

Die Programmiersprache Go

Troisdorfer Linux User Group
5. September 2019

Harald Weidner
hweidner@gmx.net

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_\(programming_language\)](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>]

Die Anfänge von Go

- Entwickelt seit 2007
 - Erste Ideen in 45-minütigen Compiler-Kaffeepausen
 - Unzufriedenheit mit C/C++, Java und Skriptsprachen
- Ansatz: C
 - alles, was unsichere Programmierung fördert
 - alles, was den Compiler langsam macht
 - + Nebenläufigkeit / Parallelität
 - + moderne Datentypen und Objektsystem
 - + umfangreiche Standardbibliothek
 - + moderne Toolchain

Die Erfinder von Go

Robert Grisemer

- Java Hotspot VM

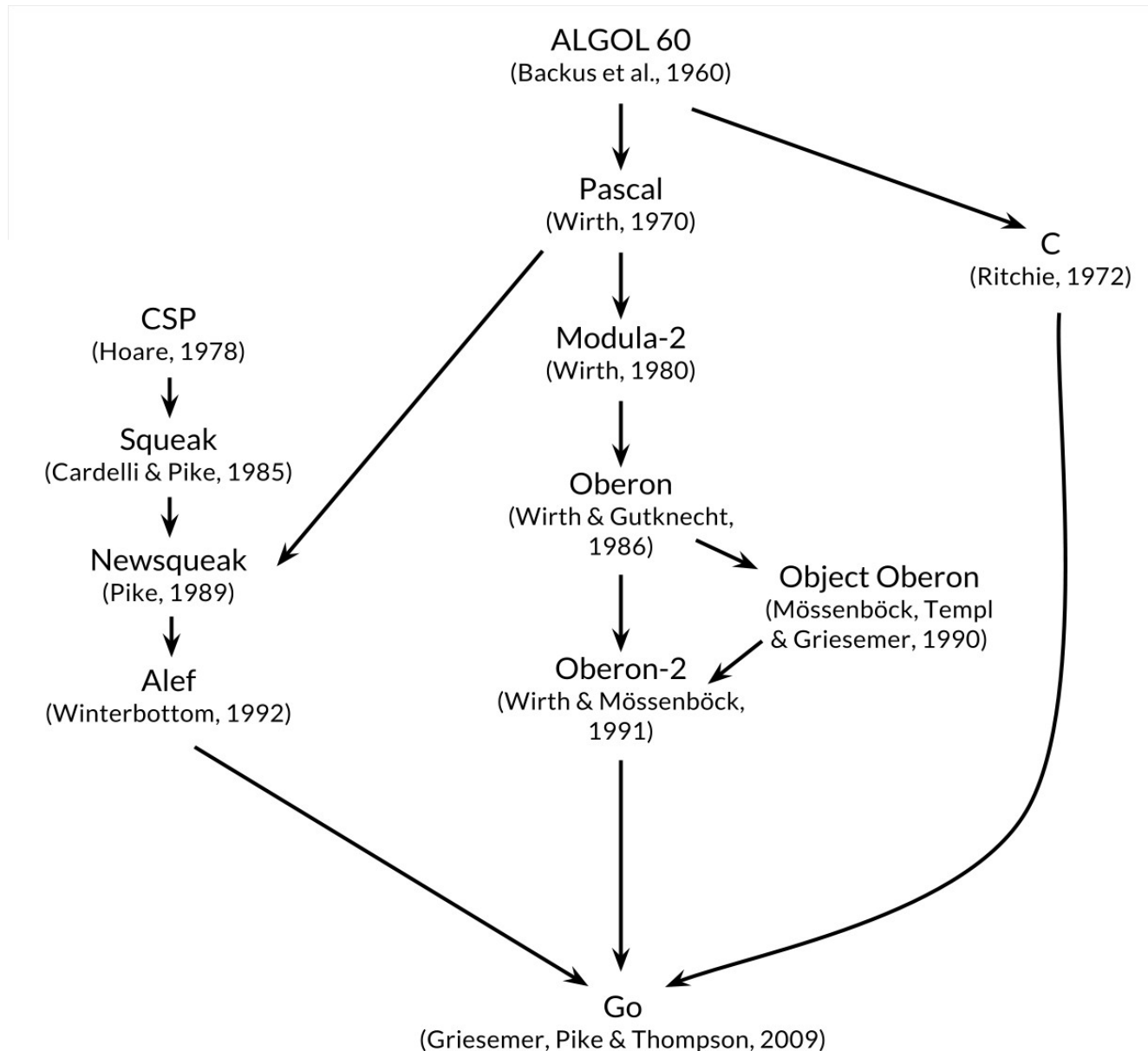
Rob Pike

- Plan 9, Newsqueak, UTF-8

Ken Thomson

- B, Multics, Plan 9, UTF-8

Go Stammbaum



Quelle:

A. Donovan,
B. Kernighan:

The Go
Programming
Language

Addison-Wesley,
2016

Go Timeline

- Erste Entwicklung seit 2007
- Erste Veröffentlichung im Nov. 2009
 - BSD-Style Lizenz
- Go 1.0 im März 2012
 - Go 1 Kompatibilitätsversprechen
- Alle 6 Monate ein Minor Release (Feb. / Aug.)
 - Jeweils 12 Monate Support für ein Release
 - Patch Releases bei Bedarf
- Aktuell: Go 1.13 (seit 4.9.2019)

Go Compiler

Go Frontend für GCC

- Debian-Paket: gccgo
- Dynamisch gelinkte Binaries
- Viele Plattformen und Betriebssysteme
- Teilweise bessere Performance der Programme

Go Compiler (Gc)

- Debian-Paket: golang
- Statisch gelinkte Binaries
- Verfügbar für i386, amd64, arm, ppc, mips, sparc, s390, wasm
- Linux, *BSD, AIX, Plan 9, Windows, MacOSX, Android, Solaris, NaCl
- Der schnellere Compiler

Bekannte Anwendungen in Go

Cloud und Infrastruktur

- Docker, RKT, Kubernetes, Juju, uRoot, LXD, Terraform, etcd, Consul, KataContainers, gVisor, CloudFoundry, ...

