Praktični Go

Što i kako nakon "Hello World"?

Igor Rumiha
CEO, CTO, Programer, Veseli Mačak

Hello world?

Za one koji to još nisu napravili:

```
package main

import "fmt"

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

Run

Sleepsort

```
func main() {
    sortSize := 1000
    out := make(chan int32)
    start := make(chan struct{})

    for i := 1; i <= sortSize; i++ {
        newInt := rand.Int31n(int32(i))
        go sleepsort(newInt, start, out)
    }
    close(start)

    collectResults(sortSize, out)
}
```

```
func sleepsort(number int32, start chan struct{}, out chan int32) {
    <-start
    time.Sleep(time.Duration(number) * time.Millisecond)
    out <- number
}
```

Sleepsort

```
func collectResults(sortSize int, rc chan int32) {
    for {
        select {
        case outInt := <-rc:
            fmt.Println(outInt)
            sortSize--
        default:
        }
        if sortSize == 0 {
            break
        }
    }
}
```

Sleepsort

```
func main() {
    sortSize := 1000
    out := make(chan int32)
    start := make(chan struct{})

    for i := 1; i <= sortSize; i++ {
        newInt := rand.Int31n(int32(i))
        go sleepsort(newInt, start, out)
    }
    close(start)

    collectResults(sortSize, out)
}
```

Run

Natrag u stvarni svijet

Instalacija Go okruženja

- golang.org – uvijek najnovije verzije, za Windows platformu jedino rješenje
- paketni sustavi – apt, yum, itd. – kasne i nekoliko verzija, no možete koristiti neslužbene repozitorije
- Gentoo linux, Arch linux, OS X: homebrew – razumno brzo se pojavljuju nove verzije

Go alati

```
/usr/local/go/bin
├── go
├── godoc
└── gofmt
```

```
/usr/local/go/pkg/tool/
└── darwin_amd64
    ├── 6a
    ├── 6c
    ├── 6g
    ├── 6l
    ├── addr2line
    ├── cgo
    ├── cover
    ├── dist
    ├── fix
    ├── nm
    ├── objdump
    ├── pack
    ├── pprof
    ├── tour
    ├── vet
    └── yacc
```

Go alati

Cijeli zoološki vrt pun čudnih imena

- compiler *c (5c, 6c, 8c), *g (5g, 6g, 8g)
- assembler *a (5a, 6a, 8a)
- linker *l (5l, 6l, 8l)

Detalji na golang.org/cmd/ (https://golang.org/cmd/)

Svi važni Go alati objedinjeni su pod "go" alat.

Često korištene "go" naredbe:

- go get
- go build
- go run
- go fmt

GOPATH

- globalni "workspace" za sav Go kod, i vaš i od drugih programera.
- moj GOPATH:

```
/Users/igorrumiha/Work/gopath:/Users/igorrumiha/Work/gopath-local
```

GOPATH

```
/Users/igorrumiha/Work/gopath
├── bin
├── pkg
|   └── darwin_amd64
|       ├── github.com
|       |   ├── ActiveState
|       |   ├── andlabs
...
└── src
    ├── git.eclipse.org
    |   └── gitroot
    |       └── paho
    ├── github.com
    |   ├── ActiveState
    |   |   └── tail
    |   ├── GeertJohan
    |   |   └── gomatrix
    |   ├── andlabs
    |   |   └── ui
...
```

Go package sustav

- zamišljen kao efikasnija zamjena za C/C++ include direktive + linkanje.
- cilj je postignut: prevođenje Go koda je nemjerljivo brže od C i C++ koda.

Importi

```
import (
    "fmt"

    "github.com/irumiha/preza/intro"
)
```

```
func main() {
    fmt.Println(medo.Hey())
    fmt.Println(medo.Hey2())
}
```

Run

Što se desilo?

- u datoteci `$GPOPATH/src/github.com/irumiha/preza/intro/intro.go`:

```
package medo

func Hey() string {
    return "Hej medo 1"
}
```

- nepisano je pravilo da package imenujemo jednako kao i zadnja komponenta u stazi datoteke, ali ništa nas ne sprečava da koristimo različita imena.

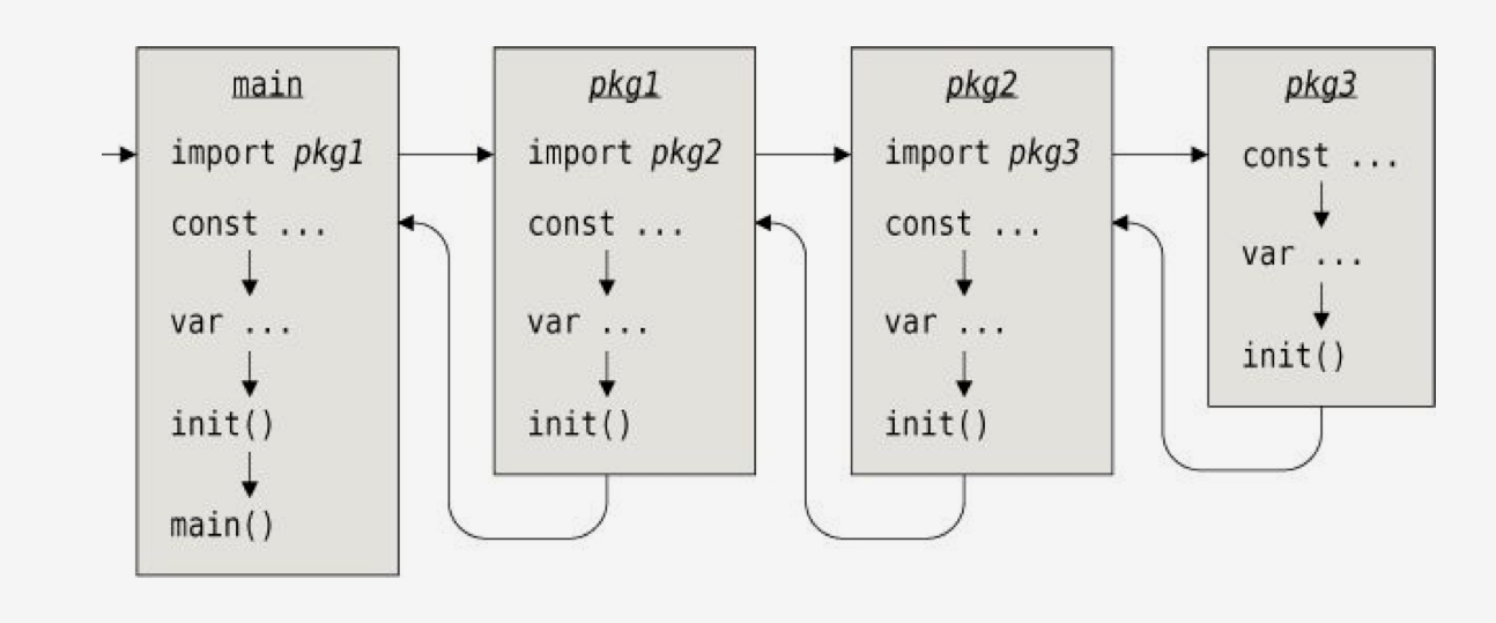
Relativni importi (van GOPATH-a)

```
package main

import (
    "fmt"

    "github.com/irumiha/preza/custom/other"

    "./custom/pkg"
)

func main() {
    fmt.Println(pkg.Hey())
    fmt.Println(other.Other())
}
```

Run

Što se dešava pri pokretanju aplikacije

- import svih paketa iz main
- rekurzivno import svih paketa koje ti paketi importaju



- incijalizacija svih toplevel konstanti i varijabli
- pozivanje init() funkcija
- pozivanje main() funkcije u main paketu

Zgodna primjena init() funkcija

```
// image decoders
import (
    "image"

    _ "image/png"
    _ "image/jpeg"
)
```

```
// database drivers
import (
    "database/sql"

    _ "github.com/lib/pq"
)
```

Kako raditi projekte u $GOPATH

- Go projekti nemaju "uobičajenu" bin, doc, etc, lib, src itd. strukturu
- Go team preporuča (zapravo propisuje) korištenje samo jednog $GOPATH–a na računalu
- preporuča se korištenje nekog domain prefiksa – preporučljivo je da to bude staza za `go get` alat

```
github.com/irumiha/preza
```

Vendoring

- go get i github.com kao najpopularnija kombinacija distribucije Go koda rade super dok radite razvoj.
- što se dešava nakon 2 godine kad poželite ponovno kompajlirati vaš kod?
- API paketa koje ste koristili su se promijenili
- neki paketi više i ne postoje
- rješenje (u nastajanju): vendoring i import path rewriting

Vendoring

- alati nastali u Go zajednici (trenutno popularni su "godep" i "nut")
- smjesti kopije vaših dependencija u poddirektorij vašeg projekta,
- snime se metapodaci kao git revizija ili tag i sl.
- pomažu u podešavanju $GOPATH varijable da se koristi lokalna kopija.

Vendoring

- ako ste se odlučili za **import path rewriting** tada source vaših depencencija postaje "vaš".
- import izgleda ovako:

```
import "github.com/irumiha/preza/_internal/github.com/gorilla/mux"
```

Vaš workflow će morati uključivati povremeno pokretanje vendoring alata koji će osvježiti svoje metapodatke ili napraviti nove kopije koda ako ste dodavali dependencije

Alternativni pristupi build proceduri

- Makefile!
- lokalno kreiranje $GOPATH okruženja
- kakvo god rješenje bilo, uključivat će nekakvi oblik vendoringa.

Go doc

- Go alati uključuju i generator dokumentacije:

```
godoc -http=:6060
```

- rezultat je mini site po dizajnu sličan golang.org, koji sadrži tutoriale i standardnu dokumentaciju ali i dokumentaciju svih paketa koje ste vi napravili ili naknadno instalirali u $GOPATH
- kako dokumentirati objašnjeno je na blog.golang.org/godoc-documenting-go-code (http://blog.golang.org/godoc-documenting-go-code)

Go test

- testovi se tipično pišu u istom packageu u kojem je kod koji se testira
- u datoteku *_test.go smjeste se test funkcije čije ime mora počinjati sa Test i moraju primati samo jedan argument tipa `t *testing.T`
- go test alat se stalno razvija (u verziji 1.4 dobio je coverage feature)
- go test ima mnogo opcija, istražite ih!

Go test

```
package sessions

import (
    "net/http"
    "testing"
)

// Test for GH-8 for CookieStore
func TestGH8CookieStore(t *testing.T) {
    originalPath := "/"
    store := NewCookieStore()
    store.Options.Path = originalPath
    req, err := http.NewRequest("GET", "http://www.example.com", nil)
    if err != nil {
        t.Fatal("failed to create request", err)
    }
    store.Options.Path = "/foo"
    if session.Options.Path != originalPath {
        t.Fatalf("bad session path: got %q, want %q", session.Options.Path, originalPath)
    }
}
```

Ostalo: error vrijednosti

Exceptioni – zašto

- "elegantan" postupak za tretiranje grešaka
- svi smo naučeni na exceptione iz ostalih mainstream jezika

Exceptioni – zašto ne

- funkcije prestaju biti referencijalno transparentne – vrlo važan koncept ako se želite stilom programiranja približiti funkcionalnom programiranju
- pojam "referentially transparent" znači da funkcija uz isti ulaz uvijek daje isti izlaz
- funkcija koja "baca" exception ne samo da povremeno vraća različite rezultate već ponekad vraća rezultate različitog tipa!
- nisu type-safe

errors – osnove

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

```
// Copyright 2011 The Go Authors.  All rights reserved.
// Use of this source code is governed by a BSD-style
// license that can be found in the LICENSE file.

// Package errors implements functions to manipulate errors.
package errors

// New returns an error that formats as the given text.
func New(text string) error {
    return &errorString{text}
}

// errorString is a trivial implementation of error.
type errorString struct {
    s string
}

func (e *errorString) Error() string {
    return e.s
}
```

panic/recover

- mehanizam koji Go-u daje exception-like ponašanje

```
package main

import "fmt"
import "math"

func ConvertInt64ToInt(x int64) int {
    if math.MinInt32 <= x && x <= math.MaxInt32 {
        return int(x)
    }
    panic(fmt.Sprintf("%s is out of range"))
}

func IntFromInt64(x int64) (i int, err error) {
    defer func() {
        if e := recover(); e != nil {
            err = fmt.Errorf("%v", e)
        }
    }()
    return ConvertInt64ToInt(x), nil
}
```

errors – lista za čitanje

- blog.golang.org/error-handling-and-go (http://blog.golang.org/error-handling-and-go)
- justinas.org/best-practices-for-errors-in-go/ (https://justinas.org/best-practices-for-errors-in-go/)
- groups.google.com/forum/#!topic/golang-nuts/HOXNBQu5c-Q%5B1-25%5D

  (https://groups.google.com/forum/#!topic/golang-nuts/HOXNBQu5c-Q%5B1-25%5D)

Go iznenađenja

- go call by value semantika
- range operator stvara kopije objekata
- multicore: bez dodatnih opcija ili koda vaš Go program neće koristiti više od jedne CPU jezgre

Go pozitivna iznenađenja

Interfaces.
Može ih se koristiti na 100 načina:

- elegantno "spajanje" funkcionalnosti vašeg programa
- dependency injection
- testiranje (mocking funkcionalnost)

Potreba za resursima (memorija, CPU).

Gušt u programiranju :)

Thank you

Igor Rumiha
CEO, CTO, Programer, Veseli Mačak
igor-rumiha@veseli-macak.net (mailto:igor-rumiha@veseli-macak.net)
@irrummi (http://twitter.com/irrummi)