Die Programmiersprache Go

http://golang.org

Robert Clausecker

25. Mai 2013

Inhalt

1 Go – Was ist das?

2 Sprachdesign
- Überblick
- Syntax

3 Umgebung
- Bibliotheken
- Werkzeuge

Entstehung

- entstanden als Nebenprojekt der Autoren bei Google
- erster Entwurf 2007 von Robert Griesemer, Ken Thompson und Rob Pike
- Go 1 erschien am 28. März 2012
- gegenwärtig: Go 1.1 vom 14. Mai 2013

Warum Go?

- alle wichtigen Programmiersprachen sind mehr als 10 Jahre alt
- aber die Anforderungen haben sich geändert
  - Parallelität
  - Netzwerkprogrammierung
  - komplexe Frameworks
- Softwareentwicklung ist anstrengend und langsam

Warum Go?

Go ist

- einfach
- parallel
- universell
- schnell
- flexibel
- langweilig

Inhalt

1 Go – Was ist das?

2 Sprachdesign
- Überblick
- Syntax

3 Umgebung
- Bibliotheken
- Werkzeuge

```
package main

import "fmt"

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

> *Consensus drove the design. Nothing went into the language until [Ken Thompson, Robert Griesemer, and myself] all agreed that it was right. Some features didnt get resolved until after a year or more of discussion.*
>
> — *Rob Pike*

- Funktionalität ausgewählt nach Nutzen für den Programmierer
- kein schlampiges Programmieren fördern
- Orthogonalität aller Sprachbestandteile
- mache nur eine Sache und mache sie gut

Daher:

- Zeiger, aber keine Zeigerarithmetik
- Objekte, aber weder Klassen noch Vererbung
- Parallelität als Kernbestandteil der Sprache
- automatische Speicherverwaltung

Zeiger

- Zeiger geben Kontrolle über die Ausrichtung von Datenstrukturen
- Speicherschutz durch fehlende Zeigerarithmetik

```
type Point struct {
    X, Y int
}

var imageA [][]Point
var imageB [][]*Point
```

Objekte

- klassische Objektorientierung ist unflexibel
- existierende Klassen zu korrigieren ist aufwendig
- Schnittstellen bieten einen flexiblen Ansatz
- Schnittstellen werden implizit implementiert

```
type Stringer interface {
    String() string
}

// *Point implementiert Stringer
func (p *Point) String() string {
    return fmt.Sprint(p.X, p.Y)
}
```

Syntax soll . . .

- intuitiv sein
- sich kurz fassen
- einfach analysierbar sein
- nicht vom Code ablenken

Syntax soll . . .

- intuitiv sein
- sich kurz fassen
- einfach analysierbar sein
- nicht vom Code ablenken

Die Syntax von Go . . .

- ist einfach (z. B. nur 25 Schlüsselwörter)
- hat wenige, mächtige Kontrollstrukturen
- ist leicht analysierbar
- wird automatisch formatiert

Typen

```
var a, b int
var c rune = '~'      // UTF-8
var buf []byte        // slice von bytes
var chan1 chan Point // channel von Point
var chan2 chan int    // channel von int
// map von uint32 auf string
var dict map[uint32]string

type foo int // foo ist unterscheidbar von int
n := int(c)  // Deklaration mit Zuweisung
primes := [5]uint{2, 3, 5, 7, 11}
isZero := func(x int) bool { return x == 0 }
```

Parallelität

```
c1, c2, c3 := make(chan int), make(chan int), make(chan int)

throw := func(a <-chan int, b chan<- int, s string) {
    for x := range a {
        fmt.Printf("%s gibt den Ball %d ab.\n", s, x)
        b <- x
    }
    fmt.Printf("%s ist fertig.\n",s)
}

go throw(c1, c2, "Alice") ; go throw(c2, c3, "Bob")

for i := 0; i < 100; i++ {
    c1 <- i     // Ball werfen
    c1 <- <-c3  // auffangen und werfen
    <-c3        // nochmal werfen
}

close(c1); close(c2); close(c3)
```

Komplettes Programm

```
package main

import ( "os" ; "image/png" ; "log" )
import "github.com/fuzxxl/ppm"

func main() {
    image, err := ppm.Decode(os.Stdin)
    if err != nil { log.Fatal(err) }

    err = png.Encode(os.Stdout, image)
    if err != nil { log.Fatal(err) }
}
```

Inhalt

1 Go – Was ist das?

2 Sprachdesign
- Überblick
- Syntax

3 Umgebung
- Bibliotheken
- Werkzeuge

Standardbibliothek

Universelle Werkzeuge zur Lösung häufiger Aufgaben

- Betriebssystemabstraktion
- vollständig Unicode-kompatibel
- Bildverarbeitung
- Kryptographie
- Webentwicklung (HTML-Templatesystem, CGI, Webserver)
- generische Datenbankanbindung
- Serialisierung (JSON, XML)
- vieles mehr

http://golang.org/pkg

weitere Bibliotheken

Anbindungen an

- GTK, QT, wxWidgets, curses, X11, Windows GUI
- MySQL, MongoDB, PostgreSQL, SQLite
- OpenGL, SDL
- OpenAL, PulseAudio
- GStreamer, ffmpeg, libVLC
- JavaScript, Python, Lua, Perl
- vieles mehr

http://code.google.com/p/go-wiki/wiki/Projects http://go-lang.cat-v.org/library-bindings

Eigene Bibliotheken

Bibliotheken erstellen ist einfach

- Alle Dateien in einem Verzeichnis bilden ein Paket
- Großgeschriebene Symbole werden exportiert
- Importpfad ist URL unter der die Bibliothek zu finden ist `code.google.com/p/gosqlite/sqlite`
- Plattformabhängiger Code wird in extra Dateien abgelegt
- `go build` baut Code vollautomatisch *ohne Konfiguration*
- Jedes Projekt kann eigenen `$GOPATH` haben

Compiler

Zwei Implementationen

- gc von Ken Thomposon, basierend auf Plan 9-Konventionen
- gccgo als Frontend für gcc

Werkzeug go abstrahiert den Compiler

- `go help` für Details
- `apt-get install golang`

Dokumentation

godoc erzeugt automatisch Dokumentation

- Wahlweise HTML oder Plaintext
- godoc kann als Webserver laufen
- Dokumentation wird aus Kommentaren erstellt
- kein besonderes Format für Kommentare
- `func ExampleFoo` ist ein Beispiel, dass in der Doku ausgewiesen wird

Tests und Benchmarks

`go test` führt Unittests aus

- `TestFoo` ist ein Test, `BenchBar` ein Benchmark
- Tests können flexibel und automatisch bei jeder Kompilation ausgeführt werden
- Mehr dazu im Paket `testing` http://golang.org/pkg/testing

weitere Werkzeuge

- `go fmt` formatiert Quelltext
- `cgo` linkt gegen C-Code
- `go fix` korrigiert veralteten Code
- `go vet` funktioniert wie lint
- `go get` lädt Pakete und Abhängigkeiten aus dem Internet

Weiterführendes

- http://golang.org
- http://code.google.com/p/go-wiki
- http://fuz.su/~fuz/go