Datenbanken

- Prometheus, Grafana, InfluxDB, TiDB, Dgraph, Vitess, ...

WWW und webbasierte Anwendungen

- Caddy, Traefik, Hugo, Perkeep, Gitea, Gogs, Mattermost, Keybase, ...

Beispiel: hello.go

```
package main

import (
    "fmt"
    "runtime"
)

func main() {
    fmt.Println("Hello, Gophers!")
    fmt.Println("Go", runtime.Version(),
        "on", runtime.GOOS,
        "/", runtime.GOARCH)
}
```

Go Syntax – Schlüsselwörter

Nur 25 Schlüsselwörter

- dürfen nicht anderweitig verwendet werden
- garantiert keine Änderungen in Go 1
(Grund: Go Kompatibilitätsversprechen)

<code>break</code>	<code>default</code>	<code>func</code>	<code>interface</code>	<code>select</code>
<code>case</code>	<code>defer</code>	<code>go</code>	<code>map</code>	<code>struct</code>
<code>chan</code>	<code>else</code>	<code>goto</code>	<code>package</code>	<code>switch</code>
<code>const</code>	<code>fallthrough</code>	<code>if</code>	<code>range</code>	<code>type</code>
<code>continue</code>	<code>for</code>	<code>import</code>	<code>return</code>	<code>var</code>

Go Syntax – vordefinierte Namen

20 vordefinierte Typen

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

4 vordefinierte Konstanten

nil false true iota

15 vordefinierte Funktionen

append	complex	imag	new	println
cap	copy	len	panic	real
close	delete	make	print	recover

Go Syntax – Datentypen

Zusammengesetzte Datentypen

```
var a [32]byte           // Array
var s []string           // Slice
var m map[string]int     // Map
var p *int               // Pointer
var f func(int32) int64  // 1st Class Funktion

type IPv6 [16]uint8      // Typdefinition

type Person struct {     // Struct
    Name, Vorname string
    Alter           uint
}
```

