"}' |
apex invoke greeter2 {"greeting": "Hello, Go Coders"}

```

- Взгляните на журналы:

```
$ apex logs greeter2
apex logs greeter2
/aws/lambda/go-cookbook_greeter2 START RequestId:
7c0f9129-3830-11e7-8755-75aeb52a51b9 Version: 1
/aws/lambda/go-cookbook_greeter2 END RequestId: 7c0f9129-
3830-11e7-8755-75aeb52a51b9
/aws/lambda/go-cookbook_greeter2 REPORT RequestId:
7c0f9129-3830-11e7-8755-75aeb52a51b9 Duration: 93.84 ms
Billed Duration: 100 ms
Memory Size: 128 MB Max Memory Used: 19 MB
```

- Очистите развернутые службы:

```
$ apex delete
The following will be deleted:
```

- greeter1
- greeter2

```
Are you sure? (yes/no) yes
```

- deleting env= function=greeter
- function deleted env= function=greeter

## Как это работает...

AWS Lambda позволяет легко запускать функции по запросу без обслуживания сервера. Apex предоставляет средства для развертывания, управления версиями и тестирования функций по мере их отправки в Lambda.

Библиотека Go (<https://github.com/aws/aws-lambda-go>) обеспечивает нативную компиляцию Go в Lambda и позволяет нам развертывать код Go как функции Lambda. Это достигается путем определения обработчика, обработки полезной нагрузки входящего запроса и возврата ответа. В настоящее время функции, которые вы определяете, должны соответствовать следующим правилам:

- Обработчик должен быть функцией.
- Обработчик может принимать от нуля до двух аргументов.
- При наличии двух аргументов первый аргумент должен удовлетворять интерфейсу `context.Context`.

- Обработчик может вернуть от нуля до двух аргументов.
- Если есть два возвращаемых значения, второй аргумент должен быть ошибкой.
- Если есть одно возвращаемое значение, это должна быть ошибка.

В этом рецепте мы определили две функции приветствия: одну, которая принимает полное имя, и другую, в которой мы разделяем имя на имя и фамилию. Если бы мы изменили одну функцию (`greeter`) вместо создания двух, Арех вместо этого развернул бы новую версию и вызвал бы ее в `v2`, а не в `v1` во всех предыдущих примерах. Также можно было бы откатиться и с `apex rollback greeter`.

## Бессерверное ведение журналов и метрик Арех

При работе с бессерверными функциями, такими как Lambda, полезно иметь переносимые структурированные журналы. Кроме того, вы можете комбинировать более ранние рецепты, связанные с ведением журнала, с этим рецептом. Рецепты, которые мы рассмотрели в [Главе 4 «Обработка ошибок в Go»](#), не менее актуальны. Поскольку мы используем Арех для управления нашими функциями Lambda, мы решили использовать регистратор Арех для этого рецепта. Мы также будем полагаться на метрики, предоставляемые Арех, а также на консоль AWS. В более ранних рецептах рассматривались более сложные примеры ведения журналов и метрик, и они все еще применимы — регистратор Арех можно легко настроить для агрегирования журналов с помощью, например, Amazon Kinesis или Elasticsearch.

### Подготовка

Обратитесь к разделу «Подготовка» рецепта *«Go программирование на Lambda с Арех»* в этой главе.

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter13/logging` и перейдите к нему.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter13/logging
```

Вы должны увидеть файл с именем `go.mod`, содержащий следующее содержимое:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter13/logging
```

- Создайте учетную запись Amazon и роль IAM, которая может редактировать функции Lambda, что можно сделать по адресу <http://aws.amazon.com/lambda/>.
- Создайте файл `~/.aws/credentials` со следующим содержимым, скопировав свои учетные данные из того, что вы настроили в консоли Amazon:

```
[default]
aws_access_key_id = xxxxxxxx
aws_secret_access_key = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

- Создайте переменную среды для хранения нужного региона:

```
export AWS_REGION=us-west-2
```

- Запустите команду `apex init` и следуйте инструкциям на экране:

```
$ apex init
```

```
Enter the name of your project. It should be machine-friendly, as this is used to prefix your functions in Lambda.
```

```
Project name: logging
```

```
Enter an optional description of your project.
```

```
Project description: An example of apex logging and metrics
```

```
[+] creating IAM logging_lambda_function role
[+] creating IAM logging_lambda_logs policy
[+] attaching policy to lambda_function role.
[+] creating ./project.json
[+] creating ./functions
```

**Setup complete, deploy those functions!**

### **\$ apex deploy**

- Удалите каталог `lambda/functions/hello`.
- Создайте новый файл `lambda/functions/secret/main.go` со следующим содержимым:

```
package main

import (
    "context"
    "os"

    "github.com/apex/log"
    "github.com/apex/log/handlers/text"
    "github.com/aws/aws-lambda-go/lambda"
)

// Input takes in a secret
type Input struct {
    Secret string `json:"secret"`
}

// HandleRequest will be called when the Lambda function
// is invoked
// it takes an input and checks if it matches our super
// secret value
func HandleRequest(ctx context.Context, input Input)
(string, error) {
    log.SetHandler(text.New(os.Stderr))

    log.WithField("secret", input.Secret).Info("secret
    guessed")

    if input.Secret == "klaatu barada nikto" {
```

```

        return "secret guessed!", nil
    }
    return "try again", nil
}

func main() {
    lambda.Start(HandleRequest)
}

```

- Разверните его в указанном вами регионе:

```

$ apex deploy
• creating function env= function=secret
• created alias current env= function=secret version=1
• function created env= function=secret
name=logging_secret version=1

```

- Чтобы вызвать его, выполните следующую команду:

```

$ echo '{"secret": "open sesame"}' | apex invoke secret
"try again"

```

```

$ echo '{"secret": "klaatu barada nikto"}' | apex invoke
secret
"secret guessed!"

