STRUČNI UVID

GO

OD POČETNIKA DO PROFESIONALCA

Kreirajte Golang aplikacije za produkciju, koristeći mrežne biblioteke, istovremene tokove, mašinsko učenje i napredne strukture podataka.

II izdanje

Mihalis Tsoukalos

kompjuter biblioteka

Packt›

GO OD POČETNIKA DO PROFESIONALCA

Kreirajte Golang aplikacije za produkciju,
koristeći mrežne biblioteke, istovremene tokove,
mašinsko učenje i napredne strukture podataka.

II izdanje

Skenirajte QR kod,
registrujte knjigu
i osvojite nagradu

Često (pogrešno) nazivan Golang, Go je sistemski jezik visokih performansi, jezik budućnosti. „Go od početnika do profesionalca (drugo izdanje)“ vam pomaže da postanete produktivni stručnjak u Go programiranju, gradeći i unaređujući revolucionarno prvo izdanje.

„Go od početnika do profesionalca (drugo izdanje)“ pokazuje kako se pokreće Go da radi na stvarnim produkcijskim sistemima. Programerima koji već poznaju osnove Go jezika ova knjiga nudi primere, obrasce i jasna objašnjenja koja će im pomoći da duboko razumeju mogućnosti Goa i da ih primene u svom programerskom radu.

Knjiga pokriva nijanse Goa, sa detaljnim upitstvima o tipovima i strukturama, paketima, istovremenom toku, mrežnom programiranju, dizajnu kompajlera, optimizaciji i još mnogo čemu. Svako poglavlje se završava vežbama i resursima da biste upotpunili svoje novo znanje.

Ovo drugo izdanje sadrži potpuno novo poglavlje o mašinskom učenju u Gou, koje vas vodi od osnovnih statističkih tehnika, kroz jednostavnu regresiju i klasterisanje, do klasifikacije, neuronskih mreža i detekcije anomalija. Ostala poglavlja su proširena da bi se „pokrila“ upotreba Goa sa Dokerom i Kuberneteom, Gitom, WebAssemblyjem, JSON-om i još mnogo čime.

Ako ozbiljno shvatite programski jezik Go, drugo izdanje ove knjige biće vam neophodan priručnik za ekspertske tehnike.

Naučićete sledeće:

- jasne smernice za upotrebu Go za proizvodne sisteme
- detaljna objašnjenja kako funkcionišu Go komponente, odabir dizajna koji stoji iza jezika i način kako se optimizuje Go kod
- potpuni vodič za sve tipove Go podataka, kompozitne tipove i strukture podataka
- master pakete, refleksiju i interfejse za efikasno Go programiranje
- gradnju mrežnog koda za sisteme visokih performansi, uključujući servere i aplikacije na strani klijenta
- interfejs sa drugim sistemima koji koriste WebAssembly, JSON i gRPC
- pisanje pouzdanog koda, istovremenog toka i visokih performansi
- gradnju u Gou sisteme za mašinsko učenje, od jednostavne statističke regresije, do složenih neuronskih mreža

ISBN: 978-86-7310-554-3

GO

OD POČETNIKA DO PROFESIONALCA

Mihalis Tsoukalos

kompjuter biblioteka

Packt›

Izdavač:

kompjuter biblioteka

Obalskih radnika 4a, Beograd

Tel: 011/2520272

e-mail: kombib@gmail.com

internet: www.kombib.rs

Urednik: Mihailo J. Šolajić

Za izdavača, direktor:
Mihailo J. Šolajić

Autor: Mihalis Tsoukalos

Prevod: Đorđe Badža

Lektura: Miloš Jevtović

Slog: Zvonko Aleksić

Znak Kompjuter biblioteke:
Miloš Milosavljević

Štampa: „Pekograf“, Zemun

Tiraž: 500

Godina izdanja: 2020.

Broj knjige: 531

Izdanje: Prvo

ISBN: 978-86-7310-554-3

Mastering Go

Second Edition

Mihalis Tsoukalos

ISBN 978-1-83855-933-5

O AUTORU

Mihalis Tsoukalos je UNIX administrator, programer, DBA i matematičar, koji uživa u pisanju tehničkih knjiga i članaka. Autor je knjiga Go Systems Programming i Mastering Go. Napisao je više od 250 tehničkih članaka za mnoge časopise, uključujući Sys Admin, MacTech, Linux User and Developer, Usenix; login, Linux format i Linux Journal. Njegova istraživačka interesovanja uključuju baze podataka, vizuelizaciju, statistiku i mašinsko učenje.

Mihalis je takođe fotograf.

Možete da kontaktirate sa njim pomoću adrese https://www.mtsoukalos.eu/ i @mactsouk.

Zahvaljujem se ljudima iz izdavačke kuće „Packt“ koji su mi pomogli da napišem ovu knjigu, uključujući mog tehničkog recenzenta Mat Rajer i Kišor Rit, za odgovore na sva moja pitanja i ohrabrivanje tokom čitavog procesa pripreme knjige.

Ovu knjigu posvećujem sećanju na moje drage roditelje Joanisa i Argete.

O RECENZENTU

Mat Rajer programira računare od svoje šeste godine. Izrađivao je igre i programe sa svojim ocem, najpre u BASIC-u na ZX Spectrumu, a potom na AmigaBASIC i AMOS na Komodoru Amiga. Mnogo sati je potrošio na ručno kopiranje koda iz časopisa „Amiga Format“ i ugađanje varijabli ili pomeranje GOTO instrukcije okolo da se vidi šta bi se moglo dogoditi. Isti duh istraživanja i opsesije programiranjem vodio je Mata da započne posao sa lokalnom agencijom u Mansfieldu, u Engleskoj, kada je imao 18 godina, gde je počeo da izrađuje veb stranice i druge internetske servise.

Nakon nekoliko godina korišćenja raznih tehnologija u raznim industrijama u Londonu i širom sveta, Mat je primetio novi sistemski jezik pod nazivom Go, koji je nastao u “Googleu”. Budući da se bavio veoma značajnim i relevantnim modernim tehničkim izazovima, počeo je da koristi Go za rešavanje problema još dok je taj jezik bio u beta-fazi i koristi ga još uvek. Mat doprinosi projektima otvorenog koda i osnovao je Go pakete, uključujući Testifaj, Mok, Silk i Is, kao i mekOS alat za programere, nazvan BitBar.

Mat je 2018. godine postao suosnivač Mašin Boksa, a još uvek provodi mnogo vremena govoreći na konferencijama, pišući o Gou na svom blogu i aktivan je član Go zajednice.

Predgovor

Knjiga *Go od početnika do profesionalca (drugo izdanje)* treba da vam pomogne da postanete bolji Go programer!

U ovom izdanju mnogo je novih uzbudljivih tema, uključujući potpuno novo poglavlje o mašinskom učenju u Gou, kao i informacija i primera koda koji se odnose na Vajper i Kobra Go pakete, gRPC, rad uz korišćenje Doker slika, YAML datoteka, `go/scanner` i `go/token` paketa i generisanje WebAssembly koda od Go koda. Ukupno, u ovom drugom izdanju ima više od 130 novih stranica.

ZA KOGA JE OVA KNJIGA

Ova knjiga je namenjena Go programerima sa početnim i srednjim znanjem koji žele podići svoje poznavanje Goa na sledeći nivo, kao i iskusnim programerima u drugim programskim jezicima koji žele da nauče Go bez ponovnog učenja kako radi `for` petlja.

Neke informacije koje se nalaze u ovoj knjizi se mogu naći i u mojoj drugoj knjizi *Go Systems Programming*. Glavna razlika između ove dve knjige je u tome što je *Go Systems Programming* posvećena razvoju sistemskih alata koji koriste mogućnosti Goa, dok je u knjizi *Go od početnika do profesionalca* u pitanju objašnjavanje mogućnosti i Go internala da biste postali bolji programer za Go. Obe knjige se mogu koristiti kao referenca nakon što ih pročitate prvi ili drugi put.

ŠTA OVA KNJIGA POKRIVA

Poglavlje 1, *Go i operativni sistem*, započinje pričom o istoriji Goa i njegovim prednostima pre nego što počnu opisivanje korisničke alatke `godoc` i objašnjenja kako možete kompajlirati i izvršavati Go programe. Nakon toga, govori se o

ispisu rezultata i dobijanju korisnikovog unosa, upotrebi argumenata komandne linije programa i korišćenju datoteka dnevnika. Završna tema u prvom poglavlju je rukovanje greškama, koje u Gou igra ključnu ulogu.

U Poglavlju 2, *Razumevanje Go komponenata*, upoznaćete sakupljač smeća u Gou i način na koji on operiše. Takođe će biti reči o nesigurnom kodu i `unsafe` paketu, kao i načinima kako pozvati program napisan u C kodu iz Goa i kako pozvati Go kod iz C programa.

Nakon toga, prikazana je upotreba ključne reči `defer` i predstavljeni su alati `strace(1)` i `dtrace(1).` U ostalim odeljcima ovog poglavlja naučićete kako da pronađete informacije o okruženju vašeg Goa, kako se upotrebljava Go asembler i kako se iz Go koda generiše WebAssembly kod.

Poglavlje 3, *Korišćenje osnovnih tipova Go podataka*, posvećeno je tipovima podataka koje nudi Go, što uključuje nizove, kriške i mape, kao i Go pokazivače, konstante, petlje i korišćenje datuma i vremena.

Poglavlje 4, *Korišćenje složenih tipova*, počinje učenjem o Go strukturama i ključnoj reči `struct`, pre nego što budemo predstavili slogove, nizove, rune, kriške bajtova i literale stringova. U ostatku poglavlja se govori o regularnim izrazima i podudaranju uzoraka, instrukciji `switch`, paketu `strings`, paketu `math/big`, razvoju spremišta ključ-vrednost u Gou i korišćenju XML i JSON datoteka.

Poglavlje 5, *Kako poboljšati Go kod pomoću struktura podataka*, odnosi se na razvoj vlastitih struktura podataka kada strukture koje nudi Go ne odgovaraju određenom problemu. Ovo uključuje razvoj binarnih stabala, povezane liste, hash tabele, stekove i redove čekanja i učenje o njihovim prednostima. U ovom poglavlju takođe je prikazana upotreba struktura koje se mogu pronaći u standardnom Go paketu `container` i način kako se Go koristi za verifikaciju Sudoku zagonetki i kako se generišu slučajni brojevi.

Poglavlje 6, *Ono što možda ne znate o Go paketima i funkcijama*, u celini je posvećeno paketima i funkcijama, upotrebi funkcije `init()`, standardnom Go paketu `syscall` i paketima `text/template` i `html/template`. Pored toga, prikazana je upotreba naprednih paketa `go/scanner`, `go/parser` i `go/token`. Ovo poglavlje će vas definitivno učiniti boljim Go developerom!

U Poglavlju 7, *Refleksije i interfejsi za sve sezone*, upoznaćete tri napredna Go koncepta: refleksije, interfejse i metode tipa. Pored toga, biće reči o objektno-orijentisanim mogućnostima Goa i uklanjanju grešaka u Go programima pomoću Delve.

Poglavlje 8, *Naložiti UNIX sistemu šta da radi*, odnosi se na programiranje sistema u Gou, koji obuhvata teme poput flag paketa za rad uz korišćenje argumenata komandne linije, rukovanje UNIX signalima, ulaznim i izlaznim datotekama, paket bytes, interfejse io.Reader i io.Writer i upotrebu Viper i Kobra Go paketa.

Ako stvarno radite na programiranju sistema u Gou, toplo preporučujem da, nakon knjige *Mastering Go (drugo izdanje)*, pročitate *Go Systems Programming*.

U Poglavlju 9, *Istovremeni tokovi u Gou - goroutine, kanali i cevovodi*, razmatramo o goroutinama, kanalima i cevovodimai. Takođe ćete učiti o razlikama između procesa, niti i gorutina, sync paketima i načinu kako radi Go planer.

Poglavlje 10, *Istovremeni tokovi u Gou - napredne teme*, započeće od tačke gde je završeno prethodno poglavlje i učiniće vas majstorom goroutina i kanala! Naučićete više o Go planeru, upotrebi moćne ključne reči `select` i raznim tipovima Go kanala, kao i o deljenoj memoriji, muteksima, `sync.Mutex` tipu i `sync.RWMutex` tipu. Završni deo poglavlja posvećen je `context` paketu, udruživanju izvršilaca i načinu kako se detektuju uslovi trke.

U Poglavlju 11, *Testiranje koda, optimizacija i profilisanje*, biće reči o testiranju koda, optimizaciji i profilisanju koda, kao i unakrsnom kompajliranju, kreiranju dokumentacije, benčmarkingu Go koda, kreiranju primera funkcija i pronalaženju nedostupnog Go koda.

U Poglavlju 12, *Osnove mrežnog programiranja u Gou*, predstavićemo net/http paket i saznaćete kako možete da razvijate veb klijente i veb servere u Gou. Ovaj razvoj takođe uključuje upotrebu http.Response, http.Request i http.Transport struktura i http.NewServeMux tip. Čak ćete naučiti kako da u Gou razvijete čitavu veb stranicu! Nadalje, u ovom poglavlju ćete naučiti kako da čitate konfiguraciju vašeg mrežnog interfejsa, kako se izvodi DNS pretraga u Gou i kako se koristi gRPC sa Goom.