Go Syntax – Kontrollstrukturen

```
// for-Schleife  
for i:=0; i<10; i++ {  
    fmt.Println(i)  
}
```

```
// while-Schleife  
for i<=10 { ... }
```

```
// Endlosschleife  
for { ... }
```

```
if a == 0 {  
    return 0  
} else if a > 0 {  
    return 1  
} else {  
    return -1  
}
```

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

Go Syntax – Funktionen

```
func mult(a, b int64) int64 {  
    return a * b  
}  
  
func add(a, b int64) (c int64) {  
    c = a + b  
    return  
}  
  
func div_mit_rest(a, b int64) (q, r int64) {  
    q = a / b  
    r = a % b  
    return  
}
```

Go Packages

- Programme bestehen aus Packages
 - Ein oder mehrere Sourcefiles pro Package
 - I.d.R.: ein Package = ein Verzeichnis
- Exportierte Bezeichner beginnen mit Großbuchstaben
 - Gilt für **alles**: Variablen, Konstanten, Typen, Interfaces, Funktionen, Methoden, struct-Elemente
 - Alles andere ist nicht außerhalb des Package sichtbar
- Import ungenutzter Packages ist ein Fehler!
- Pseudo-Packages: builtin, C, unsafe
- Neu ab Go 1.12: Modules (Sammlung von Packages)

Go Compiler (1)

- C/C++ Compiler (und andere) sind langsam
 - Ältere, historisch gewachsene Compiler
 - Umfangreiche Sprachen, komplexe Parser
 - Das Hauptproblem sind die zirkulären Abhängigkeiten der Header

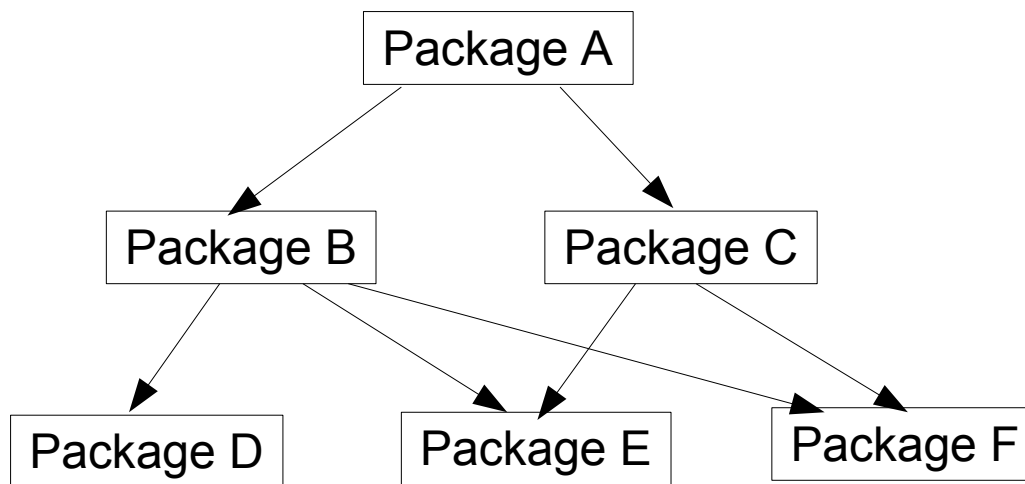
Go Compiler (2)

- Auf einem Mac (OS X 10.5.7, GCC 4.0.1):
 - C: `#include <stdio.h>`
liest 360 Zeilen aus 9 Dateien
 - C++: `#include <iostream>`
liest 25.326 Zeilen aus 131 Dateien
 - Objective-C: `#include <Cocoa/Cocoa.h>`
liest 112.047 Zeilen aus 689 Dateien
- Go: `import "fmt"`
 - Liest 195 Zeilen aus **einer** Datei
(enthält auch Infos über 6 abhängige Packages)

[Quelle: <http://web.stanford.edu/class/ee380/Abstracts/100428-pike-stanford.pdf>]

Go Compiler (3)

- Go erlaubt keine zirkulären Abhängigkeiten.