```

- Проверьте журналы:

```

$ apex logs secret
/aws/lambda/logging_secret START RequestId: cfa6f655-
3834-11e7-b99d-89998a7f39dd Version: 1
/aws/lambda/logging_secret INFO[0000] secret guessed
secret=open sesame
/aws/lambda/logging_secret END RequestId: cfa6f655-3834-
11e7-b99d-89998a7f39dd
/aws/lambda/logging_secret REPORT RequestId: cfa6f655-
3834-11e7-b99d-89998a7f39dd Duration: 52.23 ms Billed
Duration: 100 ms Memory Size: 128 MB Max Memory Used: 19
MB
/aws/lambda/logging_secret START RequestId: d74ea688-
3834-11e7-aa4e-d592c1fbc35f Version: 1
/aws/lambda/logging_secret INFO[0012] secret guessed
secret=klaatu barada nikto
/aws/lambda/logging_secret END RequestId: d74ea688-3834-

```



```
11e7-aa4e-d592c1fbc35f
/aws/lambda/logging_secret REPORT RequestId: d74ea688-
3834-11e7-aa4e-d592c1fbc35f Duration: 7.43 ms Billed
Duration: 100 ms
Memory Size: 128 MB Max Memory Used: 19 MB
```

- Проверьте свои показатели:

```
$ apex metrics secret
```

```
secret
total cost: $0.00
invocations: 0 ($0.00)
duration: 0s ($0.00)
throttles: 0
errors: 0
memory: 128
```

- Очистите развернутые службы:

```
$ apex delete
Are you sure? (yes/no) yes
• deleting env= function=secret
• function deleted env= function=secret
```

## Как это работает...

В этом рецепте мы создали новую лямбда-функцию под названием `secret`, которая будет отвечать, угадали ли вы секретную фразу или нет. Функция анализирует входящий запрос JSON, выполняет некоторую регистрацию с помощью `Stderr` и возвращает ответ.

После использования этой функции несколько раз мы видим, что наши журналы видны с помощью команды `apex logs`. Эту команду можно запустить для одной функции Lambda или для всех наших управляемых функций. Это особенно полезно, если вы объединяете команды Apex в цепочку и хотите просматривать журналы многих служб.

Кроме того, мы показали, как использовать команду `apex metrics` для сбора общих показателей вашего приложения, включая стоимость и вызовы. Вы также можете увидеть много этой информации

непосредственно в консоли AWS в разделе Lambda. Как и в других рецептах, в конце мы попытались убрать за собой.

## Google App Engine с Go

App Engine – это служба Google, которая упрощает развертывание веб-приложений. Эти приложения имеют доступ к облачному хранилищу и различным другим API Google. Общая идея заключается в том, что App Engine будет легко масштабироваться с нагрузкой и упростит любое управление операциями, связанными с размещением приложения. В этом рецепте показано, как создать и при необходимости развернуть базовое приложение App Engine. Этот рецепт не будет вдаваться в подробности настройки учетной записи Google Cloud, настройки выставления счетов или особенностей очистки вашего экземпляра. Как минимум, для работы этого рецепта требуется доступ к Google Cloud Datastore (<https://cloud.google.com/datastore/docs/concepts/overview>).

### Подготовка

Настройте среду в соответствии с этими шагами:

- Загрузите и установите Go 1.11.1 или более позднюю версию в своей операционной системе с <https://golang.org/doc/install>.
- Загрузите Google Cloud SDK со страницы <https://cloud.google.com/appengine/docs/flexible/go/quickstart>.
- Создайте приложение, позволяющее выполнять доступ к хранилищу данных, и запишите имя приложения. Для этого рецепта мы будем использовать [go-cookbook](#).
- Установите компонент движка приложения Go `gcloud components install app-engine-go`.
- Откройте терминал или консольное приложение, создайте и перейдите в каталог проекта, например `~/projects/go-programming-cookbook`. Весь код, который мы рассмотрим в этом рецепте, будет запускаться и модифицироваться из этого каталога.
- Скопируйте последний код в `~/projects/go-programming-cookbook-original`. Здесь у вас есть возможность работать из этого каталога, а не вводить примеры вручную:

```
$ git clone git@github.com:PacktPublishing/Go-Programming-Cookbook-Second-Edition.git go-programming-cookbook-original
```

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter13/appengine` и перейдите в него.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter13/appengine
```

Вы должны увидеть файл с именем `go.mod`, содержащий следующее содержимое:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter13/appengine
```

- Создайте файл с именем `app.yml` со следующим содержимым, заменив `go-cookbook` на имя приложения, которое вы создали в разделе «Подготовка»:

```
runtime: go112

manual_scaling:
  instances: 1

#[START env_variables]
env_variables:
  GCLOUD_DATASET_ID: go-cookbook
#[END env_variables]
```

- Создайте файл с именем `message.go` со следующим содержимым:

```
package main

import (
    "context"
    "time"
```

```

        "cloud.google.com/go/datastore"
    )

    // Message is the object we store
    type Message struct {
        Timestamp time.Time
        Message string
    }

    func (c *Controller) storeMessage(ctx
context.Context, message
string) error {
        m := &&Message{
            Timestamp: time.Now(),
            Message: message,
        }

        k := datastore.IncompleteKey("Message", nil)
        _, err := c.store.Put(ctx, k, m)
        return err
    }

    func (c *Controller) queryMessages(ctx
context.Context, limit
int) ([]*Message, error) {
        q := datastore.NewQuery("Message").
            Order("-Timestamp").
            Limit(limit)

        messages := make([]*Message, 0)
        _, err := c.store.GetAll(ctx, q,
&&messages)
        return messages, err
    }

```

- Создайте файл с именем `controller.go` со следующим содержимым:

```

package main

import (
    "context"
    "fmt"