Poglavlje 13, *Mrežno programiranje - izrada vlastitih servera i klijenata*, posvećeno je aktivnostima u vezi sa HTTPS saobraćajem i kreiranju UDP i TCP servera i klijenata u Gou pomoću funkcionalnosti koju nudi net paket. Ostale teme sadržane u ovom poglavlju su kreiranje RPC klijenata i servera, razvoj TCP servera istovremenog toka u Gou i čitanje „sirovih" mrežnih paketa.

Poglavlje 14, *Mašinsko učenje u Go*, posvećeno je mašinskom učenju u Gou, uključujući klasifikaciju, klasterisanje, otkrivanje anomalija, izuzetke, neuronske mreže i TensorFlow, kao i korišćenju Apači Kafke pomoću Goa.

Ovu knjigu smo podelili u tri logička dela. Prvi deo sadrži sofisticirani pogled na neke važne koncepte Goa, uključujući unos korisnika i izlaz, preuzimanje spoljnih Go paketa, kompajliranje Go koda, pozivanje C koda iz Goa i kreiranje WebAssembly iz Goa, kao i upotrebu Go osnovnih i kompozitnih Go tipova.

Drugi deo čine Poglavlje 5, *Kako poboljšati Go kod pomoću struktura podataka*, Poglavlje 6, *Ono što možda ne znate o Go paketima i funkcijama*, i Poglavlje 7, *Refleksije i interfejsi za sve sezone*. U ova tri poglavlja su predstavljeni organizacija Go koda u pakete i module, dizajn Go projekata i neke Go napredne karakteristike, respektivno.

U preostalih sedam poglavlja, koja čine poslednji deo knjige, bavimo se praktičnijim Go temama. Poglavlja 8, 9, 10 i 11 posvećena su programiranju sistema u Gou, istovremenimm tokovima u Gou, testiranju koda, optimizaciji i profilisanju. U poslednja tri poglavlja biće reči o mrežnom programiranju i mašinskom učenju u Gou.

Knjiga uključuje sadržaje kao što su Go i WebAssembly, upotrebu Dokera sa Goom, kreiranje profesionalnih alata za naredbenu liniju sa paketima Viper i Kobra, koji parsiraju JSON i YAML slogove, izvođenje operacija s matricama, korišćenje Sudoku zagonetki, go/scanner i go/token, upotrebu git(1) i GitHuboma, atomic paket, gRPC i Go i HTTPS.

Knjiga sadrži relativno male, ali kompletne Go programe koji ilustruju predstavljene koncepte. To ima dve glavne prednosti: prvo, ne morate gledati beskrajne listinge koda kada pokušavate da naučite određenu tehniku i, drugo, taj kod možete koristiti kao polazište prilikom kreiranja vlastitih aplikacija i uslužnih programa.

Zbog važnosti **kontejnera** i **Dokera**, ova knjiga sadrži razne primere izvršnih Go datoteka koje se koriste iz Doker slike, jer te slike nude odličan način za razmeštanje serverskog softvera.

DA BISTE IZVUKLI MAKSIMUM IZ OVE KNJIGE...

Za čitanje ove knjige vam je potreban UNIX računar sa relativno skorašnjom instaliranom verzijom Goa, što uključuje bilo kakvu drugu mašinu koja radi na MacOS X, macOS ili Linuksu.Većina predstavljenog koda će raditi i na Microsoft Windows mašinama.

Da biste izvukli maksimum iz ove knjige, pokušajte što pre da znanje koje steknete u svakom poglavlju primenite u vlastitom programu. Kao što sam naglasio ranije, pokušajte da rešite vežbe koje možete pronaći na kraju svakog poglavlja ili kreirajte svoje probleme u programiranju.

Preuzmite primere datoteka sa kodom

Datoteke sa primerima koda za ovu knjigu možete preuzeti sa:

https://bit.ly/3527lIF

Nakon preuzimanja datoteke obavezno raspakujte ili ekstrahujte mapu pomoću najnovijih verzija:

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

Preuzmite slike u boji

Nudimo i PDF datoteku koja sadrži slike u boji/dijagrame u boji koji se koriste u ovoj knjizi. Možete je preuzeti na adresi https://static.packt-cdn.com/downloads/9781838559335.pdf.

Korišćene konvencije

U ovoj knjizi je korišćen niz tekstualnih konvencija.

`CodeInText`: Označava reči koda u tekstu, nazive tabela baze podataka, nazive direktorijuma, nazive datoteka, ekstenzije datoteka, nazive putanja, lažne URL--ove, korisnički unos i Twitter rukovaoce. Evo primera: „Prvi način je sličan korišćenju naredbe `man(1)`, ali za Go funkcije i pakete."

Blok koda je postavljen na sledeći način:

```
package main
import (
    "fmt"
)
func main() {
    fmt.Println("This is a sample Go program!")
}
```

Kada želimo da skrenemo vašu pažnju na određeni deo bloka koda, odgovarajuće linije ili stavke su postavljene podebljano:

```
import (
"fmt"
)
func main() {
    fmt.Println("This is a sample Go program!")
}
```

Svaki ulaz ili izlaz komandne linije se piše na sledeći način:

```
$ date
Sat Oct 21 20:09:20 EEST 2017
$ go version
go version go1.12.7 darwin/amd64
```

Bold: Označava novi termin, važnu reč ili reči koje vidite na ekranu - na primer, reči u menijima ili okvirima za dijalog koji se pojavljuju u ovom tekstu. Evo primera: „Izaberite Informacije o **sistemu** sa panela za **administraciju**."

Upozorenja ili važne napomene se pojavljuju ovako.

Saveti i trikovi se pojavljuju ovako.

OSTANIMO U KONTAKTU

Povratne informacije naših čitalaca su uvek dobrodošle.

Generalne povratne informacije: Ako imate pitanja o bilo kojem aspektu ove knjige, pošaljite nam e-mailom na customercare@packtpub.com.

Errata: Iako smo preduzeli sve da osiguramo tačnost našeg sadržaja, greške se dešavaju. Ako ste pronašli neku grešku u ovoj knjizi, bili bismo vam zahvalni ako nam to prijavite. Posetite adresu www.packt.com/submit-errata, odaberite knjigu, kliknite na link Obrazac za prijavu Errata i unesite detalje.

Piratstvo: Ako naiđete na bilo kakve ilegalne kopije naših dela u bilo kojem obliku na Internetu, mi bismo vam bili zahvalni ako biste nam dali adresu ili naziv veb lokacije. Molimo vas da u tom slučaju kontaktirate sa nama na adresi copyright@packt.com, sa linkom na materijal.

RECENZIJE

Nakon što ste pročitali i iskoristili ovu knjigu, zašto ne biste ostavili recenziju na veb stranici na kojoj ste je kupili? Potencijalni čitaoci mogu tada videti i koristiti vaše nepristrasno mišljenje da donesu odluku o kupovini, mi u „Packtu“ možemo razumeti što mislite o našem proizvodu, a naši autori mogu videti vaše povratne informacije o njihovoj knjizi. Hvala!

Postanite član Kompjuter biblioteke

Kupovinom jedne naše knjige stekli ste pravo da postanete član Kompjuter biblioteke. Kao član možete da kupujete knjige u pretplati sa 40% popusta i učestvujete u akcijama kada ostvarujete popuste na sva naša izdanja. Potrebno je samo da se prijavite preko formulara na našem sajtu.
Link za prijavu: http://bit.ly/2TxekSa

Skenirajte QR kod
registrujte knjigu
i osvojite nagradu

1

Go i operativni sistem

Ovo poglavlje je uvod u raznovrsne Go teme koje će početnicima biti veoma korisne. Iskusniji Go programeri mogu koristiti ovo poglavlje kao kurs za „osvežavanje“ znanja o osnovama Goa. Najbolji način za razumevanje nečega je da se eksperimentiše time. U ovom slučaju eksperimentisanje znači da pišete Go kod na svoj način, praveći svoje greške i učeći iz njih! Samo ne dozvolite da vas obeshrabre poruke o greškama i greške.

U ovom poglavlju ćete saznati mnogo štošta o:

- istoriji i budućnosti programskog jezika Go
- prednostima Goa
- kompajliranju Go koda
- izvršavanju Go koda
- preuzimanju i korišćenju spoljnih Go paketa
- UNIX standardnom ulazu, izlazu i grešci
- ispisivanju podataka na ekran
- dobijanju korisničkog unosa
- ispisu podataka na standardnu grešku
- radu sa datotekama dnevnika (log)
- korišćenju Dokera za kompajliranje i izvršavanje izvorne Go datoteke
- rukovanju greškama u Gou

Istorija Goa

Go je moderan, programski jezik opšte namene, otvorenog koda, koji je službeno najavljen krajem 2009. godine. Go je započeo kao interni Gugleov projekat, što znači da je započet kao eksperiment, a od tada je inspirisan mnogim drugim programiskim jezicima, uključujući **C**, **Paskal**, **Alef** i **Oberon**.

Duhovni očevi Goa su profesionalni programeri *Robert Griesemer*, *Ken Tomson* i *Rob Pajk*. Oni su dizajnirali Go kao jezik za profesionalne programere koji žele da grade pouzdan, robustan i efikasan softver. Osim sa svojom sintaksom i standardnim funkcijama, Go dolazi i sa prilično bogatom standardnom bibliotekom.

U vreme pisanja ove knjige najnovija stabilna Go verzija je 1.13. Međutim, čak i ako je broj vaše verzije veći, sadržaj knjige će i dalje biti relevantan.

Ako ćete prvi put da instalirate Go, možete početi tako što ćete posetiti adresu https://golang.org/dl/. Međutim, postoji velika verovatnoća da vaša UNIX varijanta ima spreman paket za instalaciju za programski jezik Go, tako da ćete možda hteti da postavite Go pomoću vašeg uobičajenog menadžera paketa.

Kuda ide Go?

Go zajednica već raspravlja o sledećoj osnovnoj verziji Goa, koja bi se zvala Go 2.

Namera trenutnog Go 1 tima je da učini da se Go 2 više vodi u zajednici. Iako je ovo uglavnom dobra ideja, uvek je opasno kada mnoštvo ljudi pokušava da donese važne odluke o programskom jeziku koji je u početku dizajnirala i razvila kao unutrašnji projekat mala grupa sjajnih stručnjaka.

Neke od velikih promena koje se razmatraju za Go 2 su generici, verzioniranje paketa i poboljšano rukovanje greškama. O ovim novim karakteristikama se trenutno vode rasprave i ne bi trebalo da brinete za njih, ali vredi znati u kojem smeru ide Go.

Prednosti Goa

Go ima mnogo prednosti, a neke od njih su jedinstvene za njega, dok druge deli sa drugim programskim jezicima.

Lista najznačajnijih Go prednosti i karakteristika uključuje sledeće:

- Go je moderan programski jezik koji je lak za čitanje i razumevanje, a napravili su ga iskusni programeri.
- Go želi srećne programere, jer oni pišu bolji kod!
- Kompajler Goa ispisuje praktična upozorenja i poruke o greškama, što pomaže da rešite stvarne probleme. Jednostavno rečeno, Go prevodilac je

tu da vam pomogne, a ne da vaš život učinite nesrećnim štampanjem besmislenog ispisa!

- Go kod je prenosiv, posebno između UNIX mašina.
- Go ima podršku za proceduralno programiranje, programiranje istvremenog toka i distribuirano programiranje.
- Go podržava **sakupljanje smeća**, tako da se ne morate baviti alokacijom i dealokacijom memorije.
- Go nema **pretprocesor** i vrši kompajliranje velikom brzinom. Kao posledica toga, Go se može koristiti i kao skriptni jezik.
- Go može graditi veb aplikacije, a obezbeđuje jednostavan veb server za svrhe testiranja.
- Standardna Go biblioteka nudi mnogo paketa koji programeru pojednostavljuju rad. Uz to su funkcije koje se nalaze u standardnoj Go biblioteci unapred testirane i greške su otklonili ljudi koji razvijaju Go, što znači da, uglavnom, funkcije dolaze bez grešaka.
- Go koristi **statičko povezivanje** prema zadatim postavkama, što znači da se proizvedene binarne datoteke mogu lako prebacivati na druge mašine sa istim OS-om. Kao posledica toga, ada se Go program uspešno kompajlira i kada se generiše izvršna datoteka, ne treba više da brinete o bibliotekama, zavisnostima i različitim verzijama biblioteke.
- Nije potreban **grafički korisnički interfejs** (**GUI**) za razvoj, uklanjanje pogrešaka i testiranje Go aplikacija, pošto se Go može koristiti iz komandne linije.
- Go podržava **Unicode**, što znači da vam nije potreban dodatni kod za ispis znakova iz više prirodnih jezika.
- Go zadržava ortogonalne koncepte, jer nekoliko ortogonalnih karakteristika bolje funkcioniše od više preklapajućih.

