Programmieren in Go

OpenRheinRuhr 2013, Oberhausen
9. November 2013

Harald Weidner
hweidner@gmx.net

Über den Referenten

Historie

- Basic seit 1984 (C 64)
- C seit 1986 (Atari ST)
- Pascal, Lisp, Prolog, Smalltalk im Studium
- C++ seit 1993 (Linux)
- Perl seit 1994 (Internet)
- Oberflächliche Kontakte mit Java, PHP, Ruby, Python

Suche nach einer Sprache

- Schnell wie C/C++
- Bequem und sicher wie Perl
- Für kleine und große Projekte
- Als Unterrichtssprache geeignet
- Freie Software

Informationen zum Vortrag

http://www.hweidner.de/golang

Die Programmiersprache Go

„Go, otherwise known as Golang, is an open source, compiled, garbage-collected, concurrent system programming language.“
[http://en.wikipedia.org/wiki/Go_(programming_language)]

„It’s a fast, statically typed, compiled language that feels like a dynamically typed, interpreted language.“
[http://golang.org/doc]

Go

- Entwickelt seit 2007
  - Ideen in 45-minütigen Compilier-Kaffeepausen
- Veröffentlicht 2009, Go 1.0 2012 (aktuell 1.1.2)
- Ursprünglich 20%-Projekt bei Google
  - Ken Thompson (B, Multics, Unix, Plan 9, UTF-8)
  - Rob Pike (Plan 9, UTF-8)
  - Robert Griesemer (ETH Zürich, Java HotSpot VM)
- BSD-ähnliche Lizenz

Let's Go

- Go Homepage http://golang.org/
- Tutorial http://tour.golang.org/
- Playground http://play.golang.org/
- Golang Book http://golang-book.com/
- Learning Go http://www.miek.nl/files/go/

- Artikel über Ziele und Designentscheidungen
  http://talks.golang.org/2012/splash.article

Go Compiler

Go Frontend für GCC

- Debian-Paket: gccgo
- Dynamisch gelinkte Binaries
- Viele Plattformen, u.a. i386, amd64, arm, mips, ia64, s390, ppc, ...
- Derzeit bessere Performance der Programme

Gc von Google

- Debian-Paket: golang
- Statisch gelinkte Binaries
- Verfügbar für i386, amd64, arm
- Linux, FreeBSD, Windows, Apple
- Derzeit der schnellere Compiler

Go CLI Tool

go build	Package compilieren
go clean	Compilatdateien löschen
go doc	Dokumentation aus Quelltext extrahieren
go env	Für Go relevantes Environment anzeigen
go fix	Quelltext-Reparaturen ausführen
go fmt	Quelltext formatieren
go get	Pakete herunterladen
go install	Heruntergeladene Pakete installieren
go list	Pakete anzeigen
go run	Programm compilieren und ausführen
go test	Unit Tests ausführen
go tool	Tool aus der Go Suite ausführen
go version	Version anzeigen
go vet	Probleme im Quelltext suchen

Entwicklungsumgebungen

- GoClipse (auf Basis von Eclipse)
  http://code.google.com/p/goclipse/
- GolangIDE (auf Basis von LiteIDE)
  http://code.google.com/p/golangide/
- Go-IDE (auf Basis von IntelliJ IDEA)
  http://go-ide.com/

- Syntax Highlighting für div. Editoren
  - Emacs, Vi, Jedit, Geany, Notepad++

Hello World

```
// hello.go

package main

import "fmt"

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

```
$ go run hello.go
Hello, World!
```

```
$ go build hello.go
$ ./hello
Hello, World!
```

Packages (Module)

- Mindestens ein Package „main“
  - 1..n Sourcefiles pro Package
- Globale Bezeichner beginnen mit Großbuchstaben
  - Gilt für **alles**: Variablen, Konstanten, Typen, Interfaces, Funktionen, Methoden, struct-Elemente
  - Alles andere ist nicht außerhalb des Package sichtbar
- Import mit vollem Pfad, Benutzung über Name
- Es werden nur die benötigten Packages importiert
  - Import ungenutzter Module ist ein Fehler!

Variablen und Konstanten (1)

```
// Beispiele für Deklarationen

var a int
var b int8 = 1
var c = int64(2)