```

```

        "log"
        "net/http"

        "cloud.google.com/go/datastore"
    )

    // Controller holds our storage and other
    // state
    type Controller struct {
        store *datastore.Client
    }

    func (c *Controller) handle(w http.ResponseWriter,
r
        *http.Request) {
        if r.Method != http.MethodGet {
            http.Error(w, "invalid method",
                http.StatusMethodNotAllowed)
            return
        }

        ctx := context.Background()

        // store the new message
        r.ParseForm()
        if message := r.FormValue("message"); message
!= "" {
            if err := c.storeMessage(ctx, message);
err != nil {
                log.Printf("could not store message:
%v", err)

                http.Error(w, "could not store
message",
                    http.StatusInternalServerError)
                return
            }
        }

        // get the current messages and display them
        fmt.Fprintln(w, "Messages:")
        messages, err := c.queryMessages(ctx, 10)
        if err != nil {
            log.Printf("could not get messages: %v",

```

```

err)
    http.Error(w, "could not get messages",
        http.StatusInternalServerError)
    return
}

for _, message := range messages {
    fmt.Fprintln(w, message.Message)
}
}

```

- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "log"
    "net/http"
    "os"

    "cloud.google.com/go/datastore"
    "golang.org/x/net/context"
    "google.golang.org/appengine"
)

func main() {
    ctx := context.Background()
    log.SetOutput(os.Stderr)

    // Set this in app.yaml when running in
production.
    projectID := os.Getenv("GLOUD_DATASET_ID")

    datastoreClient, err :=
datastore.NewClient(ctx, projectID)
    if err != nil {
        log.Fatal(err)
    }

    c := Controller{datastoreClient}

    http.HandleFunc("/", c.handle)
}

```

```

port := os.Getenv("PORT")
if port == "" {
    port = "8080"
    log.Printf("Defaulting to port %s", port)
}

log.Printf("Listening on port %s", port)

log.Fatal(http.ListenAndServe(fmt.Sprintf(":%s", port),
nil))
}

```

- Запустите команду `gcloud config set project go-cookbook`, где `go-cookbook` — это проект, который вы создали в разделе «Подготовка».
- Запустите команду `cloud auth application-default login` и следуйте инструкциям.
- Запустите команду `export PORT=8080`.
- Запустите команду `export GCLOUD_DATASET_ID=go-cookbook`, где `go-cookbook` — это проект, который вы создали в разделе «Подготовка».
- Запустите команду `go build`.
- Запустите команду `./appengine`.
- Перейдите по адресу `http://localhost:8080/?message=hello%20there`.
- Попробуйте отправить еще несколько сообщений (`?message=other`).
- При необходимости разверните приложение в своем экземпляре с помощью `gcloud app deploy`.
- NПерейдите к развернутому приложению с помощью `gcloud app browse`.
- При желании очистите свой экземпляр `appengine` и хранилище данных по следующим URL-адресам:
  - <https://console.cloud.google.com/datastore>
  - <https://console.cloud.google.com/appengine>
- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, запустите команду `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Как только облачный SDK настроен так, чтобы указывать на ваше приложение, и прошел проверку подлинности, инструмент GCloud позволяет быстро развертывать и настраивать, позволяя локальным приложениям получать доступ к службам Google.

После аутентификации и установки порта мы запускаем приложение на `localhost` и можем приступать к работе с кодом. Приложение определяет объект сообщения, который можно сохранить и извлечь из хранилища данных. Это демонстрирует, как вы можете изолировать такой код. Вы также можете использовать интерфейс хранилища/базы данных, как показано в предыдущих главах.

Затем мы настраиваем обработчик, который пытается вставить сообщение в хранилище данных, а затем извлекает все сообщения, отображая их в браузере. Это создает что-то похожее на простую гостевую книгу. Вы можете заметить, что сообщение не всегда появляется сразу. Если вы перемещаетесь без параметра сообщения или отправляете другое сообщение, оно должно появиться при перезагрузке.

Наконец, убедитесь, что вы очистили экземпляры, если вы их больше не используете.

## Работа с Firebase с помощью [firebase.google.com/go](https://firebase.google.com/go)

Firebase — еще один сервис Google Cloud, который создает масштабируемую, простую в управлении базу данных, которая может поддерживать аутентификацию и особенно хорошо работает с мобильными приложениями. В этом рецепте мы будем использовать последнюю версию Firestore в качестве базы данных. Служба Firebase предоставляет значительно больше, чем то, что будет описано в этом рецепте, но мы просто рассмотрим хранение и извлечение данных. Мы также рассмотрим, как настроить аутентификацию для вашего приложения и обернуть клиент Firebase нашим собственным настраиваемым клиентом.



## Подготовка

Настройте среду в соответствии с этими шагами:

- Загрузите и установите Go 1.11.1 или более позднюю версию в своей операционной системе с <https://golang.org/doc/install..>
- Создайте учетную запись Firebase, проект и базу данных на странице <https://console.firebase.google.com/>.



*Этот рецепт работает в тестовом режиме, который по умолчанию не является безопасным.*

- Создайте токен администратора службы, перейдя по адресу <https://console.firebase.google.com/project/go-cookbook/settings/serviceaccounts/adminsdk>. Здесь `go-cookbook` заменяется названием вашего проекта.
- Переместите загруженный токен в `/tmp/service_account.json`.
- Откройте терминал или консольное приложение, создайте и перейдите в каталог проекта, например `~/projects/go-programming-cookbook`. Весь код, который мы рассмотрим в этом рецепте, будет запускаться и модифицироваться из этого каталога.
- Скопируйте последний код в `~/projects/go-programming-cookbook-original`. Здесь у вас есть возможность работать из этого каталога, а не вводить примеры вручную:

```
$ git clone git@github.com:PacktPublishing/Go-Programming-Cookbook-Second-Edition.git go-programming-cookbook-original
```

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter13/firebase` и перейдите в него.
- Выполните следующую команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter13/firebase
```

Вы должны увидеть файл с именем `go.mod`, содержащий следующее содержимое:

```
module github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter13/firebase
```