Da li je Go savršen?

Ne postoji savršen programski jezik, a Go nije izuzetak od ovog pravila. Međutim, neki programski jezici su bolji u nekim oblastima programiranja ili ih više volimo od drugih programskih jezika. Meni se, na primer, ne sviđa Java, a ne sviđa mi se ni C++, koji sam nekada voleo. C++ je postao previše složen kao programski jezik, dok, po mom mišljenju, Java kod ne izgleda dobro.

Neki od nedostataka Goa su sledeći:

- Go nema direktnu podršku za **objektno-orijentisano programiranje**, što može biti problem za programere koji su navikli da pišu kod na objektno-orijentisani način. Bez obzira na to, u Gou možete koristiti kompoziciju da biste oponašali nasleđivanje.
- Za neke ljude Go nikad neće zameniti C.
- C je i dalje brži od bilo kojeg drugog programskog jezika za programiranje sistema, uglavnom zato što je u njemu napisan UNIX.

Pa, ipak, Go je prilično pristojan programski jezik koji vas neće razočarati ako nađete vreme da ga naučite i da programirate u njemu.

Šta je pretprocesor?

Ranije sam rekao da Go nema **pretprocesor** i da je to dobro. Pretprocesor je program koji obrađuje ulazne podatke i generiše izlaz koji će biti iskorišćen kao ulaz u drugi program. U kontekstu programskih jezika ulaz pretprocesora je izvorni kod koji će pretprocesor obraditi pre nego što bude dat kao ulaz kompajleru programskog jezika.

Najveći nedostatak pretprocesora je što on ne „zna“ ništa o osnovama jezika ili o njegovoj sintaksi! To znači da, kada se koristi pretprocesor, ne možete biti sigurni da će konačna verzija vašeg koda učiniti ono što stvarno želite, jer pretprocesor može da izmeni logiku i semantiku vašeg originalnog koda.

Lista programskih jezika sa pretprocesorom obuhvata C, C++, Adu i PL/SQL. „Zloglasni“ C precesor obrađuje linije koje započinju sa `#` i nazivaju se **direktive** ili **pragme**. Kao što je prethodno rečeno, direktive i pragme nisu deo programskog jezika C!

Uslužni program godoc

Go distribucija dolazi sa mnoštvom alata koji vam mogu olakšati programerski život. Jedan od tih alata je program `godoc` koji omogućava da vidite dokumentaciju postojeće Go funkcije i paketa, bez potrebe za internetskom vezom.

Uslužni program `godoc` se može izvršiti ili kao uobičajena aplikacija komandne linije, koja prikazuje svoj izlaz na terminalu, ili kao aplikacija komandne linije koja pokreće veb server. U drugom slučaju će vam biti potreban veb pretraživač da biste pogledali dokumentaciju.

Ako upišete `godoc` bez ikakvih parametara komandne linije, dobićete listu opcija komandne linije koje `godoc` podržava.

Prvi način je sličan korišćenju naredbe `man(1)`, ali za Go funkcije i pakete. Dakle, da biste pronašli informacije o funkciji `Printf()` paketa `fmt`, treba da izvršite sledeću naredbu:

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

Slično tome, možete pronaći informacije o celom `fmt` paketu, pokrećući sledeću naredbu:

```
$ go doc fmt
```

Drugi način zahteva izvršavanje programa `godoc` sa parametrom `-http`:

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

Numerička vrednost u prethodnoj naredbi, koja je u ovom slučaju `8001`, predstavlja broj porta na kojem će HTTP „slušati". Možete da odaberete bilo koji broj porta koji je dostupan, pod uslovom da imate odgovarajuće privilegije. Međutim, imajte na umu da su brojevi portova 0-1023 ograničeni i da ih može koristiti samo root korisnik, tako da je bolje izbegavati izbor jednog od njih i odabrati neki drugi, pod uslovom da ga već ne koristi neki drugi proces.

U predstavljenoj naredbi možete izostaviti znak jednakosti i, umesto njega, staviti znak razmaka. Dakle, sledeća naredba potpuno je ekvivalentna prethodnoj:

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

Nakon toga bi trebalo da usmerite svoj veb pretraživač na URL http://localhost:8001/pkg/ da biste dobili listu dostupnih Go paketa i pregledali njihovu dokumentaciju.

KOMPAJLIRANJE GO KODA

U ovom odeljku ćete naučiti kako da kompajlirate Go kod. Dobra vest je da svoj Go kod možete kompajlirati iz komandne linije, bez potrebe za grafičkom aplikacijom. Nadalje, Go ne zanima naziv datoteke izvornog koda autonomnog programa, sve dok je naziv `main` paketa i u njemu se nalazi `main()` funkcija. Razlog je činjenica da je funkcija `main()` tamo gde započinje izvršavanje programa. Kao rezultat toga, ne možete imati više `main()` funkcija u datotekama jednog projekta.

Započećemo prvo kompajliranje Go programa sa programom `aSourceFile.go`, koji sadrži sledeći Go kod:

```
package main
import (
    "fmt"
)

func main() {
    fmt.Println("This is a sample Go program!")
}
```

Primetićete da Go zajednica radije daje `source_file.go` naziv izvornoj Go datoteci, umesto `aSourceFile.go`. Što god da odlučite, budite dosledni.

Da biste kompajlirali `aSourceFile.go` i kreirali **statički linkovanu** izvršnu datoteku, treba da izvršite sledeću naredbu:

```
$ go build aSourceFile.go
```

Nakon toga, imaćete novu izvršnu datoteku, pod nazivom `aSourceFile`, koju treba izvršiti:

```
$ file aSourceFile
aSourceFile: Mach-O 64-bit executable x86_64
$ ls -l aSourceFile
-rwxr-xr-x  1 mtsouk  staff  2007576 Jan 10 21:10 aSourceFile
$ ./aSourceFile
This is a sample Go program!
```

Glavni razlog zašto je veličina datoteke `aSourceFile` tolika je činjenica da je statički povezana, što znači da ne zahteva pokretanje spoljnih biblioteka.

IZVRŠAVANJE GO KODA

Postoji još jedan način da izvršite svoj Go kod, koji ne kreira ni jednu trajnu izvršnu datoteku - on samo generiše neke intermedijarne datoteke koje se posle automatski brišu.

Način prikaza omogućava da koristite Go kao da je to skriptni programski jezik, kao što su **Pajton**, **Rubi** ili **Perl**.

Dakle, da biste mogli da pokrenete `aSourceFile.go` bez kreiranja izvršne datoteke, moraćete da izvršite sledeću naredbu:

```
$ go run aSourceFile.go
This is a sample Go program!
```

Kao što vidite, izlaz iz prethodne naredbe potpuno je isti kao i pre.

Imajte na umu da, koristeći `go run`, Go prevodilac još uvek mora da kreira izvršnu datoteku. Ne vidite je; ona se izvršava automatski i automatski se briše nakon što se program završi da biste vi mogli pomisliti da nema potrebe za izvršnom datotekom.

U ovoj knjizi za izvršavanje primera koda uglavnom koristimo `go run`, prvenstveno zato što je to jednostavnije nego pokretanje `go build` i izvršne datoteke. Uz to, `go run` ne ostavlja ni jednu datoteku na vašem tvrdom disku nakon što se program završi.

DVA GO PRAVILA

Go ima stroga pravila kodiranja koja treba da vam pomognu da izbegnete glupe nepravilnosti i greške u svom kodu, a Go zajednici da olakša čitanje vašeg koda. U ovom odeljku ćemo predstaviti dva takva Go pravila koja morate da znate.

Kao što smo ranije pomenuli, imajte na umu da je Go prevodilac tu da vam pomogne, a ne da vam „zagorča“ život. Kao rezultat toga, glavna namena Go prevodioca je kompajliranje i povećavanje kvaliteta Go koda.

Ili koristite Go paket ili ga ne uključujte

Go ima stroga pravila o korišćenju paketa. Stoga ne možete jednostavno da uključite ni jedan paket za koji možda mislite da će vam biti potreban, a zatim da ga posle ne koristite.

Pogledajte sledeći naivni program, koji je sačuvan kao `packageNotUsed.go`:

```
package main

import (
    "fmt"
    "os"
)
```

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

U ovoj knjizi videćete mnogo poruka o greškama, situacija sa greškama i upozorenja. Verujem da je korisno gledanje koda koji nije prošao kompajliranje, a ponekad je čak i korisnije od gledanja samo onog Go koda koji je kompajliran bez ikakvih grešaka. Go kompajler prikazuje obično korisne poruke o greškama i upozorenja koja će vam najverovatnije pomoći da rešite problem pojave greške; zato nemojte potcenjivati poruke o greškama i upozorenja.

Ako izvršite `packageNotUsed.go`, dobićete sledeću poruku o grešci od Goa, a program neće biti izvršen:

```
$ go run packageNotUsed.go
# command-line-arguments
./packageNotUsed.go:5:2: imported and not used: "os"
```

Ako uklonite `os` paket sa liste programa `import`, `packageNotUsed.go` će sasvim ispravno biti kompajliran.

Iako ovo nije savršeno vreme za razgovor o kršenju Go pravila, postoji način da se zaobiđe to ograničenje, što je prikazano u sledećem Go kodu koji je sačuvan u `fileNotUsedUnderscore.go` datoteci:

```
package main

import (
    "fmt"
    _ "os"
)

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

Dakle, upotreba podvlake ispred naziva paketa na `import` listi neće kreirati poruku o grešci u procesu kompajliranja, čak i ako taj paket neće biti korišćen u programu:

```
$ go run packageNotUsedUnderscore.go
Hello there!
```

Razlog zbog kojeg Go dopušta da zaobiđete ovo pravilo postaće evidentniji u Poglavlju 6, *Ono što možda ne znate o Go paketima i funkcijama*.

Postoji samo jedan način za formatiranje vitičastih zagrada

Pogledajte sledeći Go program pod nazivom `curly.go`:

```
package main

import (
    "fmt"
)

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

Iako ovo izgleda sasvim u redu, ako pokušate da ga izvršite, bićete prilično razočarani, jer ćete dobiti sledeću poruku o **grešci u sintaksi** i kod neće biti kompiliran, pa, stoga, ni pokrenut:

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

Službeno objašnjenje ove poruke o pogrešci je da Go zahteva korišćenje znaka tačka-zarez za razdvajanje instrukcija u mnogim kontekstima, a kompajler automatski ubacuje potrebne znakove tačka-zarez kad „misli“ da su potrebne. Stoga, stavljanje otvorene vitičaste zagrade (`{`) u njen vlastiti red učiniće da Go kompajler umetne znak tačka-zarez na kraju prethodnog reda (`funkc main()`), što je uzrok poruke o pogrešci.

Preuzimanje Go paketa

Iako je standardna Go biblioteka veoma bogata, postoje trenuci kada treba da preuzmite spoljne Go pakete da biste koristili njihovu funkcionalnost. U ovom odeljku ćete saznati kako da preuzmete spoljni Go paket i gde će biti smešten na vašoj UNIX mašini.

Imajte na umu da će, iako bi Go **moduli** (nova osobina Goa koja je još uvek u fazi razvoja) mogli da uvedu izmene u vaše korišćenje spoljnog Go koda, postupak preuzimanja jednog Go paketa na vaš računar ostati isti.

Više detalja o Go paketima i Go modulima saznaćete u Poglavlju 6, *Ono što možda ne znate o Go paketima i funkcijama*.

Pogledajte sledeći jednostavni Go program koji je sačuvan kao `getPackage.go`:

```
package main

import (
    "fmt"
    "github.com/mactsouk/go/simpleGitHub"
)

func main() {
    fmt.Println(simpleGitHub.AddTwo(5, 6))
}
```

Ovaj program koristi eksterni paket, jer jedna od naredbi `import` koristi internet adresu. U ovom slučaju se spoljni paket zove `simpleGitHub` i nalazi se na github.com/mactsouk/go/simpleGitHub.

