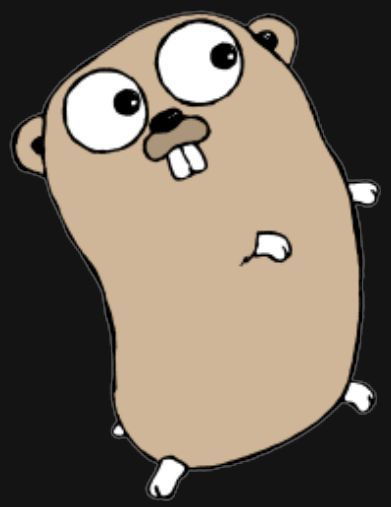
Go

eine moderne Programmiersprache



Michael „sECuRE“ Stapelberg

2012-06-09, GPN12

powered by LuaLaTeX

Inhalt

- Einleitung
- Sourcecode
- Mehr Sourcecode
- Projekte, Kultur, Kurioses (?)
- Fragen

Einleitung

- Programmiersprache, welche die Effizienz kompilierter Sprachen mit der Leichtigkeit dynamischer Sprachen vereinen will
- Besonders gute Unterstützung für Concurrency
- Schnelles Kompilieren, keine Makefiles
- Garbage Collection
- Unicode-Unterstützung, Arrays/Maps, HTTP/JSON/Crypto/… in der stdlib

Hello World

```
package main

import "fmt"

func main() {
    fmt.Println("Hallo GPN ♥, enjoy…")
}
```

Variablen/Typen

```
// Äquivalent (wegen type inference)
var foo string = "ohai"
foo := "ohai"
```

Variablen/Typen

```
// Äquivalent (wegen type inference)
var foo string = "ohai"
foo := "ohai"

// Typen
byte, (u)int{8,16,32,64}, float{32,64}
```

Variablen/Typen

```
// Äquivalent (wegen type inference)
var foo string = "ohai"
foo := "ohai"

// Typen
byte, (u)int{8,16,32,64}, float{32,64}

// Array
weekdays := [2]string{"Mo", "Di"}
```

Variablen/Typen

```
// Äquivalent (wegen type inference)
var foo string = "ohai"
foo := "ohai"

// Typen
byte, (u)int{8,16,32,64}, float{32,64}

// Array
weekdays := [2]string{"Mo", "Di"}

// Map
klausurpunkte := make(map[string]int)
klausurpunkte["Michael Stapelberg"] = 0
klausurpunkte["Sven Schönung"] = 15
```

Schleifen

```
weekdays := [...]string{"Mo", "Di"}

for index, value := range(weekdays) {
    fmt.Printf("Der %d. Wochentag ist %s\n",
        index, value)
}
```

Schleifen

```
weekdays := [...]string{"Mo", "Di"}

for index, value := range(weekdays) {
    fmt.Printf("Der %d. Wochentag ist %s\n",
        index, value)
}

for _, value := range(weekdays) {
    fmt.Printf("%s ist ein Wochentag\n",
       value)
}
```

Schleifen

```
weekdays := [...]string{"Mo", "Di"}

for index, value := range(weekdays) {
    fmt.Printf("Der %d. Wochentag ist %s\n",
        index, value)
}

for _, value := range(weekdays) {
    fmt.Printf("%s ist ein Wochentag\n",
       value)
}

for {
    fmt.Println("Der neue IBM schafft " +
        "Endlosschleifen in 5 Stunden!")
}
```

Fehlerbehandlung

```
// Fehler prüfen und reagieren
file, err := os.Open("funnycat.jpg")
if err != nil {
    fmt.Printf("Cannot open: %v\n", err)
    os.Exit(1)
}

// Fehler ignorieren (gelegentlich sinnvoll)
file, _ := os.Open("funnycat.jpg")
```

Typen (Deklaration)

```
type SizeIndex struct {
    filename string
    Index map[string]int64
}
```

Typen (Deklaration)

```
type SizeIndex struct {
    filename string
    Index map[string]int64
}

var idx SizeIndex
idx.Index = make(map[string]int64)
idx.Index["/home/michael/go.pdf"] = 934821
```

Typen (Methoden)

```
func (i *SizeIndex) Save() error {
    file, err := os.Open(i.filename)
    if err != nil {
        return err
    }
    defer file.Close()

    encoder := gob.NewEncoder(file)
    if err := encoder.Encode(i); err != nil {
        return err
    }

    return nil
}
```

Goroutinen

```
func Ready(what string, dur time.Duration) {
    time.Sleep(dur)
    fmt.Printf("%s is ready!\n", what)
}
```

Goroutinen

```
func Ready(what string, dur time.Duration) {
    time.Sleep(dur)
    fmt.Printf("%s is ready!\n", what)
}

func main() {
    go Ready("tea", 6 * time.Second)
    go Ready("coffee", 2 * time.Second)
    fmt.Println("waiting")
    time.Sleep(10 * time.Second)
}
```

Channels

```
func cacheFlusher(cacheChan chan string) {
  for {
    select {
      case url := <-cacheChan:
      fmt.Printf("read URL %s\n", url)
      case <-time.After(10 * time.Second):
      fmt.Println("writing cache to disk")
    }
  }
}

cacheChan := make(chan string)
go cacheFlusher(cacheChan)
for {
  fmt.Scanf("%s", &input)
  cacheChan <- input
}
```

Beispiel: Website-Download mit Timeout

```
result := make(chan *http.Response, 1)

go func() {
    resp, _ := http.Get("http://gulas.ch/")
    result <- resp
}()

go func() {
    time.Sleep(5 * time.Second)
    result <- nil
}()

if resp := <-result; resp != nil {
    fmt.Printf("HTTP %d\n", resp.StatusCode)
}
```