- Создайте файл `client.go` со следующим содержимым:

```
package firebase

import (
    "context"

    "cloud.google.com/go/firestore"
    "github.com/pkg/errors"
)

// Client Interface for mocking
type Client interface {
    Get(ctx context.Context, key string) (interface{},
    error)
    Set(ctx context.Context, key string, value interface{})
    error
    Close() error
}

// firestore.Client implements Close()
// we create Get and Set
type firebaseClient struct {
    *firestore.Client
    collection string
}

func (f *firebaseClient) Get(ctx context.Context, key
string) (interface{}, error) {
    data, err :=
f.Collection(f.collection).Doc(key).Get(ctx)
    if err != nil {
        return nil, errors.Wrap(err, "get failed")
    }
    return data.Data(), nil
}
```

```
}
```

```
func (f *firebaseClient) Set(ctx context.Context, key
string, value interface{}) error {
    set := make(map[string]interface{})
    set[key] = value
    _, err := f.Collection(f.collection).Doc(key).Set(ctx,
set)
    return errors.Wrap(err, "set failed")
}
```

- Создайте файл с именем **auth.go** со следующим содержимым:

```
package firebase

import (
    "context"

    firebase "firebase.google.com/go"
    "github.com/pkg/errors"
    "google.golang.org/api/option"
)

// Authenticate grabs oauth scopes using a generated
// service_account.json file from
// https://console.firebase.google.com/project/go-
// cookbook/settings/serviceaccounts/adminsdk
func Authenticate(ctx context.Context, collection string)
(Client, error) {

    opt :=
option.WithCredentialsFile("/tmp/service_account.json")
    app, err := firebase.NewApp(ctx, nil, opt)
    if err != nil {
        return nil, errors.Wrap(err, "error initializing
app")
    }

    client, err := app.Firestore(ctx)
    if err != nil {
        return nil, errors.Wrap(err, "failed to intialize
filestore")
    }
}
```

```
    return &&firebaseClient{Client: client,
collection: collection}, nil
}
```

- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```
package main
```

```
import (
    "context"
    "fmt"
    "log"
```

```
    "github.com/PacktPublishing/Go-Programming-Cookbook-
Second-Edition/chapter13/firebase"
)
```

```
func main() {
    ctx := context.Background()
    c, err := firebase.Authenticate(ctx, "collection")
    if err != nil {
        log.Fatalf("error initializing client: %v", err)
    }
    defer c.Close()

    if err := c.Set(ctx, "key", []string{"val1", "val2"});
err != nil {
        log.Fatalf(err.Error())
    }

    res, err := c.Get(ctx, "key")
    if err != nil {
        log.Fatalf(err.Error())
    }
    fmt.Println(res)

    if err := c.Set(ctx, "key2", []string{"val3", "val4"});
err != nil {
        log.Fatalf(err.Error())
    }

    res, err = c.Get(ctx, "key2")
}
```

```

    if err != nil {
        log.Fatalf(err.Error())
    }
    fmt.Println(res)
}

```

- Выполните `go run main.go`.
- Вы также можете запустить `go build ./example`. Вы должны увидеть следующий вывод:

```

$ go run main.go
[val1 val2]
[val3 val4]

```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Firebase предоставляет удобные функции, позволяющие войти в систему с помощью файла учетных данных. После того, как мы вошли в систему, мы можем сохранить любой структурированный объект, похожий на карту. В данном случае мы храним `map[string]interface{}`. Эти данные доступны ряду клиентов, в том числе в Интернете и через мобильные устройства.

Клиентский код включает все операции в интерфейс для простоты тестирования. Это распространенный шаблон при написании клиентского кода, который также используется в других рецептах. В нашем случае мы создаем функцию `Get` and `Set`, которая сохраняет и извлекает значение по ключу. Мы также выставляем `Close()`, чтобы код, использующий клиента, мог отложить `close()` и очистить наше соединение в конце.

# 14. Улучшения производительности, советы и рекомендации

В этой главе мы сосредоточимся на оптимизации приложения и обнаружении узких мест. Вот несколько советов и приемов, которые можно сразу же использовать в существующих приложениях. Многие из этих рецептов необходимы, если вам или вашей организации требуются полностью воспроизводимые сборки. Они также полезны, когда вы хотите оценить производительность приложения. Последний рецепт фокусируется на увеличении скорости HTTP; однако всегда важно помнить, что мир Интернета движется быстро, и важно освежать в памяти передовой опыт. Например, если вам требуется HTTP/2, он доступен с помощью встроенного пакета Go [net/http](#), начиная с версии 1.6.

В этой главе мы рассмотрим следующие рецепты:

- Использование инструмента `pprof`
- Бенчмаркинг и поиск узких мест
- Распределение памяти и управление кучей
- Использование `fasthttprouter` и `fasthttp`

## Технические требования

Чтобы продолжить выполнение всех рецептов в этой главе, настройте свою среду в соответствии со следующими шагами:

- Загрузите и установите Go 1.12.6 или более позднюю версию в своей операционной системе с <https://golang.org/doc/install>.
- Откройте терминал или консольное приложение, создайте и перейдите в каталог проекта, например `~/projects/go-programming-cookbook`. Весь код, который мы рассмотрим в этом рецепте, будет запускаться и модифицироваться из этого каталога.
- Скопируйте последний код в `~/projects/go-programming-cookbook-original`. Здесь у вас есть возможность работать из

этого каталога, а не вводить примеры вручную:

```
$ git clone git@github.com:PacktPublishing/Go-Programming-Cookbook-Second-Edition.git go-programming-cookbook-original
```

- При желании установите Graphviz с <http://www.graphviz.org/Home.php>.

## Использование инструмента pprof

Инструмент `pprof` позволяет приложениям Go собирать и экспортировать данные профилирования во время выполнения. Он также предоставляет веб-перехватчики для доступа к инструменту из веб-интерфейса. Этот рецепт создаст базовое приложение, которое сверяет пароль, хешированный с помощью `bcrypt`, с паролем в виде открытого текста, а затем профилирует приложение.

Возможно, вы ожидали, что инструмент `pprof` будет описан в [Главе 11 «Распределенные системы»](#) с другими показателями и рецептами мониторинга. Вместо этого он был помещен в эту главу, потому что он будет использоваться для анализа и улучшения программы почти так же, как может использоваться бенчмаркинг. Поэтому в этом рецепте основное внимание будет уделено `pprof` для анализа и улучшения использования памяти приложением.

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter14/pprof` и перейдите в этот каталог.
- Запустите эту команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter14/pprof
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

[module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter14/pprof](https://github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter14/pprof)

- Скопируйте тесты из [~/projects/go-programming-cookbook-original/chapter14/pprof](https://github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter14/pprof) или используйте это как упражнение для написания собственного кода!
- Создайте каталог с именем `crypto` и перейдите к нему.
- Создайте файл с именем `handler.go` со следующим содержимым:

```
package crypto

import (
    "net/http"

    "golang.org/x/crypto/bcrypt"
)

// GuessHandler checks if ?message=password
func GuessHandler(w http.ResponseWriter, r
*http.Request) {
    if err := r.ParseForm(); err != nil{
        // if we can't parse the form
        // we'll assume it is malformed
        w.WriteHeader(http.StatusBadRequest)
        w.Write([]byte("error reading guess"))
        return
    }

    msg := r.FormValue("message")

    // "password"
    real :=

    []byte("$2a$10$2ovnPWuIjMx2S0HvCxP/mutzdsGhyt8rq/
    JqnJg/60yC3B0APMGlK")

    if err :=
    bcrypt.CompareHashAndPassword(real, []byte(msg));
    err != nil {
        w.WriteHeader(http.StatusBadRequest)
        w.Write([]byte("try again"))
        return
    }
}
```



```

        w.WriteHeader(http.StatusOK)
        w.Write([]byte("you got it"))
        return
    }

```

- Перейдите вверх по каталогу.
- Создайте новый каталог с именем `example` и перейдите к нему.
- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "fmt"
    "log"
    "net/http"
    _ "net/http/pprof"

    "github.com/PacktPublishing/
    Go-Programming-Cookbook-Second-Edition/
    chapter14/pprof/crypto"
)

func main() {

    http.HandleFunc("/guess", crypto.GuessHandler)
    fmt.Println("server started at
localhost:8080")

    log.Panic(http.ListenAndServe("localhost:8080", nil))
}

```

- Выполните `go run main.go`.
- Вы также можете запустить следующую команду:

```

$ go build
$ ./example

```

Вы также можете запустить следующие команды:

```

$ go run main.go
server started at localhost:8080

```

- В отдельном терминале выполните следующее:

```
$ go tool pprof http://localhost:8080/debug/pprof/profile
```

- Это запустит 30-секундный таймер.
- Запустите несколько команд `curl` во время работы `pprof`:

```
$ curl "http://localhost:8080/guess?message=test"
try again
```

```
$ curl "http://localhost:8080/guess?message=password"
you got it
```

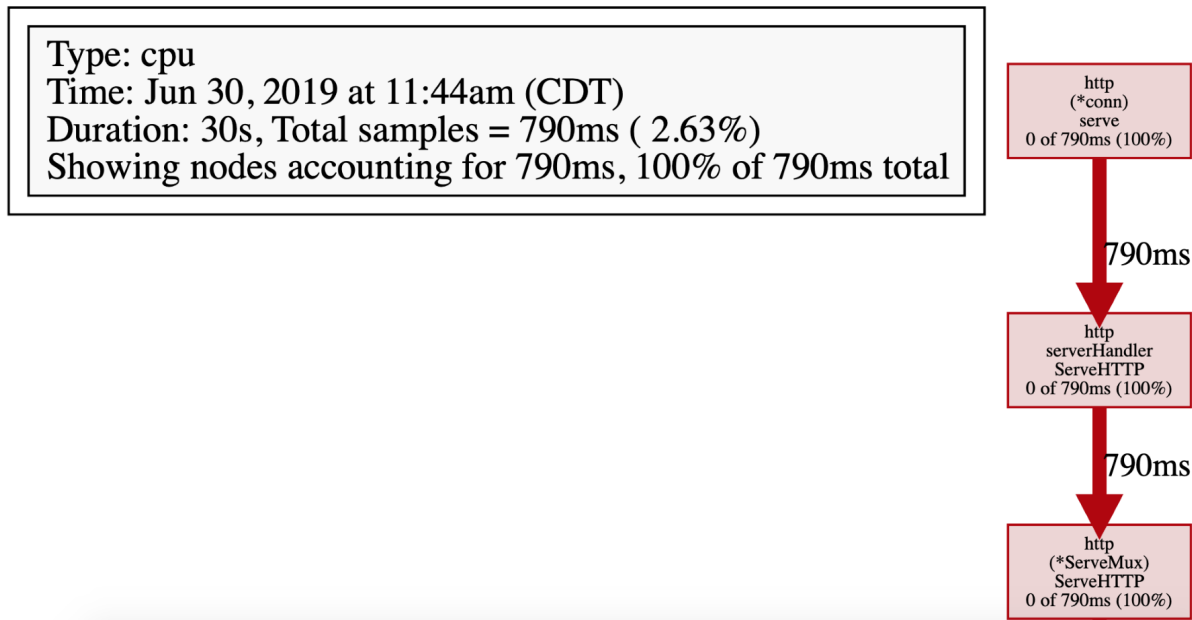
```
.
.
.
.
```

```
$ curl "http://localhost:8080/guess?message=password"
you got it
```