Ako odmah pokušate da pokrenete `getPackage.go`, razočaraćete se:

```
$ go run getPackage.go
getPackage.go:5:2: cannot find package
“github.com/mactsouk/go/simpleGitHub” in any of:
    /usr/local/Cellar/go/1.9.1/libexec/src/github.com/mactsouk/go/
simpleGitHub (from $GOROOT)
    /Users/mtsouk/go/src/github.com/mactsouk/go/simpleGitHub (from $GOPATH)
```

Dakle, moraćete da nabavite paket koji nedostaje na vašem računaru. Da biste ga preuzeli, moraćete da izvršite sledeću naredbu:

```
$ go get -v github.com/mactsouk/go/simpleGitHub
github.com/mactsouk/go (download)
github.com/mactsouk/go/simpleGitHub
```

Nakon toga, preuzete datoteke možete da pronađete u sledećem direktorijumu:

```
$ ls -l ~/go/src/github.com/mactsouk/go/simpleGitHub/
total 8
-rw-r--r--  1 mtsouk  staff  66 Oct 17 21:47 simpleGitHub.go
```

Međutim, naredba `go get` takođe kompajlira paket. Odgovarajuće datoteke možete pronaći na sledećem mestu:

```
$ ls -l ~/go/pkg/darwin_amd64/github.com/mactsouk/go/simpleGitHub.a
-rw-r--r--  1 mtsouk  staff  1050 Oct 17 21:47
/Users/mtsouk/go/pkg/darwin_amd64/github.com/mactsouk/go/simpleGitHub.a
```

Sada ste spremni da pokrenete `getPackage.go`, bez ikakvih problema:

```
$ go run getPackage.go
11
```

Možete da obrišete međudatoteke preuzetog Go paketa na sledeći način:

```
$ go clean -i -v -x github.com/mactsouk/go/simpleGitHub
cd /Users/mtsouk/go/src/github.com/mactsouk/go/simpleGitHub
rm -f simpleGitHub.test simpleGitHub.test.exe
rm -f /Users/mtsouk/go/pkg/darwin_amd64/github.com/mactsouk/go/
simpleGitHub.a
```

Slično tome, možete da izbrišete ceo Go paket koji ste preuzeli lokalno pomoću UNIX naredbe `rm(1)` za brisanje svog Go izvora nakon što upotrebite `go clean`:

```
$ go clean -i -v -x github.com/mactsouk/go/simpleGitHub
$ rm -rf ~/go/src/github.com/mactsouk/go/simpleGitHub
```

Nakon što izvršite prethodne naredbe, moraćete da ponovo preuzmete Go paket.

UNIX STDIN, STDOUT I STDERR

Svaki UNIX OS ima tri datoteke koje su stalno otvorene za njegove procese. Zapamtite da UNIX „smatra“ da je sve datoteka, čak i pisač ili miš.

UNIX koristi **deskriptore datoteka**, koji su pozitivne celobrojne vrednosti, kao unutarnji prikaz za pristup svim otvorenim datotekama, što je mnogo lepše od korišćenja dugačkih putanja.

Dakle, podrazumevano, svi UNIX sistemi podržavaju tri posebna i standardna naziva datoteka: `/dev/stdin`, `/dev/stdout` i `/dev/stderr`, kojima se takođe može pristupiti korišćenjem deskriptora 0, 1 i 2, respektivno. Ova tri deskriptora datoteka se takođe nazivaju **standardni ulaz**, **standardni izlaz** i **standardna greška**. Uz to se deskriptoru datoteka 0 može pristupiti kao `/dev/fd/0` na macOS mašini i kao `/dev/fd/0` i `/dev/pts/0` na Debian Linuks mašini.

Go koristi `os.Stdin` za pristup standardnom ulazu, `os.Stdout` za pristup standardnom izlazu i `os.Stderr` za pristup standardnoj grešci. Iako još uvek možete koristiti `/dev/stdin`, `/dev/stdout` i `/dev/stderr` ili srodne vrednosti deskriptora datoteka za pristup istom uređaju, bolje je, sigurnije i praktičnije držati se uz `os.Stdin`, `os.Stdout` i `os.Stderr` koje nudi Go.

O ISPISU IZLAZA

Kao i UNIX i C, Go nudi različite načine ispisa izlaza na ekranu. Sve funkcije ispisa u ovom odeljku zahtevaju upotrebu standardnog Go paketa `fmt` i prikazane su u programu `print.go`, koji će biti predstavljen u dva dela.

Najjednostavniji način ispisa nečega u Gou je korišćenje funkcija `fmt.Println()` i `fmt.Printf()`. Funkcija `fmt.Printf()` ima mnogo sličnosti sa C funkcijom `printf(3)`. Umesto funkcije `fmt.Println()`, možete koristiti i funkciju `fmt.Print()`. Glavna razlika između njih je što `fmt.Println()` automatski dodaje znak za novu liniju uvek kada je pozovete.

Sa druge strane, najveća razlika između funkcija `fmt.Println()` i `fmt.Printf()` je da ova druga zahteva **specifikator formata** za svaku „stvar" koju želite da ispišete, baš kao i C funkcija `printf(3)`, što znači da imate bolju kontrolu nad onim što radite, ali morate da napišete još koda. Ove **glagole** Go naziva specifikatori formata. Više informacija o glagolima možete pronaći na adresi https://golang.org/pkg/fmt/.

Ako morate da izvršite bilo kakvo formatiranje pre nego što nešto ispišete ili se mora aranžovati više varijabli, korišćenje `fmt.Printf()` bi moglo da bude bolji izbor. Međutim, ako morate da ispišete samo jednu varijablu, možda treba da odaberete `fmt.Print()` ili `fmt.Println ()`, zavisno od toga treba li vam znak za novu liniju ili ne.

Prvi deo `printing.go` sadrži sledeći Go kod:

```
package main

import (
    "fmt"
)

func main() {
    v1 := "123"
    v2 := 123
    v3 := "Have a nice day\n"
    v4 := "abc"
```

U ovom delu vidite `import` iz paketa `fmt` i definiciju četiri Go varijable. `\n` koji se koristi u `v3` je znak za novu liniju. Međutim, ako želite samo da umetnute novu liniju u vaš izlaz, možete pozvati `fmt.Println()` bez ikakvih argumenata, umesto da koristite nešto kao što je `fmt.Print("\n")`.

Drugi deo je kao u nastavku:

```
    fmt.Print(v1, v2, v3, v4)
    fmt.Println()
    fmt.Println(v1, v2, v3, v4)
    fmt.Print(v1, " ", v2, " ", v3, " ", v4, "\n")
    fmt.Printf("%s%d %s %s\n", v1, v2, v3, v4)
}
```

U ovom delu ispisujete četiri varijable, koristeći `fmt.Println()`, `fmt.Print()` i `fmt.Printf()`, da biste bolje razumeli kako se one razlikuju.

Ako izvršite `printing.go`, dobićete sledeći izlaz:

```
$ go run printing.go
123123Have a nice day
abc
123 123 Have a nice day
abc
123 123 Have a nice day
abc
123123 Have a nice day
abc
```

Kao što vidite iz prethodnog izlaza, funkcija `fmt.Println()` dodaje i karakter razmaka između njegovih parametara, što nije slučaj sa funkcijom `fmt.Print()`.

Kao rezultat, instrukcija kao što je `fmt.Println(v1, v2)` ekvivalentna je instrukciji `fmt.Print(v1, "", v2, "\n")`.

Osim `fmt.Println()`, `fmt.Printf()` i `fmt.Printf()`, koji su najjednostavnije funkcije koje se mogu koristiti za generisanje rezultata na ekranu, tu je i `S` porodica funkcija koje uključuju `fmt.Sprintln()`, `fmt.Sprint()` i `fmt.Sprintf()`. Ove funkcije se koriste za kreiranje stringova na osnovu određenog formata.

Konačno, tu je i `F` porodica funkcija koja uključuje `fmt.Fprintln()`, `fmt.Fprintf()` i `fmt.Fprintf()`. Koriste se za pisanje u datoteke, pri čemu se koristi `io.Writer`.

Više o `io.Writeru` i `io.Readeru` i **interfejsima** ćete saznati u Poglavlju 8, *Naložiti UNIX sistemu šta da radi*.

U sledećem odeljku ćete naučiti kako da ispišete podatke koristeći standardni izlaz, a to je prilično uobičajeno u UNIX svetu.

Upotreba standardnog izlaza

Standardni izlaz je, manje-više, jednak ispisu na ekran. Međutim, njegova upotreba može zahtevati upotrebu funkcija koje ne pripadaju `fmt` paketu, zbog čega je standardni izlaz i predstavljen u posebnom odeljku.

Biće ilustrovana relevantna tehnika u `stdOUT.go` i biće ponuđena u tri dela.

Prvi deo programa je kao u nastavku:

```
package main

import (
    "io"
    "os"
)
```

Paket `os` se koristi za čitanje argumenata komandne linije programa i za pristup datoteci `os.Stdout`.

Drugi deo `stdOUT.go` sadrži sledeći Go kod:

```
func main() {
    myString := ""
    arguments := os.Args
    if len(arguments) == 1 {
        myString = "Please give me one argument!"
    } else {
        myString = arguments[1]
    }
```

Varijabla `myString` čuva tekst koji će biti ispisan na ekranu, koji je ili prvi argument komandne linije programa ili je, ako je program izveden bez ikakvog argumenta komandne linije, ručno kodirana tekstualna poruka.

Treći deo programa je sledeći:

```
    io.WriteString(os.Stdout, myString)
    io.WriteString(os.Stdout, "\n")
}
```

U ovom slučaju funkcija `io.WriteString()` deluje na isti način kao i funkcija `fmt.Print()`; međutim, potrebna su samo dva parametra. Prvi parametar je datoteka u koju želite da pišete, što je, u ovom slučaju, `os.Stdout`, a drugi parametar je `string` varijabla.

Strogo govoreći, tip prvog parametra funkcije `io.WriteString()` treba da bude `io.Writer`, za šta je potrebna **kriška** bajtova kao drugi parametar. Međutim, u ovom slučaju `string` varijabla sasvim dobro obavlja posao. Više detalja o kriškama ćete saznati u Poglavlju 3, *Korišćenje osnovnih tipova Go podataka*.

Izvršavanje `stdOUT.go` će proizvesti sledeći izlaz:

```
$ go run stdOUT.go
Please give me one argument!
$ go run stdOUT.go 123 12
123
```

Prethodni izlaz proverava da li funkcija `io.WriteString()` šalje sadržaj svog drugog parametra na ekran kada joj je prvi parametar `os.Stdout`.

DOBIJANJE KORISNIČKOG UNOSA

Postoje tri glavna načina da se dobije korisnički unos: čitanje argumenata komandne linije programa, traženje unosa od korisnika i čitanje spoljnih datoteka. U ovom odeljku ćemo predstaviti prvi i drugi način. Ako želite da naučite kako da čitate spoljnu datoteku, treba da posetite Poglavlje 8, *Naložiti UNIX sistemu šta da radi*.

Nešto o operatorima := i =

Pre bilo kakvog nastavka razmatranja ove teme, biće veoma korisno da predstavimo upotrebu operatora := i objasnimo u čemu se on razlikuje od operatora =. Službeni naziv za := je **instrukcija skraćenog dodeljivanja**, koja se može upotrebiti umesto `var` deklaracije sa implicitnim tipom.

Retko ćete u Gou videti da se upotrebljava `var`. Ključna reč `var` se, uglavnom, koristi za deklarisanje globalnih varijabli u Go programima i za deklarisanje varijable bez početne vrednosti. Razlog za to je činjenica da svaka instrukcija koja postoji izvan koda neke funkcije mora započeti ključnom rečju, kao što su `func` ili `var`. To znači da se instrukcija skraćenog dodeljivanja ne može koristiti izvan funkcije, jer tamo nije dostupna.

Operator := funkcioniše na sledeći način:

```
m := 123
```

Rezultat prethodne instrukcije je nova celobrojna varijabla pod nazivom `m` sa vrednošću od `123`.

Međutim, ako pokušate da koristite := na već deklarisanoj varijabli, kompajliranje će izostati sa sledećom porukom o grešci, koja ima savršen smisao:

```
$ go run test.go
# command-line-arguments
./test.go:5:4: no new variables on left side of :=
```

Dakle, sada možete da se pitate šta će se dogoditi ako od funkcije očekujete dve ili više vrednosti, a za jednu od njih želite da koristite postojeću varijablu. Treba li koristiti := ili =? Odgovor je jednostavan: treba koristiti :=, kao u sledećem primeru koda:

```
i, k := 3, 4
j, k := 1, 2
```

Pošto se `j` varijabla prvi put koristi u drugoj instrukciji, koristite `:=` , iako je `k` već definisana u prvoj instrukciji.

Iako može izgledati dosadno govoriti o tako beznačajnim stvarima, njihovo poznavanje će vas, na duže staze, spasiti od raznih tipova grešaka!

Čitanje sa standardnog ulaza

Čitanje podataka sa standardnog ulaza će biti ilustrovano u `stdIN.gou`, koju ćete videti u dva dela. Prvi deo je sledeći:

```
package main

import (
    "bufio"
    "fmt"
    "os"
)
```

U prethodnom kodu, prvi put u ovoj knjizi, možete videti upotrebu `bufio` paketa.

Više o paketu `bufio` saznaćete u Poglavlju 8, *Naložiti UNIX sistemu šta da radi*.

Iako se paket `bufio` uglavnom koristi za ulaz i izlaz datoteke, `os` paket ćete susretati sve vreme u ovoj knjizi, jer sadrži mnogo korisnih funkcija; njegova najčešća funkcionalnost je što obezbeđuje način za pristup argumentima komandne linije za Go program (`os.Args`).

Službeni opis ukazuje da `os` paket nudi funkcije koje izvode OS operacije. Ovo uključuje funkcije za kreiranje, brisanje i preimenovanje datoteka i direktorijuma i funkcije za učenje UNIX dozvola i drugih karakteristika datoteka i direktorijuma. Glavna prednost `os` paketa je da je nezavisan od platforme. Jednostavno rečeno, njegove funkcije će raditi i na UNIX i na Microsoft Windows mašinama.