// nur in Funktionen erlaubt:
d := 3  // entspricht: var d int = 3

// Konstanten

const Pi float64 = 3.14159265358979323846
const zero = 0.0
const x, y, z = 5, 17.3, "foo"
```

Eingebaute (skalare) Datentypen

bool
int int8 int16 int32 int64
uint uint8 uint16 uint32 uint64
float32 float64
complex64 complex128
byte // entspricht int8
rune // entspricht int32
string
uintptr
error

Eingebaute Konstanten

nil true false iota

Variablen und Konstanten (2)

```
const (
        Sonntag = iota
        Montag
        Dienstag
        Mittwoch
        Donnerstag
        Freitag
        Samstag
        anzahlTage        // nicht extern sichtbar
)
```

- Gruppierung durch Klammern
- iota ist ein Zähler bei const-Definitionen
- Wiederholung von „= iota“

Zeiger

```
var a int
var b *int   // Zeiger auf int

a = 15
b = &a     // b ist ein Zeiger auf a
*b = 20    // a ist nun 20
```

```
func foo() {
      x := new(int)   // Zeiger auf anonymes int
      *x = 23
}  // int wird durch Garbage Collector gelöscht
```

- Zeiger, aber keine Zeigerarithmetik
- Zeiger auf anonyme Variablen mit new()
- Garbage Collection (kein delete)

Funktionen (1)

```
// Funktion mit Parametern
// und Rückgabewert
func mult(a, b int64) int64 {
	return a * b
}

// benannter Rückgabewert
func add(a, b int64) (c int64) {
	c = a + b
	return
}

func  main() {
	x := mult(3, 8)
	y := add(x, 10)
	fmt.Println(x, y)
}
```

```
// anonyme Funktion

func main() {
	k := func (i, j int) int {
			return i * j
		} (14, 9)
}
```

Funktionen (2)

```
// mehrere Rückgabewerte

func foo() (int, error) {
        x := ...
        return x, nil
}

func main() {
        d, err := foo()
        if err != nil { panic(err) }
        …
}
```

```
// Variadische Funktionen

func summe(s ...float64) float64 {
        // arbeitet mit [ ]s
}

x := summe(1.5, 3, -4.5)

// auch erlaubt:
a := [ ]float64{1.5, 3, -4.5}
y := summe(a...)
```

```
// Verzögerte Ausführung

func foo() {
        lock(l)
        defer unlock(l)
        …
        // unlock geschieht hier
}
```

Kontrollstrukturen

```
// for-Schleife

summe := 0
for i := 1; i <= 10; i++ {
        summe += i
}

// while Schleife

for ; i <= 10; { … }
// oder
for i <= 10 { … }

// Endlosschleife
for { … }
```

```
// if-Abfrage

if a < 0 { a = -a }

// If-else

if x := foo(); x < A {
        return A
} else if x > B {
        return B
} else {
        return x
}
```

```
// switch-case

switch m {
        default: foo()
        case 0, 2, 4, 6: bar()
        case 1, 3, 5, 7: biz()
}

// Switch ohne Variable

switch {
        case a < b: foo()
        case a > c: bar()
        case b == d: biz()
}
```

Verbundtypen

```
type Person struct {
	ID int
	Vorname string
	Nachname string
}

a := Person{1, "Harald", "Weidner"}

b := Person{                    // ID bleibt 0
	Vorname: "Erika",
	Nachname: "Mustermann",
}

fmt.Println(a, b)

// ergibt:
// {1 Harald Weidner} {0 Erika Mustermann}
```

```
// Methoden

func (p Person) FullName() string {
	return p.Vorname + " " + p.Nachname
}

fmt.Println(a.FullName())
// ergibt: Harald Weidner
```

```
// Magische Methoden ;-)

func (p Person) String() string {
	return fmt.Sprintf("[%d] %v %v",
			p.ID, p.Vorname,
			p.Nachname)
}

fmt.Println(b)
// ergibt:
// [0] Erika Mustermann
```

Interfaces

- Liste von Methoden, die ein Typ besitzen muss
- Erfüllung wird implizit ermittelt (kein „implements“)
- Polymorphie über Interfaces

```
type FullNamer interface {
        FullName() string
}

var x, y FullNamer

x = Person{23, "Hagbard", "Celine"}  // erlaubt
y = int(3)  // Fehler, int erfüllt interface nicht
```

- Erfüllung über Reflection prüfbar
- Interfaces mit einer Methode heißen Methode + „er“
- Das leere Interface interface{} wird von allen Typen erfüllt

Inklusion