- Compiliert zuerst D/E/F, dann B/C, dann A
- Parallelisierung
- Caching
- Import eines ungenutzten Package ist ein Fehler

Objektorientierung in Go

- Objektorientierung ist nicht zwingend
 - Nutzung nur wenn es sinnvoll ist
- Keine Klassen
 - (Fast) jeder selbstdefinierte Typ kann Methoden haben
- Keine Vererbung
 - Vererbung führt zu schwerfälligen Klassenhierarchien, Basisklassen schwer änderbar
 - Vererbung verwässert Trennung von Schnittstelle und Implementierung

Objektorientierung - Methoden

```
type Celsius float32
type Fahrenheit float32

func (c Celsius) Print() {
    fmt.Printf("%.1f°C", float32(c))
}

func (f Fahrenheit) Print() {
    fmt.Printf("%.1f°F", float32(f))
}

func main() {
    x := Celsius(20)
    y := Fahrenheit(65)
    x.Print()           // 20.0°C
    y.Print()           // 65.0°F
}
```

Objektorientierung - Polymorphie

- Polymorphie durch Interfaces

```
type Printer interface {  
    Print()  
}  
  
func main() {  
    var p Printer  
    p = Celsius(20)  
    p.Print()           // 20.0°C  
    p = Fahrenheit(65)  
    p.Print()           // 65.0°F  
}
```

Objektorientierung - Komposition

```
// aus dem Package io
type Reader interface { Read(p []byte) (n int, err error) }
type Writer interface { Write(p []byte) (n int, err error) }

f1, _ := os.Open("/etc/passwd")
f2, _ := os.Open("/etc/group")
f3, _ := os.Create("/tmp/pwdgrp")

var r io.Reader
r = io.MultiReader(f1, f2)
r = io.LimitReader(r, 4096)
r = io.TeeReader(r, f3)

io.Copy(os.Stdout, r)
```

[Beispiel – es fehlen das Schließen der Dateien und Fehlerbehandlung!]

Objektorientierung - Embedding

```
type LockableTime struct {  
    time.Time    // Methoden Hour(),  
                // Minute(), Second()  
    sync.Mutex   // Methoden Lock(),  
                // Unlock  
}  
  
var lt LockableTime  
  
lt.Lock()  
fmt.Println(lt.Hour(), lt.Minute(), lt.Second())  
lt.Unlock()
```

- „Flache“ Nutzung der Methoden der eingebetteten Typen
- Als Ersatz für Vererbung oft ausreichend

Sichere Software mit Go

Verzicht auf gefährliche Konstrukte

- Garbage Collection statt manueller Speicherverwaltung
- Starke, statische Typisierung
- Keine automatische Typumwandlung
- Vorinitialisierung aller Typen mit Standardwerten
- Keine Compilerwarnungen (aber `go vet`)
- Indexprüfungen bei Array-Zugriffen
- Pointer, aber keine Pointer-Arithmetik
- Kein undefiniertes Verhalten
- Increment (`x++`) und Decrement (`x--`) sind Anweisungen

Sichere Software mit Go

Garbage Collection

- Es ist erlaubt (und guter Stil), Referenzen auf lokale Objekte zu publizieren

```
func answer() *int {  
    var i int = 42  
    return &i  
}
```

- Escape Analysis: Objekt wird automatisch auf dem Heap erzeugt
- Garbage Collector löscht Objekt, wenn keine Referenz darauf mehr existiert

Sichere Software mit Go

Statische Typisierung

- Keine automatische Typumwandlung
- Benannte Typen sind unterschiedlich
Der Entwickler hat ihnen absichtlich verschiedene Namen gegeben

```
type Celsius float32
type Fahrenheit float32

var a Celsius
var b Fahrenheit

a = b    // Fehler: a und b haben
b = a    // unterschiedliche Typen
```

Sichere Software mit Go

Indexprüfung bei jedem Zugriff auf Array/Slice

```
var a, b [100]int

for i := 0; i < len(a); i++ {
    b[i] = a[i] + a[i+1]
}
// Runtime Error bei i=99
```