Drugi deo `stdIN.goa` sadrži sledeći Go kod:

```
func main() {

var f *os.File
f = os.Stdin
defer f.Close()
```

```
        scanner := bufio.NewScanner(f)
        for scanner.Scan() {
            fmt.Println(">", scanner.Text())
        }
    }
```

Prvo, poziva se `bufio.NewScanner()` korišćenjem standardnog (`os.Stdin`) ulaza kao njegovog parametra. Ovaj poziv vraća `bufio.Scanner` varijablu, koja se koristi sa `Scan()` funkcijom za čitanje red po red iz `os.Stdin`a Svaki pročitani red se ispisuje na ekranu pre nego što se dobije sledeći. Imajte na umu da svaki red koji program ispisuje započinje > znakom.

Izvođenje `stdIN.goa` će proizvesti sledeću vrstu rezultata:

```
$ go run stdIN.go
This is number 21
> This is number 21
This is Mihalis
> This is Mihalis
Hello Go!
> Hello Go!
Press Control + D on a new line to end this program!
> Press Control + D on a new line to end this program!
```

Prema UNIX načinu, kombinacijom *Ctrl* + *D* možete zatražiti od programa da prestane da čita podatke sa standardnog ulaza.

Go kodovi `stdIN.go` i `stdOUT.go` će nam biti vrlo korisni kad budemo razmatrali UNIX **cevovode** u Poglavlju 8, *Naložiti UNIX sistemu šta da radi*, tako da ne treba da potcenjujete njihovu jednostavnost.

Korišćenje argumenata komandne linije

Tehnika rada u ovom odeljku će biti ilustrovana pomoću Go koda u datoteci `cla.go`, koji će biti predstavljen u tri dela. Program će pronaći minimum i maksimum argumenata svoje komandne linije.

Prvi deo programa je sledeći:

```
package main

import (
    "fmt"
    "os"
    "strconv"
)
```

Važno je shvatite da dobijanje argumenata komadne linije zahteva upotrebu `os` paketa. Pored toga vam je potreban još jedan paket, pod nazivom `strconv`, da biste mogli da pretvorite argument komandne linije, koji je dat kao string, u aritmetički tip podataka.

Drugi deo programa je sledeći:

```
func main() {
    if len(os.Args) == 1 {
        fmt.Println("Please give one or more floats.")
        os.Exit(1)
    }

    arguments := os.Args
    min, _ := strconv.ParseFloat(arguments[1], 64)
    max, _ := strconv.ParseFloat(arguments[1], 64)
```

Ovde `cla.go` proverava da li imate neke argumente komandne linije, tako što proverava dužinu `os.Args`. Naime, programu treba bar jedan argument komandne linije da bi radio. Imajte na umu da je `os.Args` Go kriška sa vrednostima `string`. Prvi element u krišci je naziv izvršnog programa. Dakle, u cilju inicijalizacije `min` i `max` varijabli, moraćete da koristite drugi element tipa `string`, kriške `os.Args` koja ima vrednost indeksa 1.

Činjenica da očekujete jedan ili više brojeva sa pokretnim zarezom ne znači nužno da će vam korisnik dati validne brojeve sa pokretnim zarezom, slučajno ili namerno. Međutim, pošto do sada nismo govorili o rukovanju greškama u programu Go, `cla.go` pretpostavlja da su svi argumenti komandne linije u ispravnom formatu i stoga će biti prihvatljivi. Kao rezultat toga, `cla.go` ignoriše vrednost `error` koju vraća funkcija `strconv.ParseFloat ()`, koristeći sledeću instrukciju:

```
n, _ := strconv.ParseFloat(arguments[i], 64)
```

Prethodna instrukcija ukazuje Gou da želite da vratite samo prvu vrednost, koristeći `strconv.ParseFloat()`, tako što ste joj dodelili podvlaku, a da vas ne zanima druga vrednost, koja je u ovom slučaju varijabla `error`. Podvlaka, koja se naziva **prazan identifikator**, ukazuje na način na koji Go odbacuje vrednost. Ako Go funkcija vraća više vrednosti, prazan identifikator se možete koristiti više puta.

Zanemarivanje svih ili nekih od povratnih vrednosti Go funkcije, posebno vrednosti `error`, veoma je opasna tehnika koja se ne sme koristiti u produkcijskom kodu!

Treći deo programa dolazi sa sledećim Go kodom:

```
    for i := 2; i < len(arguments); i++ {
        n, _ := strconv.ParseFloat(arguments[i], 64)

        if n < min {
            min = n
        }
        if n > max {
            max = n
        }
    }

    fmt.Println("Min:", min)
    fmt.Println("Max:", max)
}
```

Ovde koristite `for` petlju koja će vam pomoći da posetite sve elemente `os.Args` kriške, koja je prethodno dodeljena varijabli `arguments`.

Izvođenje `cla.go` će kreirati sledeću vrstu rezultata:

```
$ go run cla.go -10 0 1
Min: -10
Max: 1
$ go run cla.go -10
Min: -10
Max: -10
```

Program se ne ponaša dobro kada primi pogrešan unos; najgore od svega je što on prilikom obrade argumenata komandne linije programa ne generiše nikakva upozorenja da treba da obavesti korisnika da je tu bila je greška (ili nekoliko grešaka):

```
$ go run cla.go a b c 10
Min: 0
Max: 10
```

O GREŠCI IZLAZA

U ovom odeljku ćemo predstaviti tehniku slanja podataka na **UNIX standard error**, a to je UNIX način razlikovanja stvarnih vrednosti i greške izlaza.

Go kod za ilustrovanje upotrebe standardne greške u Gou uključen je u `stdERR.go` i biće predstavljen u dva dela. Pošto pisanje na standardni izlaz za grešku zahteva upotrebu datoteke deskriptora koja se odnosi na standardnu grešku, Go kod `stdERR.go` će se zasnivati na Go kodu `stdOUT.go`.

Prvi deo programa je sledeći:

```
package main

import (
    "io"
    "os"
)
func main() {
    myString := ""
    arguments := os.Args
    if len(arguments) == 1 {
        myString = "Please give me one argument!"
    } else {
        myString = arguments[1]
    }
```

Do sada je `stdERR.go` skoro identičan sa `stdOUT.go`.

Drugi deo `stdERR.go` je sledeći:

```
    io.WriteString(os.Stdout, "This is Standard output\n")
    io.WriteString(os.Stderr, myString)
    io.WriteString(os.Stderr, "\n")
}
```

Dva puta pozivate `io.WriteString()` za pisanje na standardni izlaz za grešku (`os.Stderr`) i još jedan put za pisanje u standardni izlaz (`os.Stdout`).

Izvođenje programa `stdERR.go` će kreirati sledeći izlaz:

```
$ go run stdERR.go
This is Standard output
Please give me one argument!
```

Prethodni izlaz vam ne može pomoći u razlikovanju podataka zapisanih u standardni izlaz i podataka zapisanih u standardnu grešku. Međutim, ako koristite `bash(1)` ljusku, postoji trik koji možete da koristite da biste napravilii razliku između standardnih izlaznih podataka i standardnih podataka o grešci. Skoro sve UNIX školjke nude ovu funkcionalnost na svoj način.

Kada koristite `bash(1)`, standardni izlaz pogreške možete preusmeriti u datoteku na sledeći način:

```
$ go run stdERR.go 2>/tmp/stdError
This is Standard output
$ cat /tmp/stdError
Please give me one argument!
```

Broj nakon naziva UNIX programa ili sistemskog poziva se odnosi na sekciju priručnika kojem pripada njegova stranica. Iako se većina naziva može naći samo jednom na stranicama priručnika, što znači da stavljanje broja sekcije nije potrebno, postoje nazivi koji se mogu nalaziti u više sekcija, jer imaju više značenja, kao što su `crontab(1)` i `crontab(5)`. Stoga, ako pokušate da preuzmete stranicu priručnika sa nazivom koji ima višestruko značenje bez navođenja broja njegove sekcije, dobićete unos koji ima najmanji broj sekcije.

Slično tome, možete odbaciti izlaz greške njegovim preusmeravanjem na `/dev/null` uređaj, odnosno kao da „kažete“ UNIX-u da ga potpuno zanemari:

```
$ go run stdERR.go 2>/dev/null
This is Standard output
```

U dva primera deskriptor datoteke standardne greške smo preusmerili u datoteku i u `/dev/null`, respektivno. Ako želite sačuvati i standardni izlaz i standardnu grešku u istoj datoteci, možete preusmeriti deskriptor datoteke standardne greške (`2`) na deskriptor datoteke standardnog izlaza (`1`). Sledeća naredba pokazuje tehniku koja je prilično uobičajena u UNIX sistemima:

```
$ go run stdERR.go >/tmp/output 2>&1
$ cat /tmp/output
This is Standard output
Please give me one argument!
```

Konačno, možete poslati i standardni izlaz i standardnu grešku u `/dev/null` na sledeći način:

```
$ go run stdERR.go >/dev/null 2>&1
```

PISANJE U DATOTEKE DNEVNIKA

Paket log omogućava slanje poruka dnevnika u servis dnevnika sistema vaše UNIX mašine, dok vam Go paket `syslog`, koji je deo `log` paketa, omogućava da definišete **nivo dnevnika** i uređaj **objekat dnevnika** koji će koristiti vaš Go program.

Obično se većina datoteka sistemskih dnevnika UNIX OS-a može naći u `/var/log` direktorijumu. Međutim, datoteke dnevnika za mnoge popularne servise, kao što su Apache i Nginx, zavisno od njihove konfiguracije, mogu se nalaziti posvuda.

Generalno govoreći, korišćenje datoteke dnevnika za zapisivanje nekih informacija smatra se boljom praksom nego pisanje istog izlaza na ekranu zbog dva razloga: prvo zato što se izlaz ne gubi, jer se smešta u datoteku, i, drugo, možete pretraživati i obraditi datoteke dnevnika pomoću UNIX alata, kao što su `grep(1)`, `awk(1)` i `sed(1)`, a što se ne može uraditi kada se poruke ispisuju u prozoru terminala.

Paket `log` nudi brojne funkcije za slanje izlaza na syslog server UNIX mašine. Popis funkcija obuhvata `log.Printf()`, `log.Print()`, `log.Println()`, `log.Fatalf()`, `log.Fatalln()`, `log.Panic()`, `log.Panicln()` i `log.Panicf()`.

Imajte na umu da funkcije pisanja u dnevnik mogu biti izuzetno korisne za uklanjanje grešaka iz vaših programa, posebno serverskih procesa napisanih u Gou, tako da ne bi trebalo da potcenjujete njihovu moć.

Nivoi dnevnika

Nivoi dnevnika je vrednost koja određuje ozbiljnost unosa u dnevnik. Postoje razni nivoi dnevnika, uključujući i *debug, info, notice, warning, err, crit, alert i emerg*, sa obrnutim redosledom ozbiljnosti.

Objekti dnevnika

Objekat **dnevnika** je sličan klasi koja se koristi za beleženje informacija. Vrednost objekta dnevnika može biti *auth, authpriv, cron, daemon, kern, lpr, mail, mark, news, syslog, user,UUCP, local0, local1, local2, local3, local4, local5, local6* ili *local7*, a definiše se unutar `/etc/syslog.conf`, `/etc/rsyslog.conf` ili neke druge odgovarajuće datoteke, zavisno od procesa servera koji se koristi za zapisivanje u dnevnik sistema na vašoj UNIX mašini.

To znači da ako alat za pisanje u dnevnik nije definisan i stoga se njime ne upravlja, poruke dnevnika koje pošaljete mogu se ignorisati i, samim tim, izgubiti.

Serveri dnevnika

Sve UNIX mašine imaju zaseban proces servera koji je odgovoran za primanje podataka za dnevnik i za njihovo zapisivanje u datoteke dnevnika. Postoje razni serveri za dnevnik koji rade na UNIX mašinama. Međutim, samo se dva od njih koriste na većini UNIX varijanti: `syslogd(8)` i `rsyslogd(8)`.

Na macOS mašinama naziv procesa je `syslogd(8)`. Sa druge strane, većina Linuks mašina koristi `rsyslogd(8)`, što je poboljšana i pouzdanija verzija od procesa `(8)`, koji je bio izvorni uslužni program UNIX-a za zapisivanje poruka u dnevnik.

Međutim, bez obzira koja je UNIX varijanta koju koristite ili naziv serverskog procesa koji se koristi za zapisivanje u dnevnik, zapisivanje u dnevnik deluje na isti način na svim UNIX mašinama i stoga ne utiče na Go kod koji ćete napisati.