- Вернитесь к команде `pprof` и дождитесь ее завершения.
- Запустите команду `top10` из командной строки `pprof`:

```
(pprof) top 10
930ms of 930ms total ( 100%)
Showing top 10 nodes out of 15 (cum >= 930ms)
flat flat% sum% cum cum%
870ms 93.55% 93.55% 870ms 93.55%
golang.org/x/crypto/blowfish.encryptBlock
30ms 3.23% 96.77% 900ms 96.77%
golang.org/x/crypto/blowfish.ExpandKey
30ms 3.23% 100% 30ms 3.23% runtime.memclrNoHeapPointers
0 0% 100% 930ms 100% github.com/agtorre/go-
cookbook/chapter13/pprof/crypto.GuessHandler
0 0% 100% 930ms 100%
golang.org/x/crypto/bcrypt.CompareHashAndPassword
0 0% 100% 30ms 3.23%
golang.org/x/crypto/bcrypt.base64Encode
0 0% 100% 930ms 100% go
golang.org/x/crypto/bcrypt.bcrypt
0 0% 100% 900ms 96.77%
golang.org/x/crypto/bcrypt.expensiveBlowfishSetup
0 0% 100% 930ms 100% net/http.(*ServeMux).ServeHTTP
0 0% 100% 930ms 100% net/http.(*conn).serve
```

- Если вы установили Graphviz или поддерживаемый браузер, запустите команду `web` из командной строки `pprof`. Вы должны увидеть что-то вроде этого с гораздо более длинной цепочкой красных прямоугольников с правой стороны:



- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Инструмент `pprof` предоставляет много информации о вашем приложении во время выполнения. Использование пакета `net/pprof` обычно наиболее просто для настройки — все, что требуется, — это прослушивание порта и выполнение импорта.

В нашем случае мы написали обработчик, использующий очень ресурсоемкое приложение (`bcrypt`), чтобы продемонстрировать, как они появляются при профилировании с помощью `pprof`. Это позволит быстро изолировать фрагменты кода, создающие узкие места в вашем приложении.

Мы решили собрать общий профиль, который заставляет `pprof` опрашивать конечную точку нашего приложения в течение 30 секунд.

Затем мы сгенерировали трафик для конечной точки, чтобы получить результаты. Это может быть полезно, когда вы пытаетесь проверить один обработчик или ветвь кода.

Наконец, мы рассмотрели 10 самых популярных функций с точки зрения использования ЦП. Также можно просмотреть управление памятью/кучей с помощью команды `pprof` <http://localhost:8080/debug/pprof/heap>. Команду `web` в консоли `pprof` можно использовать для просмотра визуализации вашего профиля ЦП/памяти и помогает выделить более активный код.

## Бенчмаркинг и поиск узких мест

Еще один метод определения медленных частей кода — использование тестов. Тесты можно использовать для тестирования функций на среднюю производительность, а также можно запускать тесты параллельно. Это может быть полезно при сравнении функций или микрооптимизации определенного кода, особенно для того, чтобы увидеть, как может работать реализация функции при ее одновременном использовании. Для этого рецепта мы создадим две структуры, каждая из которых реализует атомарный счетчик. Первый будет использовать пакет `sync`, а другой — `sync/atomic`. Затем мы сравним оба решения.

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter14/bench` и перейдите в этот каталог.
- Запустите эту команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter14/bench
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter14/bench
```

- Скопируйте тесты из [~/projects/go-programming-cookbook-original/chapter14/bench](#) или используйте это как упражнение для написания собственного кода!



*Обратите внимание, что скопированные тесты также включают тесты, написанные позже в этом рецепте.*

- Создайте файл с именем `lock.go` со следующим содержимым:

```
package bench

import "sync"

// Counter uses a sync.RWMutex to safely
// modify a value
type Counter struct {
    value int64
    mu *sync.RWMutex
}

// Add increments the counter
func (c *Counter) Add(amount int64) {
    c.mu.Lock()
    c.value += amount
    c.mu.Unlock()
}

// Read returns the current counter amount
func (c *Counter) Read() int64 {
    c.mu.RLock()
    defer c.mu.RUnlock()
    return c.value
}
```

- Создайте файл с именем `atomic.go` со следующим содержимым:

```
package bench

import "sync/atomic"

// AtomicCounter implements an atomic lock
```

```
// using the atomic package
type AtomicCounter struct {
    value int64
}

// Add increments the counter
func (c *AtomicCounter) Add(amount int64) {
    atomic.AddInt64(&c.value, amount)
}

// Read returns the current counter amount
func (c *AtomicCounter) Read() int64 {
    var result int64
    result = atomic.LoadInt64(&c.value)
    return result
}
```

- Создайте файл с именем `lock_test.go` со следующим содержимым:

```
package bench

import "testing"

func BenchmarkCounterAdd(b *testing.B) {
    c := Counter{0, &sync.RWMutex{}}
    for n := 0; n < b.N; n++ {
        c.Add(1)
    }
}

func BenchmarkCounterRead(b *testing.B) {
    c := Counter{0, &sync.RWMutex{}}
    for n := 0; n < b.N; n++ {
        c.Read()
    }
}

func BenchmarkCounterAddRead(b *testing.B) {
    c := Counter{0, &sync.RWMutex{}}
    b.RunParallel(func(pb *testing.PB) {
        for pb.Next() {
            c.Add(1)
        }
    })
}
```

```

        c.Read()
    }
    })
}

```

- Создайте файл с именем `atomic_test.go` со следующим содержимым:

```

package bench

import "testing"

func BenchmarkAtomicCounterAdd(b *testing.B) {
    c := AtomicCounter{0}
    for n := 0; n < b.N; n++ {
        c.Add(1)
    }
}

func BenchmarkAtomicCounterRead(b *testing.B) {
    c := AtomicCounter{0}
    for n := 0; n < b.N; n++ {
        c.Read()
    }
}

func BenchmarkAtomicCounterAddRead(b *testing.B) {
    c := AtomicCounter{0}
    b.RunParallel(func(pb *testing.PB) {
        for pb.Next() {
            c.Add(1)
            c.Read()
        }
    })
}

```

- Запустите `go test -bench .` команду, и вы увидите следующий вывод:

```

$ go test -bench .
BenchmarkAtomicCounterAdd-4 2000000000 8.38 ns/op
BenchmarkAtomicCounterRead-4 1000000000 2.09 ns/op
BenchmarkAtomicCounterAddRead-4 500000000 24.5 ns/op
BenchmarkCounterAdd-4 500000000 34.8 ns/op

```

```
BenchmarkCounterRead-4 20000000 66.0 ns/op
BenchmarkCounterAddRead-4 10000000 146 ns/op
PASS
ok github.com/PacktPublishing/Go-Programming-Cookbook-Second-
Edition/chapter14/bench 10.919s
```

- Если вы скопировали или написали свои собственные тесты, перейдите на один каталог вверх и запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Этот рецепт является примером сравнения критического пути кода. Например, иногда ваше приложение должно выполнять определенные функции часто, может быть, каждый вызов. В этом случае мы написали атомарный счетчик, который может добавлять или считывать значения из нескольких горутин.

Первое решение использует объекты `RWMutex` и `Lock` или `RLock` для записи и чтения соответственно. Второй использует пакет `atomic`, предоставляющий тот же функционал из коробки. Мы делаем сигнатуры наших функций одинаковыми, поэтому эталонные тесты можно использовать повторно с небольшими изменениями и чтобы они удовлетворяли одному и тому же целочисленному интерфейсу `atomic`.

Наконец, мы пишем стандартные тесты для добавления значений и их чтения. Затем мы пишем параллельный тест, который вызывает функции добавления и чтения. Параллельный бенчмарк создаст множество конфликтов за блокировку, поэтому мы ожидаем замедления. Возможно неожиданно, пакет `atomic` значительно превосходит `RWMutex`.

## Распределение памяти и управление кучей

Некоторые приложения могут значительно выиграть от оптимизации. Возьмем, к примеру, маршрутизаторы, которые мы рассмотрим в следующем рецепте. К счастью, набор инструментов для тестирования



предоставляет флаги для сбора количества выделений памяти, а также размера выделенной памяти. Может быть полезно настроить определенные критические пути кода, чтобы свести к минимуму эти два атрибута.

Этот рецепт покажет два подхода к написанию функции, которая склеивает строки с помощью пробела, подобно `strings.Join("a", "b", "c")`. В одном подходе будет использоваться конкатенация, а в другом — пакет строк. Затем мы сравним производительность и распределение памяти между ними.

## Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В своем терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter14/tuning` и перейдите в этот каталог.
- Запустите эту команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter14/tuning
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-
Cookbook-Second-Edition/chapter14/tuning
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter14/tuning` или используйте это как упражнение для написания собственного кода!



*Обратите внимание, что скопированные тесты также включают тесты, написанные позже в этом рецепте.*

- Создайте файл с именем `concat.go` со следующим содержимым:

```
package tuning

func concat(vals ...string) string {
```

```

        finalVal := ""
        for i := 0; i < len(vals); i++ {
            finalVal += vals[i]
            if i != len(vals)-1 {
                finalVal += " "
            }
        }
        return finalVal
    }
}

```

- Создайте файл с именем `join.go` со следующим содержимым:

```

package tuning

import "strings"

func join(vals ...string) string {
    c := strings.Join(vals, " ")
    return c
}

```

- Создайте файл с именем `concat_test.go` со следующим содержимым:

```

package tuning

import "testing"

func Benchmark_concat(b *testing.B) {
    b.Run("one", func(b *testing.B) {
        one := []string{"1"}
        for i := 0; i < b.N; i++ {
            concat(one...)
        }
    })
    b.Run("five", func(b *testing.B) {
        five := []string{"1", "2", "3", "4", "5"}
        for i := 0; i < b.N; i++ {
            concat(five...)
        }
    })
    b.Run("ten", func(b *testing.B) {
        ten := []string{"1", "2", "3", "4", "5",

```

```

        "6", "7", "8", "9", "10"}
    for i := 0; i < b.N; i++ {
        concat(ten...)
    }
})
}

```

- Создайте файл с именем `join_test.go` со следующим содержимым:

```

package tuning

import "testing"

func Benchmark_join(b *testing.B) {
    b.Run("one", func(b *testing.B) {
        one := []string{"1"}
        for i := 0; i < b.N; i++ {
            join(one...)
        }
    })
    b.Run("five", func(b *testing.B) {
        five := []string{"1", "2", "3", "4", "5"}
        for i := 0; i < b.N; i++ {
            join(five...)
        }
    })
    b.Run("ten", func(b *testing.B) {
        ten := []string{"1", "2", "3", "4", "5",
            "6", "7", "8", "9", "10"}
        for i := 0; i < b.N; i++ {
            join(ten...)
        }
    })
}

```

- Запустите `GOMAXPROCS=1 go test -bench=. -benchmem -benchtime=1s`, и вы увидите следующий вывод:

```

$ GOMAXPROCS=1 go test -bench=. -benchmem -benchtime=1s
Benchmark_concat/one 100000000 13.6 ns/op 0 B/op 0
allocs/op
Benchmark_concat/five 5000000 386 ns/op 48 B/op 8

```