Interfaces

```
type Shaper interface {
    Area() int
}

type Rectangle struct {
    length, width int
}

func (r Rectangle) Area() int {
    return r.length * r.width
}

r := Rectangle{length:5, width:3}
s := Shaper(r)
fmt.Println("area: ", s.Area())
```

Bekannte Interfaces, Komposition

- `io.Writer`:
  `Write(p []byte) (n int, err error)`
- `hash.Hash`:
  `Sum(b []byte) []byte`
  …
- `hash/crc32` implementiert `io.Writer` und `hash.Hash`

Vererbung

```
type Notebook struct {
    model string
}

func (nb Notebook) Describe() {
    fmt.Println(nb.model)
}
```

Vererbung

```
type Notebook struct {
    model string
}

func (nb Notebook) Describe() {
    fmt.Println(nb.model)
}

type ThinkPad struct {
    Notebook
}

var t ThinkPad
t.model = "X200"
t.Describe()
```

Slices

```
days := [7]string{"Mo", "Di", "Mi", "Do",
                  "Fr", "Sa", "So"}

// This GPN day is Friday
gpnDay := 4

// "Nerd planning"
window := days[gpnDay:gpnDay+2]

fmt.Printf("It is %s ALREADY?\n", window[0])
fmt.Printf("Tomorrow is %s\n", window[1])
```

Slices (Beispiel: Strings)

```
func main() {
    http.HandleFunc("/follow/", Follow)
}

func Follow(response http.ResponseWriter,
   req *http.Request) {
    // Extract name from URL
    name := req.URL.Path[len("/follow/"):]

    // Send response
    fmt.Fprintf(response, "Following %s",
       name)
}
```

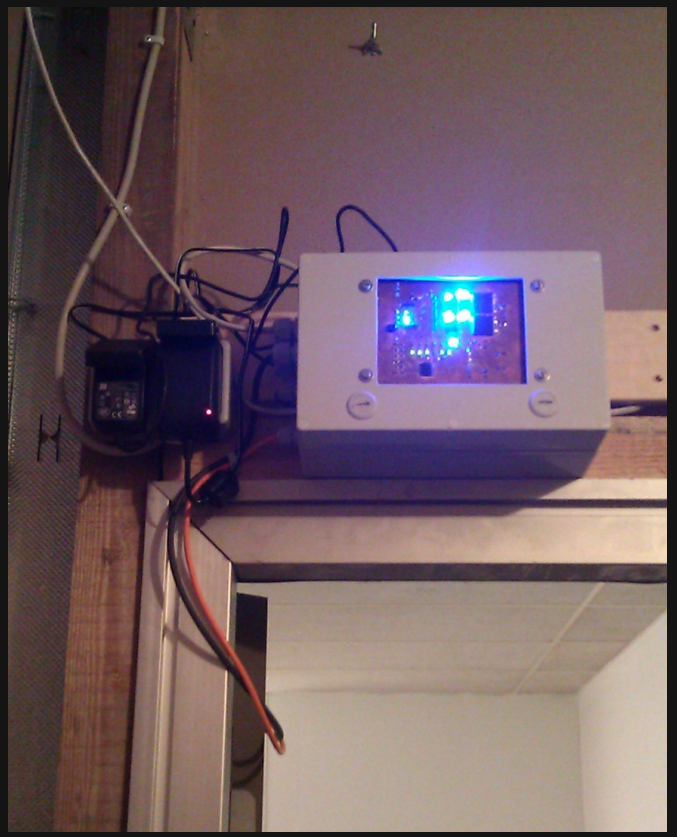
Tests

```
import "testing"

// func TestXXXX(t *testing.T)
func TestKeypresses(t *testing.T) {
    fe := testfrontend.NewTestFrontend()
    fe.FillBuffer("^PAD 2  $")
    select {
        case <-time.After(1 * time.Second):
        // Test failed
        t.Error("Ran into timeout")
        return
        case <-fe.Keypresses:
        return
    }
}
```

Projekt: Pinpad-Controller

- Raspberry Pi + Pinpad + Hometec
- performant (ggü Scriptsprache)
- keine Abhängigkeiten
- Gleichzeitigkeit, Netzwerk, Daemon
- `http://code.stapelberg.de/git/pinpad-controller/`



Projekt: Backups

- Backup-Software für Festplatte $\rightarrow$ Festplatte
- Maximale Auslastung (Lesen/Schreiben gleichzeitig)
- Checksumming, Scrubbing
- `http://code.stapelberg.de/git/rfb/`

Projekt: IRC-Bot für #i3

- HTTP-Endpunkt, zu dem Buildbot Status pushed (JSON)
- Posted zu URLs den HTML `<title>`
- Posted zu `>userguide` die URL `http://i3wm.org/docs/userguide.html` etc.
- `http://code.stapelberg.de/git/go-buildbot-announce/`

Projekt: Google App Engine

- Unterstützung größtenteils gut
- Manche APIs eher hässlich umgesetzt
- Go degradiert zur Syntax

Kultur/Eigenheiten von Go

- einheitliche Formatierung (gofmt)
- statisch gelinkte Binaries
- Packages und Code-Hosting
- Tests
- i386 keine Priorität

Demo Tools

- Demo!

Quellen/Links

- http://en.wikipedia.org/wiki/Go_(programming_language)
- http://www.golang.org/
- http://tour.golang.org/ bzw. http://play.golang.org/
- http://go-book.appspot.com/