- Beeinträchtigung der Performance (Benchmarks)
- In anderen Sprachen schwer zu findende Laufzeitfehler

Nebenläufigkeit in Go

- **Communicating Sequential Processes (CSP)**

- Tony Hoare (University of Oxford), 1978
- Prozessalgebra zur Beschreibung von Interaktionen zwischen unabhängigen Prozessen

- **Goroutine**

- Funktion, die nebenläufig ausgeführt wird
- Syntax: `go f()` oder als Closure: `go func() {...}()`
- Implizite Verwendung, z.B. durch Bibliothek `net/http`

- **Channel**

- Typisierter Kanal (Kommunikation, Synchronisation)
- Buffered oder unbuffered

Nebenläufigkeit (Beispiel)

```
func fib(c chan int, n int) {  
    x, y := 0, 1  
    for i := 0; i < n; i++ {  
        c <- x  
        x, y = y, x+y  
    }  
    close(c)  
}  
  
func main() {  
    c := make(chan int)  
    go fib(c, 30)  
  
    for f := range c {  
        fmt.Println(f)  
    }  
}
```

Nebenläufigkeit in Go

- Goroutinen sind wesentlich leichtgewichtiger als Betriebssystem-Threads
 - Initiale Stackgröße 2 kB (1 Mio. Goroutinen = 2 GB)
 - Scheduler in Go Runtime verteilt Goroutinen auf Threads des Betriebssystems
- Parallelität durch OS-Threads
 - Gesteuert durch Env.-variable GOMAXPROCS
 - Default (seit Go 1.5): Anzahl CPUs
 - Oder explizit im Programm:
`runtime.GOMAXPROCS(16)`

Go Toolchain

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

Cross Compile

- Beispiel: auf einem Linux/amd64 System

```
$ go build -o hello_linux
```

```
$ GOOS=linux GOARCH=arm64 go build -o hello_raspi
```

```
$ GOOS=windows GOARCH=amd64 go build -o hello_windows
```

```
$ GOOS=darwin GOARCH=amd64 go build -o hello_mac
```


Dokumentation

```
// Package signum calculates the sign of a number.
//
// Package signum provides functions which calculate
// the signum of a number of different types.
package signum

// Function SignumInt calculates the signum of
// an integer value.
func SignumInt(x int) int {
    switch {
        case x == 0: return 0
        case x < 0:  return -1
        default:    return 1
    }
}
```

Dokumentation

```
$ go doc fmt.Println  
  
$ go doc signum  
$ go doc signum SignumInt  
  
$ godoc -http=:8000
```

Unit Tests (1)

- Beispiel: zu testende Funktion

```
// signum.go
package signum

func SignumInt(x int) int {
    switch {
        case x == 0: return 0
        case x < 0:  return -1
        default: return 1
    }
}
```

Unit Tests (2)

```
// signum_test.go
package signum

import "testing"

var testcases = []struct { in, out int } {
    {0, 0}, {-5, -1}, {7, 1} }

func TestSignumInt(t *testing.T) {
    for _, testcase := range testcases {
        v := SignumInt(testcase.in)
        if v != testcase.out {
            t.Fatalf("sign(%d) is %d (expected %d)\n",
                testcase.in, v, testcase.out);
        }
    }
}
```

Unit Tests (3)

```
$ go test
```

```
$ go test -v
```

```
$ go test -cover
```

```
$ go test -coverprofile out
```

```
$ go tool cover -html=out
```

Weiterführende Informationen

- Go Homepage <http://golang.org/>
- Tutorial <http://tour.golang.org/>
- Playground <http://play.golang.org/>
- Golang Book <http://golang-book.com/>
- Language Design in the Service of Software Engineering
<http://talks.golang.org/2012/splash.article>
- Less is exponentially more
<http://commandcenter.blogspot.de/2012/06/less-is-exponentially-more.html>
- Another Go at Language Design
<http://web.stanford.edu/class/ee380/Abstracts/100428-pike-stanford.pdf>
- Building Large-Scale Distributed Systems
<http://static.googleusercontent.com/media/research.google.com/de//people/jeff/stanford-295-talk.pdf>