Konfiguracijska datoteka `rsyslogd(8)` se, obično, naziva `rsyslog.conf` i nalazi se u `/etc`. Sadržaj konfiguracijske datoteke `rsyslog.conf`, bez redova sa komentarima i linija koje počinju sa `$`, može izgledati ovako:

```
$ grep -v '^#' /etc/rsyslog.conf | grep -v '^$' | grep -v '^\$'
auth,authpriv.*                 /var/log/auth.log
*.*;auth,authpriv.none          -/var/log/syslog
daemon.*                        -/var/log/daemon.log
kern.*                          -/var/log/kern.log
lpr.*                           -/var/log/lpr.log
mail.*                          -/var/log/mail.log
user.*                          -/var/log/user.log
mail.info                       -/var/log/mail.info
mail.warn                       -/var/log/mail.warn
mail.err                        /var/log/mail.err
news.crit                       /var/log/news/news.crit
news.err                        /var/log/news/news.err
news.notice                     -/var/log/news/news.notice
*.=debug;\
    auth,authpriv.none;\
    news.none;mail.none         -/var/log/debug
*.=info;*.=notice;*.=warn;\
    auth,authpriv.none;\
    cron,daemon.none;\
    mail,news.none              -/var/log/messages
*.emerg                 :omusrmsg:*

daemon.*;mail.*;\
    news.err;\
    *.=debug;*.=info;\
    *.=notice;*.=warn    |/dev/xconsole
local7.* /var/log/cisco.log
```

Dakle, da biste poslali svoje informacije za dnevnik u `/var/log/cisco.log`, morate da koristite objekat *local7* za objekat dnevnika. Znak zvezde nakon naziva objekta ukazuje serveru dnevnika da treba da hvata svaki nivo dnevnika koji ide na *local7* objekat i da ga zapisuje u `/var/log/cisco.log`.

Server syslogd(8) ima prilično sličnu konfiguracijsku datoteku koja je, obično, `/etc/syslog.conf`. Na macOS Haj Sijera uređaju `/etc/syslog.conf` datoteka je skoro prazna i zamenjena je sa `/etc/asl.conf`. Ipak, logika koja stoji iza konfiguracije u `/etc/syslog.conf`, `/etc/rsyslog.conf` i `/etc/asl.conf` je ista.

Go program koji šalje informacije u dnevnik

Go kod u `logFiles.gou` će objasniti korišćenje paketa `log` i `log/syslog` za pisanje u datoteke dnevnika sistema.

Imajte na umu da `log/syslog` paket nije implementiran u Microsoft Windows verziji Go.

Prvi deo `logFiles.goa` je sledeći:

```
package main

import (
    "fmt"
    "log"
    "log/syslog"
    "os"
    "path/filepath"
)

func main() {
    programName := filepath.Base(os.Args[0])
    sysLog, err := syslog.New(syslog.LOG_INFO|syslog.LOG_LOCAL7,
 programName)
```

Prvi parametar funkcije `syslog.New()` je prioritetan, a predstavlja kombinaciju objekta dnevnika i nivoa dnevnika. Stoga će prioritet `LOG_NOTICE | LOG_MAIL`, koji se navodi kao primer, poslati poruku o nivou dnevnika `MAIL` objektu dnevnika.

Kao rezultat toga, prethodni kod postavlja zadato upisivanje u objekat *local7* dnevnika, koristeći *info* nivo za upisivanje. Drugi parametar funkcije `syslog.New()` je naziv procesa koji će se pojaviti u dnevnicima kao pošiljalac poruke. Generalno gledano, smatra se dobrom praksom da se koristi pravi naziv izvršne datoteke da bi se lako mogle pronaći željene informacije u datotekama dnevnika u neko drugo vreme.

Drugi deo programa sadrži sledeći Go kod:

```
if err != nil {
    log.Fatal(err)
} else {
    log.SetOutput(sysLog)
}
log.Println("LOG_INFO + LOG_LOCAL7: Logging in Go!")
```

Nakon poziva na `syslog.New()`, moraćete da proverite `error` varijablu koja je vraćena iz njega, tako da možete biti sigurni da je sve u redu. Ako je sve u redu (to znači da je vrednost varijable `error` jednaka `nil`), pozivate funkciju `log.SetOutput()`, koja postavlja odredište izlaza zadatog dnevnika, što je u ovom slučaju dnevnik koji ste kreirali ranije (`sysLog`). Zatim, možete koristiti `log.Println()` za slanje informacija u dnevnik servera.

Treći deo `logFiles.goa` dolazi sa sledećim kodom:

```
    sysLog, err = syslog.New(syslog.LOG_MAIL, "Some program!")
    if err != nil {
        log.Fatal(err)
    } else {
        log.SetOutput(sysLog)
    }

    log.Println("LOG_MAIL: Logging in Go!")
    fmt.Println("Will you see this?")
}
```

Poslednji deo pokazuje da u svojim programima možete promeniti konfiguraciju zapisivanja u dnevnik onoliko puta koliko želite i da još uvek možete koristiti `fmt.Println()` za ispis rezultata na ekran.

Izvršavanje `logFiles.goa` će kreirati sledeći izlaz na ekranu Debian Linuks mašine:

```
$ go run logFiles.go
Broadcast message from systemd-journald@mail (Tue 2017-10-17 20:06:08
EEST):
logFiles[23688]: Some program![23688]: 2017/10/17 20:06:08 LOG_MAIL:
Logging in Go!
Message from syslogd@mail at Oct 17 20:06:08 ...
Some program![23688]: 2017/10/17 20:06:08 LOG_MAIL: Logging in Go!
Will you see this?
```

Izvođenje istog Go koda na macOS Haj Sijera uređaju generiše sledeći izlaz:

```
$ go run logFiles.go
Will you see this?
```

Imajte na umu da većina UNIX mašina smešta podatke dnevnika u više datoteka dnevnika, što je slučaj i sa Debian Linuks mašinom koja se koristi u ovom odeljku. Kao rezultat toga, `logFiles.go` šalje svoj izlaz u više datoteka evidencije, što se može verifikovati izlazom sledećih naredbi ljuske:

```
$ grep LOG_MAIL /var/log/mail.log
Oct 17 20:06:08 mail Some program![23688]: 2017/10/17 20:06:08 LOG_MAIL:
Logging in Go!
$ grep LOG_LOCAL7 /var/log/cisco.log
Oct 17 20:06:08 mail logFiles[23688]: 2017/10/17 20:06:08 LOG_INFO +
LOG_LOCAL7: Logging in Go!
$ grep LOG_ /var/log/syslog
Oct 17 20:06:08 mail logFiles[23688]: 2017/10/17 20:06:08 LOG_INFO +
LOG_LOCAL7: Logging in Go!
Oct 17 20:06:08 mail Some program![23688]: 2017/10/17 20:06:08 LOG_MAIL:
Logging in Go!
```

Prethodni izlaz pokazuje da je poruka instrukcije `log.Println("LOG_INFO +LOG_LOCAL7: Logging in Go!")` upisana u `/var/log/cisco.log` i u `/var/log/syslog`, dok je poruka instrukcije `log.Println("LOG_MAIL: Logging in Go!")` upisana u `/var/log/syslog` i u `/var/log/mail.log`.

Važno je da zapamtite da, ako server dnevnika UNIX mašine nije konfigurisan za hvatanje svih objekata dnevnika, neki unosi u dnevnik koje pošaljete mogu biti odbačeni bez ikakvih upozorenja.

Nešto o funkciji log.Fatal()

U ovom odeljku ćete na delu videti funkciju `log.Fatal()`. Ona se koristi kada se dogodi nešto loše i samo želite da što brže izađete iz svog programa nakon prijavljivanja loše situacije.

Upotreba funkcije `log.Fatal()` prikazana je u programu `logFatal.go`, koji sadrži sledeći go kod:

```
package main

import (
    "fmt"
    "log"
    "log/syslog"
)

func main() {
    sysLog, err := syslog.New(syslog.LOG_ALERT|syslog.LOG_MAIL, "Some
program!")
    if err != nil {
        log.Fatal(err)
    } else {
        log.SetOutput(sysLog)
    }

    log.Fatal(sysLog)
    fmt.Println("Will you see this?")
}
```

Izvršavanje funkcije `log.Fatal()` će kreirati sledeći izlaz:

```
$ go run logFatal.go
exit status 1
```

Upotreba funkcije `log.Fatal()` prekida Go program u tački koja se zove `log.Fatal()`, što je razlog što niste videli izlaz iz instrukcije `fmt.Println("Will you see this?")`.

Međutim, zbog parametara poziva `syslog.New()`, unos dnevnika je dodat u datoteku dnevnika koja se odnosi na poštu, a to je `/var/log/mail.log`:

```
$ grep "Some program" /var/log/mail.log
Jan 10 21:29:34 iMac Some program![7123]: 2019/01/10 21:29:34 &{17 Some
program! iMac.local   {0 0} 0xc00000c220}
```

Nešto o funkciji log.Panic()

Postoje situacije u kojima jedan program definitivno neće proći, a želite da imate što više informacija o otkazu.

U takvim teškim vremenima možete razmisliti o mogućnosti da koristite `log.Panic()`, a to je funkcija za pisanje u dnevnik koja je prikazana u ovom odeljku pomoću Go koda `logPanic.go`.

Go kod `logPanic.go` je sledeći:

```
package main

import (
    "fmt"
    "log"
    "log/syslog"
)

func main() {
    sysLog, err := syslog.New(syslog.LOG_ALERT|syslog.LOG_MAIL, "Some
program!")
    if err != nil {
        log.Fatal(err)
    } else {
        log.SetOutput(sysLog)
    }

    log.Panic(sysLog)
    fmt.Println("Will you see this?")
}
```

Izvršavanje `logPanic.goa` na macOS Mojave proizvešće sledeći izlaz:

```
$ go run logPanic.go
panic: &{17 Some program! iMac.local   {0 0} 0xc0000b21e0}
goroutine 1 [running]:
log.Panic(0xc00004ef68, 0x1, 0x1)
    /usr/local/Cellar/go/1.11.4/libexec/src/log/log.go:326 +0xc0
main.main()
    /Users/mtsouk/Desktop/mGo2nd/Mastering-Go-Second-
Edition/ch01/logPanic.go:17 +0xd6
exit status 2
```

Pokretanje istog programa na Debian Linuksu sa Go verzijom 1.3.3 generisaće sledeći izlaz:

```
$ go run logPanic.go
panic: &{17 Some program! mail   {0 0} 0xc2080400e0}
goroutine 16 [running]:
runtime.panic(0x4ec360, 0xc208000320)
    /usr/lib/go/src/pkg/runtime/panic.c:279 +0xf5
```

```
log.Panic(0xc208055f20, 0x1, 0x1)
    /usr/lib/go/src/pkg/log/log.go:307 +0xb6
main.main()
    /home/mtsouk/Desktop/masterGo/ch/ch1/code/logPanic.go:17 +0x169
goroutine 17 [runnable]:
runtime.MHeap_Scavenger()
    /usr/lib/go/src/pkg/runtime/mheap.c:507
runtime.goexit()
    /usr/lib/go/src/pkg/runtime/proc.c:1445
goroutine 18 [runnable]:
bgsweep()
    /usr/lib/go/src/pkg/runtime/mgc0.c:1976
runtime.goexit()
    /usr/lib/go/src/pkg/runtime/proc.c:1445
goroutine 19 [runnable]:
runfinq()
    /usr/lib/go/src/pkg/runtime/mgc0.c:2606
runtime.goexit()
    /usr/lib/go/src/pkg/runtime/proc.c:1445
exit status 2
```

Dakle, izlaz `log.Panic()` sadrži dodatne informacije na niskom nivou koje će vam, nadamo se, pomoći da nalazite rešenja u teškim situacijama koje se događaju u vašem Go kodu.

Analogno funkciji `log.Fatal()`, upotreba funkcije `log.Panic()` će dodati unos u odgovarajuću datoteku dnevnika i odmah će prekinuti Go program.

Zapisivanje u datoteku prilagođenog dnevnika

Ponekad je potrebno samo da upišete svoje podatke dnevnika u datoteku po vašem izboru. Ovo se može desiti zbog više razloga, uključujući pisanje podataka za uklanjanje grešaka, što ponekad može biti i previše, a da ne pravite nered sa datotekama dnevnika sistema, čuvajući vlastite podatke dnevnika odvojene od dnevnika sistema kako biste ih preneli ili sačuvali u bazu podataka, odnosno kako biste sačuvali svoje podatke koristeći drugačiji format. U ovom pododeljku ćete naučiti kako da pišete u datoteku prilagođenog dnevnika.

Naziv uslužnog Go programa će biti `customLog.go`, a upotrebljava se datoteka dnevnika `/tmp/mGo.log`.

Go kod `customLog.goa` biće predstavljen u tri dela. Prvi deo je sledeći:

```
package main

import (
```

```
    "fmt"
    "log"
    "os"
)

var LOGFILE = "/tmp/mGo.log"
```

Putanja do datoteke dnevnika se ručno kodira u `customLog.go` korišćenjem globalne varijable pod nazivom `LOGFILE`. Za potrebe ovog poglavlja datoteka dnevnika se nalazi u `/tmp` direktorijumu, što nije uobičajeno mesto za skladištenje podataka, jer se, obično, `/tmp` direktorijum isprazni nakon svakog ponovnog pokretanja sistema. Međutim, u ovom trenutku će vas to spasiti od potrebe za izvršavanjem datoteke `customLog.go` sa root privilegijama i stavljanjem nepotrebnih datoteka u sistemske direktorijume. Ako se ikad odlučite da koristite kod `customLog.go` u stvarnoj aplikaciji, tu putanju bi trebalo da promenite u nešto racionalnije.