```

allocs/op
Benchmark_concat/ten 2000000 992 ns/op 256 B/op 18
allocs/op
Benchmark_join/one 200000000 6.30 ns/op 0 B/op 0
allocs/op
Benchmark_join/five 10000000 124 ns/op 32 B/op 2
allocs/op
Benchmark_join/ten 10000000 183 ns/op 64 B/op 2 allocs/op
PASS
ok github.com/PacktPublishing/Go-Programming-Cookbook-
Second-
Edition/chapter14/tuning 12.003s

```

- Если вы скопировали или написали свои собственные тесты, запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Бенчмаркинг помогает нам настраивать приложения и выполнять определенные микрооптимизации таких вещей, как распределение памяти. При сравнительном анализе выделений для приложений с входными данными важно попробовать разные размеры входных данных, чтобы определить, влияет ли это на выделения. Мы написали две функции, `concat` и `join`. Оба объединяют `variadic` строковый параметр с пробелами, поэтому аргументы (*a*, *b*, *c*) возвращают строку *a b c*.

Подход `concat` реализует это исключительно за счет конкатенации строк. Мы создаем строку и добавляем строки в список и пробелы в цикле `for`. Мы опускаем добавление пробела в последнем цикле. Функция `join` использует внутреннюю функцию `Strings.Join` для более эффективного выполнения этой задачи в большинстве случаев. Может быть полезно сравнить стандартную библиотеку с вашими собственными функциями, чтобы лучше понять компромиссы между производительностью, простотой и функциональностью.

Мы использовали вспомогательные тесты для проверки всех наших параметров, которые также отлично работают с табличными тестами. Мы можем видеть, как подход `concat` приводит к гораздо большему распределению, чем `join`, по крайней мере, для входных данных

одинарной длины. Хорошим упражнением было бы попробовать это со входными строками переменной длины, а также с рядом аргументов.

## Использование `fasthttp` и `fasthttp`

Хотя стандартная библиотека Go предоставляет все необходимое для запуска HTTP-сервера, иногда вам необходимо дополнительно оптимизировать такие вещи, как маршрутизация и время запроса. В этом рецепте будет рассмотрена библиотека, ускоряющая обработку запросов, называемая `fasthttp` (<https://github.com/valyala/fasthttp>), и маршрутизатор, который значительно повышает производительность маршрутизации, называемый `fasthttp` (<https://github.com/buaazp/fasthttprouter>). Хотя `fasthttp` работает быстро, важно отметить, что он не поддерживает HTTP/2 (<https://github.com/valyala/fasthttp/issues/45>).

### Как это сделать...

Эти шаги охватывают написание и запуск вашего приложения:

- В терминале или консольном приложении создайте новый каталог с именем `~/projects/go-programming-cookbook/chapter14/fastweb` и перейдите в этот каталог.
- Запустите эту команду:

```
$ go mod init github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter14/fastweb
```

Вы должны увидеть файл с именем `go.mod`, который содержит следующее:

```
module github.com/PacktPublishing/Go-Programming-Cookbook-Second-Edition/chapter14/fastweb
```

- Скопируйте тесты из `~/projects/go-programming-cookbook-original/chapter14/fastweb` или используйте это как упражнение для написания собственного кода!
- Создайте файл с именем `items.go` со следующим содержимым:

```

package main

import (
    "sync"
)

var items []string
var mu *sync.RWMutex

func init() {
    mu = &sync.RWMutex{}
}

// AddItem adds an item to our list
// in a thread-safe way
func AddItem(item string) {
    mu.Lock()
    items = append(items, item)
    mu.Unlock()
}

// ReadItems returns our list of items
// in a thread-safe way
func ReadItems() []string {
    mu.RLock()
    defer mu.RUnlock()
    return items
}

```

- Создайте файл с именем `handlers.go` со следующим содержимым:

```

package main

import (
    "encoding/json"
    "github.com/valyala/fasthttp"
)

// GetItems will return our items object
func GetItems(ctx *fasthttp.RequestCtx) {
    enc := json.NewEncoder(ctx)

```

```

        items := ReadItems()
        enc.Encode(&items)
        ctx.SetStatusCode(fasthttp.StatusOK)
    }

    // AddItems modifies our array
    func AddItems(ctx *fasthttp.RequestCtx) {
        item, ok := ctx.UserValue("item").(string)
        if !ok {

ctx.SetStatusCode(fasthttp.StatusBadRequest)
            return
        }

        AddItem(item)
        ctx.SetStatusCode(fasthttp.StatusOK)
    }

```

- Создайте файл с именем `main.go` со следующим содержимым:

```

package main

import (
    "fmt"
    "log"

    "github.com/buaazp/fasthttprouter"
    "github.com/valyala/fasthttp"
)

func main() {
    router := fasthttprouter.New()
    router.GET("/item", GetItems)
    router.POST("/item/:item", AddItems)

    fmt.Println("server starting on
localhost:8080")

    log.Fatal(fasthttp.ListenAndServe("localhost:8080",
        router.Handler))
}

```

- Запустите команду `go build`.
- Запустите команду `./fastweb`.

```
$ ./fastweb
server starting on localhost:8080
```

- В отдельном терминале протестируйте его с помощью некоторых команд `curl`:

```
$ curl "http://localhost:8080/item/hi" -X POST
```

```
$ curl "http://localhost:8080/item/how" -X POST
```

```
$ curl "http://localhost:8080/item/are" -X POST
```

```
$ curl "http://localhost:8080/item/you" -X POST
```

```
$ curl "http://localhost:8080/item" -X GET
["hi", "how", "are", "you"]
```

- Файл `go.mod` может быть обновлен, и теперь файл `go.sum` должен присутствовать в каталоге рецептов верхнего уровня.
- Если вы скопировали или написали свои собственные тесты, запустите `go test`. Убедитесь, что все тесты пройдены.

## Как это работает...

Пакеты `fasthttp` и `fasthttprouter` могут многое сделать для ускорения жизненного цикла веб-запроса. Оба пакета в значительной степени оптимизируют «горячий путь» кода, но с досадной оговоркой, заключающейся в переписывании ваших обработчиков для использования нового объекта контекста, а не традиционного средства записи запросов и ответов.

Существует ряд фреймворков, использующих аналогичный подход к маршрутизации, а некоторые из них напрямую включают `fasthttp`. Эти проекты хранят актуальную информацию в своих файлах `README`.

В нашем рецепте реализован простой объект `list`, к которому мы можем добавить одну конечную точку и который будет возвращен другой. Основная цель этого рецепта — продемонстрировать работу с параметрами, настроить маршрутизатор, который теперь явно определяет поддерживаемые методы вместо общих `Handle` и `HandleFunc`, и показать, насколько они похожи на стандартные обработчики, но имеют множество других преимуществ.