```
type A struct {
    a1 int
    a2 int
}

type B struct {
    A        // namenlose Einbindung
    b1 int
    b2 int
}

func main() {
    var b B

    b.a1 = 1   // entspricht b.A.a1
    b.a2 = 2   // entspricht b.A.a2
    b.b1 = 3
    b.b2 = 4

    fmt.Println(b)
}
```

- Einbindung eines Objektes in ein anderes
- Magischer Punkt-Operator bei Eindeutigkeit
- Ersatz für Vererbung / Mehrfachvererbung

Array und Slice

```
var a [5]int64                  // Array mit 5 Feldern
var b = [3]int{1, 2, 3}         // entspricht [1, 2, 3]
var c = [...]int{1, 7, 4, 8, 3} // automatische Längenerkennung

var s [ ]int                    // uninitialisierter Slice
var t = make([ ]int32, 5)       // Slice der Länge 5
var r = make([ ]int32, 5, 10)   // Slice der Länge 5 und Kapazität 10

s = c[1:4]                      // s ist [7, 4, 8]

r = append(r, 9)  // Anhängen eines Elementes, len(r) ist jetzt 6
t = append(t, 1)  // Kapazitätserweiterung nötig, t wird umkopiert
```

- Arrays sind statisch (Inhalte änderbar)
- Slices sind dynamisch
  - Umkopieren bei Kapazitätsüberschreitung

Map

```
var Size map[string]float32

Size["Alice"] = 1.67
Size["Bob"] = 1.82


age := map[Person]uint {
	Person{1, "Harald", "Weidner"}: 43,
	Person{2, "Erika", "Mustermann"}: 29,
}

for k, v := range(age) {
	fmt.Println(k, „=>“, v)
}
```

- Assoziatives Array
- Schlüsseltyp muss Vergleiche unterstützen
- range() iteriert durch Map / Array / Slice

Goroutine

```
import (
	"fmt"
	"time"
)

func foo() {
	fmt.Println("foo")
}

func bar() {
	fmt.Println("bar")
}

func main() {
	go foo()
	go bar()
	time.Sleep(time.Second)
}
```

- Goroutinen sind nebenläufige Funktionen
- Terminieren
  - am Ende der Funktion
  - bei Ende des Hauptprogrammes
- Nicht zwingend parallel!

```
import "runtime"

func main() {
	runtime.GOMAXPROCS(runtime.NumCPU())
	…
}
```

Channel (1)

```
func fib(chan c) {
	var x, y int = 0, 1
	for {
		c <- x
		x, y = y, x+y
	}
}

func main() {
	c := make(chan int)
	go fib(c)
	for i:=0; i<25; i++ {
		fmt.Println(<- c)
	}
}
```

- Typisierter Transportkanal
- Kann bidirektional genutzt werden
- Kann von allen Funktionen / Goroutinen verwendet werden, die ihn kennen
- Hier: ungepuffert und damit blockierend

Channel (2)

```
// gepufferter Channel

c := make(chan int, 5)

// Prüfung auf geschlossenen Channel

val, ok := <- c
// ok == false wenn c geschlossen und leer

// Multiplexing

var c1, c2 chan int
var i1, i2 int
select {
        case i1 = <- c1: { … }
        case i2 = <- c2: { … }
        default: { … }  // macht die Schleife nichtblockierend
}
```

Multiplexing

```
package main

import (
	"fmt"
	"time"
)

func fib(c chan int) {
	var x, y int = 0, 1
	for {
		select {
			case c <- x: {
				x, y = y, x + y
			}
			case <- c: {
				fmt.Println("fib() has ended")
				return
			}
		}
	}
}

// Fortsetzung in nächster Spalte
```

```
func main() {
	c := make(chan int)
	go fib(c)
	for i:=0; i<20; i++ {
		fmt.Println(<- c)
	}
	c <- 0
	time.Sleep(time.Second)
}
```

- Goroutine
  - rechnet, wenn sie schreiben kann
  - terminiert kontrolliert, wenn sie lesen kann
  - blockiert sonst

Reflection (1)

```
// Type switch

func foo(t interface{}) {
	switch t := t.(type) {
		default:
			fmt.Printf("unexpected type %T\n", t)
		case bool:
			fmt.Printf("boolean %t\n", t)
		case int:
			fmt.Printf("integer %d\n", t)
		case *bool:
			fmt.Printf("pointer to boolean %t\n", *t)
		case *int:
			fmt.Printf("pointer to integer %d\n", *t)
	}
}
```

Reflection (2)