Drugi deo `customLog.goa` je sledeći:

```
func main() {
    f, err := os.OpenFile(LOGFILE, os.O_APPEND|os.O_CREATE|os.O_WRONLY,
0644)

    if err != nil {
        fmt.Println(err)
        return
    }
    defer f.Close()
```

Poslednji deo `customLog.goa` je sledeći:

```
    iLog := log.New(f, "customLogLineNumber ", log.LstdFlags)

    iLog.SetFlags(log.LstdFlags)
    iLog.Println("Hello there!")
    iLog.Println("Another log entry!")
}
```

Ako pogledate stranicu sa dokumentacijom `log` paketa koju možete naći na adresi https://golang.org/pkg/log/, videćete da funkcija `SetFlags` omogućava da postavite izlazne zastavice (opcije) za trenutnog zapisivača. Zadate vrednosti, kako su definisane u funkciji `LstdFlags`, su `Ldate` i `Ltime`, što znači da ćete dobiti trenutni datum i vreme u svakom zapisu koji unesete u svoju datoteku dnevnika.

Izvršavanje `customLog.goa` neće generisaati vidljiv izlaz. Međutim, nakon što se izvrši dva puta, sadržaj `/tmp/mGo.loga` će biti sledeći:

```
$ go run customLog.go
$ cat /tmp/mGo.log
customLog 2019/01/10 18:16:09 Hello there!
customLog 2019/01/10 18:16:09 Another log entry!
$ go run customLog.go
$ cat /tmp/mGo.log
customLog 2019/01/10 18:16:09 Hello there!
customLog 2019/01/10 18:16:09 Another log entry!
customLog 2019/01/10 18:16:17 Hello there!
customLog 2019/01/10 18:16:17 Another log entry!
```

Ispis brojeva linija u unosima dnevnika

U ovom odeljku ćete naučiti kako da ispišete broj reda izvorne datoteke koja je izvršila instrukciju, a instrukcija je upisala unos u datoteku dnevnika, koristeći Go kod `customLogLineNumber.go`. Ovo će biti predstavljeno u dva dela. Prvi deo je sledeći:

```
package main

import (
    "fmt"
    "log"
    "os"
)

var LOGFILE = "/tmp/mGo.log"

func main() {
    f, err := os.OpenFile(LOGFILE, os.O_APPEND|os.O_CREATE|os.O_WRONLY,
0644)

    if err != nil {
        fmt.Println(err)
        return
    }
    defer f.Close()
```

Za sada, nema ništa posebno u poređenju sa kodom `customLog.go`.

Preostali Go kod `customLogLineNumber.go` je sledeći:

```
    iLog := log.New(f, "customLogLineNumber ", log.LstdFlags)
    iLog.SetFlags(log.LstdFlags | log.Lshortfile)
    iLog.Println("Hello there!")
    iLog.Println("Another log entry!")
}
```

Sva „magija“ se događa sa instrukcijom `iLog.SetFlags` (`log.LstdFlags | log.Lshortfile`), koja, osim `log.LstdFlags`, uključuje i `log.Lshortfile`. Potonja zastavica samom unosu u dnevnik dodaje puni naziv datoteke i broj linije Go instrukcije koja je ispisala unos u dnevnik.

Izvršavanje `customLogLineNumber.goa` neće generisati vidljivi izlaz. Međutim, nakon dva izvršavanja `customLogLineNumber.goa`, sadržaj datoteke `/tmp/mGo.log` dnevnika će biti sličan sledećem:

```
$ go run customLogLineNumber.go
$ cat /tmp/mGo.log
customLogLineNumber 2019/01/10 18:25:14 customLogLineNumber.go:26: Hello
there!
customLogLineNumber 2019/01/10 18:25:14 customLogLineNumber.go:27: Another
log entry!
$ go run customLogLineNumber.go
$ cat /tmp/mGo.log
customLogLineNumber 2019/01/10 18:25:14 customLogLineNumber.go:26: Hello
there!
customLogLineNumber 2019/01/10 18:25:14 customLogLineNumber.go:27: Another
log entry!
customLogLineNumber 2019/01/10 18:25:23 customLogLineNumber.go:26: Hello
there!
customLogLineNumber 2019/01/10 18:25:23 customLogLineNumber.go:27: Another
log entry!
```

Kao što vidite, korišćenje dugačkih naziva za alate komandne linije predstavlja razlog što je teško čitati datoteke dnevnika.

U Poglavlju 2, *Razumevanje Go komponenata*, naučićete kako se koristi ključna reč `defer` za elegantnije ispisivanja poruka dnevnika Go funkcije.

RUKOVANJU GREŠKAMA U GOU

Greške i rukovanje greškama su dve vrlo važne Go teme. Go „mnogo voli“ poruke o greškama, čak toliko da ima zaseban tip podataka za greške, pod nazivom `error`. To takođe znači da možete lako da kreirate vlastite **poruke o grešci** ako ustanovite da ono što vam pruža Go nije adekvatno.

Najverovatnije će biti potrebno da, kada razvijate vlastite Go pakete, kreirate vlastite greške i da upravljate njima.

Imajte na umu da je stanje greške jedno, a da je odlučivanje kako reagovati na stanje greške potpuno drugo. Jednostavno rečeno, nisu svi uslovi za greške podjednako kreirani, što znači da bi neki uslovi greške mogli da zahtevaju da se odmah zaustavi izvršavanje programa, dok bi za druge situacije sa greškom mogao biti potreban ispis poruke upozorenja koju korisnik može da vidi tokom nastavka izvršavanja programa. Programer treba da koristi zdrav razum i odluči šta da radi sa svakom vrednošću `error` koju program može dati.

Greške u Gou nisu poput izuzetaka ili grešaka u drugim programskim jezicima; to su normalni Go objekti koji se vraćaju iz funkcija ili metoda, kao i svaka druga vrednost.

Tip podataka o grešci

Postoji mnogo scenarija u kojima ćete možda morati da rešavate novi slučaj greške dok razvijate vlastitu Go aplikaciju. Tip podataka `error` treba da vam pomogne da definišete vlastite greške.

U ovom pododeljku ćete naučiti kako da kreirate vlastite `error` varijable. Kao što ćete videti, da biste kreirali novu `error` varijablu, morate da pozovete funkciju `New()` iz `error` standardnog Go paketa.

Primer Go koda za ilustraciju ovog procesa se može naći u datoteci `newError.go` koja će biti predstavljena u dva dela. Prvi deo programa je sledeći:

```
package main

import (
    "errors"
    "fmt"
)

func returnError(a, b int) error {
    if a == b {
```

```
            err := errors.New("Error in returnError() function!")
            return err
        } else {
            return nil
        }
    }
```

Ovde se događa mnogo štošta zanimljivo. Pre svega, možete prvi put u ovoj knjizi videti definiciju Go funkcije koja nije `main()`. Naziv ove nove naivne funkcije je `returnError()`. Uz to, možete da vidite u akciji `errors.New()` funkciju koja uzima vrednost `string` kao svoj parametar. Na kraju, ako funkcija treba da vrati varijablu `error`, ali nema greške za prijavljivanje, umesto toga, vraća `nil`.

O različitim tipovima Go funkcija saznaćete više u Poglavlju 6, *Ono što možda ne znate o Go paketima i Go funkcijama*.

Drugi deo `newError.goa` je sledeći:

```
func main() {
    err := returnError(1, 2)
    if err == nil {
        fmt.Println("returnError() ended normally!")
    } else {
        fmt.Println(err)
    }

    err = returnError(10, 10)
    if err == nil {
        fmt.Println("returnError() ended normally!")
    } else {
        fmt.Println(err)
    }

    if err.Error() == "Error in returnError() function!" {
        fmt.Println("!!")
    }
}
```

Kao što kod ilustruje, mnogo vremena morate da posvetite proveravanju da li je `error` varijabla jednaka `nil` i onda da postupite u skladu sa tim. Ovde je takođe predstavljena upotreba `errors.Error()` funkcije koja omogućava da pretvorite `error` varijablu u `string` varijablu. Ova funkcija omogućava poređenje `error` varijable sa `string` varijablom.

Smatra se dobrom praksom da svoje poruke o grešci šaljete u servis za zapisivanje u dnevnik vašeg UNIX uređaja, pogotovo kada je Go program server ili neka druga kritična aplikacija. Međutim, kod u ovoj knjizi neće svuda slediti ovaj princip da biste izbegli punjenje datoteka dnevnika nepotrebnim podacima.

Izvršavanje datoteke `newError.go` će proizvesti sledeći izlaz:

```
$ go run newError.go
returnError() ended normally!
Error in returnError() function!
!!
```

Ako pokušate da uporedite `error` varijablu sa `string` varijablom, a da se prethodno ne konvertuje `error` varijabla u `string` varijablu, Go prevodilac će kreirati sledeću poruku o grešci:

```
# command-line-arguments
./newError.go:33:9: invalid operation: err == "Error in returnError()
function!" (mismatched types error and string)
```

Rukovanje greškama

Rukovanje greškama je veoma važna karakteristika Goa, jer skoro sve Go funkcije vraćaju poruku o grešci ili `nil`, što je način da Go ukaže da li je postojao uslov za grešku dok je izvršavao neku funkciju. Najverovatnije ćete se umoriti od gledanja sledećeg Go koda ne samo u ovoj knjizi, već i u svakom drugom Go programu koji možete pronaći na Internetu:

```
if err != nil {
    fmt.Println(err)
    os.Exit(10)
}
```

Ne brkajte rukovanje greškama sa ispisom greške na izlaz, jer su to dve potpuno različita pojma. Rukovanje greškama ima veze sa Go kodom koji obrađuje uslove greške, dok ispis greške na izlazu mora da piše nešto u standardni deskriptor datoteke greške.

Prethodni kod ispisuje generisanu poruku o grešci na ekran i izlazi iz vašeg programa pomoću funkcije `os.Exit()`. Imajte na umu da takođe možete izaći iz svog programa pozivom ključne reči `return` unutar `main()` funkcije. Generalno govoreći, pozivanje `os.Exit()` iz funkcije koja nije `main()` smatra se lošom praksom. Funkcije koje nisu `main()` imaju tendenciju da vraćaju poruku o grešci pre izlaska, kojom rukuje funkcija poziva.

Ako želite da pošaljete poruku o grešci servisu dnevnika, umesto na ekran, treba da koristite sledeću varijaciju prethodnog Go koda:

```
if err != nil {
    log.Println(err)
    os.Exit(10)
}
```

Postoji još jedna varijanta prethodnog koda koja se koristi kad se dogodi nešto veoma loše i želite da prekinete program:

```
if err != nil {
    panic(err)
    os.Exit(10)
}
```

Panic je ugrađena funkcija Goa koja zaustavlja izvršavanje programa i počinje da „paniči" (izbacuje nizove poruka o greškama, blinka ekranom itd). Ako prečesto koristite `panic`, možda biste hteli da preispitate svoju Go implementaciju. Ljudi nastoje da izbjegavaju panične situacije u korist grešaka kad god je to moguće.

Kao što ćete videti u sledećem poglavlju, Go takođe nudi funkciju `recover`, koja bi mogla da vas spasi od nekih loših situacija. U Poglavlju 2, *Razumevanje Go komponenti*, saznaćete više o moći funkcija `panic` i `recover`.

Sada je vreme da vidite Go program koji ne samo da rukuje porukama o greškama koje generišu standardne Go funkcije, nego definiše vlastitu poruku o grešci. Program se zove `errors.go` i biće vam predstavljen u pet delova. Kao što ćete videti, uslužni program `errors.go` pokušava da poboljša funkcionalnost programa `cla.go`, koji ste videli ranije u ovom poglavlju prilikom ispitivanja da li su argumenti njegove komandne linije prihvatljivi brojevi sa pokretnim zarezom.

Prvi deo programa `errors.go` je sledeći:

```
package main

import (
    "errors"
    "fmt"
    "os"
    "strconv"
)
```

Ovaj deo programa `errors.go` sadrži očekivane import instrukcije.

Drugi deo programa `errors.go` dolazi sa sledećim Go kodom:

```
func main() {
    if len(os.Args) == 1 {
        fmt.Println("Please give one or more floats.")
        os.Exit(1)
    }

    arguments := os.Args
    var err error = errors.New("An error")
    k := 1
    var n float64
```

Ovde kreirate novu `error` varijablu pod nazivom `err` da biste je inicijalizirali sa vašom vrednošću.

Treći deo programa `errors.go` je sledeći:

```
for err != nil {
    if k >= len(arguments) {
    fmt.Println("None of the arguments is a float!")
    return
    }
    n, err = strconv.ParseFloat(arguments[k], 64)
    k++
}

min, max := n, n
```

Ovo je najvarljiviji deo programa, jer, ako prvi argument komandne linije nije odgovarajući broj sa pokretnim zarezom, moraćete da proverite sledeći i da nastavite proveravanje, dok ne pronađete prikladan argument komandne linije. Ako ni jedan od argumenata komandne linije nije ispravnog formata, `errors.go` će se zaustaviti i ispisaće poruku na ekran. To proveravanje se vrši ispitivanjem `error` vrednosti koju vraća `strconv.ParseFloat ()`. Sav ovaj kod je samo radi ispravne inicijalizacije varijabli `min` i `max`.

Četvrti deo programa `errors.go` dolazi sa sledećim Go kodom:

```
    for i := 2; i < len(arguments); i++ {
        n, err := strconv.ParseFloat(arguments[i], 64)
        if err == nil {
        if n < min {
            min = n
        }
        if n > max {
            max = n
        }
    }
}
```

Ovde samo obrađujete sve prave argumente komandne linije da biste među njima pronašli minimalni i maksimalni broj sa pokretnim zarezom.

Konačno, poslednji deo programa se bavi upravo ispisom trenutnih vrednosti `min` i `max` varijabli:

```
        fmt.Println("Min:", min)
        fmt.Println("Max:", max)
    }
```

Kao što možete da vidite iz Go koda datoteke `errors.go`, veći deo njenog koda se odnosi na rukovanje greškama, a ne na stvarnu funkcionalnost programa. Nažalost, to je slučaj sa najviše modernog softvera koji je razvijen u Gou, kao i u većini drugih programskih jezika.

Ako izvršite `errors.go`, dobićete sledeću vrstu izlaza:

```
$ go run errors.go a b c
None of the arguments is a float!
$ go run errors.go b c 1 2 3 c -1 100 -200 a
Min: -200
Max: 100
```

KORIŠĆENJE DOKERA

U poslednjem delu ovog poglavlja ćete naučiti kako da koristite Doker sliku da biste kompajlirali i izvršavali svoj Go kod unutar te slike.

Kao što možda već znate, sve u Dokeru počinje sa Doker slikom. Možete ili da izgradite vlastitu Doker sliku od početka ili ćete započeti sa postojećom? Za namenu u ovom odeljku preuzeli smo osnovnu Doker sliku sa Doker Haba i nastavićemo unutar nje izradu Go verzije `Hello World!` programa.

Sadržaj `Dockerfile` koji će se koristiti je sledeći:

```
FROM golang:alpine

RUN mkdir /files
COPY hw.go /files
WORKDIR /files
RUN go build -o /files/hw hw.go
ENTRYPOINT ["/files/hw"]
```

Prvi red definiše Doker sliku koja će se koristiti. Preostale tri naredbe kreiraju novi direktorijum u Doker slici, kopiraju datoteku (`hw.go`) iz trenutnog direktorijuma korisnika u Doker sliku i menjaju trenutni radni direktorijum Doker slike, respektivno. Poslednje dve naredbe kreiraju binarnu izvršnu datoteku od izvorne Go datoteke i specificiraju putanju binarne datoteke koja će biti izvršena kada pokrenete tu Doker sliku.

Kako koristite taj `Dockerfile`? Pod uslovom da datoteka `hw.go` postoji u trenutnom radnom direktorijumu, novu Doker sliku možete izgraditi na sledeći način:

```
$ docker build -t go_hw:v1 .
Sending build context to Docker daemon  2.237MB
Step 1/6 : FROM golang:alpine
alpine: Pulling from library/golang
cd784148e348: Pull complete
7e273b0dfc44: Pull complete
952c3806fd1a: Pull complete
ee1f873f86f9: Pull complete
7172cd197d12: Pull complete
Digest:
sha256:198cb8c94b9ee6941ce6d58f29aadb855f64600918ce602cdeacb018ad77d647
Status: Downloaded newer image for golang:alpine
 ---> f56365ec0638
Step 2/6 : RUN mkdir /files
 ---> Running in 18fa7784d82c
Removing intermediate container 18fa7784d82c
 ---> 9360e95d7cb4
Step 3/6 : COPY hw.go /files
 ---> 680517bc4aa3
Step 4/6 : WORKDIR /files
 ---> Running in f3f678fcc38d
Removing intermediate container f3f678fcc38d
 ---> 640117aea82f
Step 5/6 : RUN go build -o /files/hw hw.go
 ---> Running in 271cae1fa7f9
Removing intermediate container 271cae1fa7f9
 ---> dc7852b6aeeb
Step 6/6 : ENTRYPOINT ["/files/hw"]
 ---> Running in cdadf286f025
Removing intermediate container cdadf286f025
 ---> 9bec016712c4
Successfully built 9bec016712c4
Successfully tagged go_hw:v1
```

Naziv novokreirane Doker slike je `go_hw: v1`.

Ako je Doker slika `golang:alpine` već prisutna na vašem računaru, izlaz prethodne naredbe će biti sledeći:

```
$ docker build -t go_hw:v1 .
Sending build context to Docker daemon  2.237MB
Step 1/6 : FROM golang:alpine
 ---> f56365ec0638
Step 2/6 : RUN mkdir /files
 ---> Running in 982e6883bb13
Removing intermediate container 982e6883bb13
 ---> 0632577d852c
Step 3/6 : COPY hw.go /files
 ---> 68a0feb2e7dc
Step 4/6 : WORKDIR /files
 ---> Running in d7d4d0c846c2
Removing intermediate container d7d4d0c846c2
 ---> 6597a7cb3882
Step 5/6 : RUN go build -o /files/hw hw.go
 ---> Running in 324400d532e0
Removing intermediate container 324400d532e0
 ---> 5496dd3d09d1
Step 6/6 : ENTRYPOINT ["/files/hw"]
 ---> Running in bbd24840d6d4
    Removing intermediate container bbd24840d6d4
 ---> 5a0d2473aa96
    Successfully built 5a0d2473aa96
    Successfully tagged go_hw:v1
```

Da li na vašoj mašini postoji Doker slika `go_hw: v1` možete proveriti na sledeći način:

```
$ docker images
REPOSITORY    TAG       IMAGE ID      CREATED              SIZE
go_hw         v1        9bec016712c4  About a minute ago   312MB
golang        alpine    f56365ec0638  11 days ago          310MB
```

Sadržaj `hw.go` datoteke je sledeći:

```
package main

import (
    "fmt"
)

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

Doker sliku koja se nalazi na vašem lokalnom računaru možete da koristite na sledeći način:

```
$ docker run go_hw:v1
Hello World!
```

Postoje i drugi složeniji načini izvršavanja Doker slike, ali za tako naivnu Doker sliku ovo je najjednostavniji način da se koristi.

Ako želite, možete da sačuvate (**push**) Doker sliku u Doker registru na Internetu da biste mogli docnije odatle da je povučete (**pull**).

Doker Hab može biti Doker registar, pod uslovom da imate Doker Hab račun, koji je besplatan i koji se može lako kreirati. Nakon kreiranja Doker Hab računa, treba da izvršite sledeće naredbe na UNIX mašini i sačuvate tu sliku na Doker Habu:

```
$ docker login
Authenticating with existing credentials...
Login Succeeded
$ docker tag go_hw:v1 "mactsouk/go_hw:v1"
$ docker push "mactsouk/go_hw:v1"
The push refers to repository [docker.io/mactsouk/go_hw]
bdb6946938e3: Pushed
99e21c42e35d: Pushed
0257968d27b2: Pushed
e121936484eb: Pushed
61b145086eb8: Pushed
789935042c6f: Pushed
b14874cfef59: Pushed
7bff100f35cb: Pushed
v1: digest:
sha256:c179d5d48a51b74b0883e582d53bf861c6884743eb51d9b77855949b5d91dd
e1 size: 1988
```

Prva naredba je potrebna za prijavu na Doker Hab i trebalo bi se izvrši samo jednom. Naredba `docker tag` je potrebna za određivanje naziva lokalne slike na Doker Habu i treba je izvršiti pre naredbe `docker push`. Poslednja naredba šalje željenu Doker sliku na Doker Hab, odakle se generiše bogat izlaz. Ako svoju Doker sliku učinite javnom, svako će je moći da je povuče i koristi.

Jednu ili više Doker slika možete na više načina da izbrišete sa svoje lokalne UNIX mašine.

Jedan od tih načina je upotreba `IMAGE ID` Doker slike:

```
$ docker rmi 5a0d2473aa96 f56365ec0638
Untagged: go_hw:v1
Deleted:
sha256:5a0d2473aa96bcdafbef92751a0e1c1bf146848966c8c971f462eb1eb242d2
a6
Deleted:
```

```
sha256:5496dd3d09d13c63bf7a9ac52b90bb812690cdfd33cfc3340509f9bfe6215c
48
Deleted:
sha256:598c4e474b123eccb84f41620d2568665b88a8f176a21342030917576b9d82
a8
Deleted:
sha256:6597a7cb3882b73855d12111787bd956a9ec3abb11d9915d32f2bba4d0e92e
c6
Deleted:
sha256:68a0feb2e7dc5a139eaa7ca04e54c20e34b7d06df30bcd4934ad6511361f2c
b8
Deleted:
sha256:c04452ea9f45d85a999bdc54b55ca75b6b196320c021d777ec1f766d115aa5
14
Deleted:
sha256:0632577d852c4f9b66c0eff2481ba06c49437e447761d655073eb034fa0ac3
33
Deleted:
sha256:52efd0fa2950c8f3c3e2e44fbc4eb076c92c0f85fff46a07e060f5974c1007
a9
Untagged: golang:alpine
Untagged:
golang@sha256:198cb8c94b9ee6941ce6d58f29aadb855f64600918ce602cdeacb01
8ad77d647
Deleted:
sha256:f56365ec0638b16b752af4bf17e6098f2fda027f8a71886d6849342266cc3a
b7
Deleted:
sha256:d6a4b196ed79e7ff124b547431f77e92dce9650037e76da294b3b3aded709b
dd
Deleted:
sha256:f509ec77b9b2390c745afd76cd8dd86977c86e9ff377d5663b42b664357c35
22
Deleted:
sha256:1ee98fa99e925362ef980e651c5a685ad04cef41dd80df9be59f158cf9e529
51
Deleted:
sha256:78c8e55f8cb4c661582af874153f88c2587a034ee32d21cb57ac1fef51c610
9e
Deleted:
sha256:7bff100f35cb359a368537bb07829b055fe8e0b1cb01085a3a628ae9c187c7
b8
```

Doker je ogromna i zaista važna tema koju ćemo ponovo razmatrati u nekoliko poglavlja ove knjige.

VEŽBE I LINKOVI

- Posetite stranicu Go: https://golang.org/
- Posetite Doker stranicu: https://www.docker.com/.
- Posetite lokaciju Doker Hab: https://hub.docker.com/.
- Go 2 nacrt dizajna: https://blog.golang.org/go2draft.
- Pregledajte veb lokaciju Go dokumentacije: https://golang.org/doc/.
- Posetite Dokumentaciju `log` paketa na https://golang.org/pkg/log/.
- Posetite dokumentaciju `log/syslog` paketa na https://golang.org/pkg/log/syslog/.
- Posetite dokumentaciju `os` paketa na adresi https://golang.org/pkg/os/.
- Pogledajte https://golang.org/cmd/gofmt/, koja sadrži stranicu dokumentacije alata `gofmt` koji se koristi za formatiranje Go koda.
- Napišite Go program koji nalazi sumu svih argumenata komandne linije koji su validni brojevi.
- Napišite Go program koji nalazi prosečnu vrednost brojeva sa pomičnom zarezom koji su dati kao argumenti komandne linije.
- Napišite Go program koji nastavlja da čita cele brojeve dok ne dobije reč `END` kao ulaz.
- Možete li da izmenite `customLog.go` da bi njegovi podaci bili zapisani istovremeno u dve datoteke dnevnika? Ako vam je potrebna pomoć, potražite je u Poglavlju 8, *Naložiti UNIX sistemu šta da radi*.
- Ako radite na Mac računaru, proverite **TextMate** editor na adresi http://macromates.com/ i **BBEdit** na adresi https://www.barebones.com/products/bbedit/.
- Posetite stranicu sa dokumentacijom `fmt` paketa na adresi https://golang.org/pkg/fmt/ da biste saznali više detalja o glagolima i dostupnim funkcijama.
- Posetite https://blog.golang.org/why-generics da biste saznali više detalja o Gou i Genericima.

REZIME

Ovo poglavlje sadrži mnogo zanimljivih tema Goa, uključujući kompajliranje Go koda, korišćenje standardnog ulaza, standardnog izlaza i standardne greške u Go, obradu argumenata komandne linije, ispis na ekran i korišćenje servisa dnevnika UNIX sistema, kao i o rukovanje greškama i neke opšte informacije o Gou. Sve ove teme treba da smatrate osnovnim informacijama o Gou.

Sledeće poglavlje posvećeno je komponentama Goa, što uključuje razmatranje o sakupljaču smeća, Go kompajleru, pozivanju C koda iz Goa, ključnoj reči `defer`, Go asembleru, WebAssemblyju i ugrađenim funkcijama `panic` i `recover`.