```
// explizite Interface-Prüfung

type Stringer interface {
        String() string
}

func foo(a interface{}) {
        if x, ok := a.(Stringer); ok {
                fmt.Println("Methode String() vorhanden")
                fmt.Println(x.String())
        } else {
                fmt.Println("Methode String() nicht vorhanden")
        }
}
```

- Weitere Möglichkeiten über Package reflect

Standardbibliothek (Auszug)

bufio bytes errors flag fmt reflect regexp sort strconv strings time unsafe

archive	tar zip
compress	bzip2 flate gzip lzw zlib
container	heap list ring
crypto	aes cipher des dsa ecdsa elliptic md5 rand rc4 rsa sha1 sha256 sha512 tls x509
database	sql
encoding	asn1 base64 binary csv hex json pem xml
hash	alder32 crc32 crc64 fnv
html	template
image	color draw gif jpeg png
io	util
log	syslog
math	big cmplx rand
mime	multipart
net	http cgi cookiejar fcgi httptest httputil mail rpc jsonrpc smtp textproto url
os	exec signal user
path	filepath
runtime	cgo debug pprof race
sync	atomic
text	scanner tabwriter template parse
unicode	utf8 utf16

Formatierte Ein-/Ausgabe

```
package fmt

func Fprint(w io.Writer, a ...interface{}) (n int, err error)
func Fprintf(w io.Writer, format string, a ...interface{}) (n int, err error)
func Fprintln(w io.Writer, a ...interface{}) (n int, err error)
func Fscan(r io.Reader, a ...interface{}) (n int, err error)
func Fscanf(r io.Reader, format string, a ...interface{}) (n int, err error)
func Fscanln(r io.Reader, a ...interface{}) (n int, err error)
func Print(a ...interface{}) (n int, err error)
func Printf(format string, a ...interface{}) (n int, err error)
func Println(a ...interface{}) (n int, err error)
func Scan(a ...interface{}) (n int, err error)
func Scanf(format string, a ...interface{}) (n int, err error)
func Scanln(a ...interface{}) (n int, err error)
func Sprint(a ...interface{}) string
func Sprintf(format string, a ...interface{}) string
func Sprintln(a ...interface{}) string
func Sscan(str string, a ...interface{}) (n int, err error)
func Sscanf(str string, format string, a ...interface{}) (n int, err error)
func Sscanln(str string, a ...interface{}) (n int, err error)
```

Ein Webserver

```
// Ein einfacher Webserver

package main
import "net/http"

func main() {
    http.Handle("/", http.FileServer(http.Dir("/var/www")))
    http.ListenAndServe(":8080", nil)
}
```

```
// Webserver objektorientiert

package main
import "net/http"

func main() {
        h := http.FileServer(http.Dir("/var/www"))
        s := http.Server{Addr: ":8080", Handler: h}
        s.ListenAndServe()
}
```

Kommandozeilen

```
package main

import (
	"flag"
	"fmt"
)

func main() {
	flagSSL := flag.Bool("ssl", false, "Use SSL by default")
	flagHost := flag.String("host", "127.0.0.1", "Hostname or IP address")
	flagPort := flag.Int("port", 9000, "Port number")
	flag.Parse()

	fmt.Printf("SSL: %t, Host: %s, Port: %d\n", *flagSSL, *flagHost, *flagPort)
}
```

```
$ go run flag.go -ssl -host localhost -port 8192
SSL: true, Host: localhost, Port: 8192
```

Kritik an Go

- Fehlende Versionsverwaltung von Bibliotheken
- Fehlende Templates (Generics)
- (Zu) viele Möglichkeiten, eine Variable zu deklarieren
- Typen bei := unklar
- Regeln für Zeilenumbruch zu streng
- Kooperativer Scheduler für Goroutinen
- Versehentliches Erfüllen von Interfaces
- Operatoren . () … haben zu viele Bedeutungen
- Bibliotheken außerhalb des Google-Universums noch dünn

Weiterführende Informationen

- Deutschsprachige Bücher:
  - Frank Müller: Systemprogrammierung in Google Go: Grundlagen, Skalierbarkeit, Performanz, Sicherheit. dpunkt Verlag 2011
  - Christian Maurer: Nichtsequentielle Programmierung mit Go 1. Springer Vieweg, 2012
- Projekte: http://code.google.com/p/go-wiki/wiki/Projects
- Ressourcen: http://go-lang.cat-v.org/
- Foren: golang-nuts, golang-dev
- Blogs: http://blog.golang.org/ http://dave.cheney.net/ http://hackgolang.blogspot.de/ http://www.goinggo.net/ http://golang-examples.tumblr.com/