

Go

Handout zum Vortrag

Überblick

Die von Google geschaffene Programmiersprache Go vereint die Vorteile statischer Typisierung mit der Leichtigkeit einer Skriptsprache. Go wurde aus Unzufriedenheit über die bestehenden Sprachen zur Softwareentwicklung wie C++ oder Java im Kontext heutiger Computersysteme, insbesondere im Hinblick auf skalierbare Netzwerkdienste, Cluster- und Cloud-Computing, entwickelt.

Go hat eingebaute Parallelisierung und besitzt einen kompletten Satz einfach zu bedienender Werkzeuge.

Die wichtigsten Eigenschaften von Go

Go ist eine kompilierte Sprache bietet Closures und Reflexion sowie Typsicherheit und eine Garbage Collection. Objektorientierung unterstützt Go durch Interfaces und Mixins. Klassen gibt es nicht.

Die Syntax von Go ist ähnlich wie die Syntax von C oder Java, wobei weniger Klammern verwendet werden und keine Semikolons am Abschluss von Anweisungen benötigt werden.

Nebenläufigkeit ist durch die eingebaute concurrency primitives: Goroutines und Channels in Go eingebaut. Zudem besitzt Go einen eigenen Garbage Collector, erlaubt die Verwendung von Pointers, verzichtet jedoch auf Zeigerarithmetik.

Weiterführende Informationen

Go lernen

A Tour of Go, eine interaktive Einführung zur Programmiersprache

<https://tour.golang.org/welcome/1>

Das Go Playgound

<https://play.golang.org/>

Effective Go

http://golang.org/doc/effective_go.html

Go Programmiersprache Spezifikation

<https://golang.org/ref/spec>

Cheatsheets

<https://github.com/a8m/go-lang-cheat-sheet>

<https://github.com/basti1302/go-lang-cheat-sheet>

Go – Node.js Diskussionen und Meinungen

Farewell Node.js

<https://medium.com/code-adventures/farewell-node-js-4ba9e7f3e52b>

Why I Won't Switch To Go (from Node.js)

<http://duncanmismith.com/blog/why-i-wont-switch-to-go/>

Go Cheatsheet

Deklarationen

```
var foo int // Ohne Initialisierung
var foo int = 42 // Mit Initialisierung
var foo, bar int = 42, 1302 // Mehrere
var foo = 42 // Typinferenz
foo := 42 // Kurzform, immer implizit
const constant = "This is a constant"
```

Loops

```
for i := 1; i < 10; i++ {...}
for ; i < 10; {...} // while - loop
for i < 10 {...} // Semikolons nicht nötig
for {...} // while (true)
```

If

```
if x > 0 {
    return x
} else {
    return -x
}
if a := b + c; a < 42 {
    return a
} else {
    return a - 42
}
```

Arrays

```
var a [10]int // int array mit 10 Elementen
a[3] = 42 // setzen
i := a[3] // auslesen
// Deklarieren
a := [2]int{1, 2}
a := [...]int{1, 2} // Kompilierer findet die Anzahl
Elemente automatisch
```

Structs

```
type Man struct {
    name string
    age int
}
func main() {
    man1 := Man{"Ariel", 26}
    man2 := Man {
        age: 24,
        name: "Ben",
    }
    //Zugriff auf structs
    for _, man := range []Man {man1, man2} {
        println(man.name)
    }
}
```

Slices

```
var a []int // Slice deklarieren, ähnlich wie Array
aber ohne Länge
var a = []int {1, 2, 3, 4} // Slice deklarieren und
initialisieren
a := []int{ 1, 2, 3, 4 } // shorthand
var b = a[lo:hi] // slice kreieren (view of array)
von index lo bis hi-1
var b = a[1:4] // slice von index 1 to 3
var b = a[:3] // ohne unterer index impliziert 0
var b = a[3:] // ohne oberer index impliziert
len(a)
// Mit make Slices kreieren
a = make([]byte, 5, 5) // 1. Parameter: Länge,
2. Parameter: Kapazität = make([]byte, 5) //
Kapazität ist optional
```

Funktionen

```
func functionName() {}
func functionName(param1 string, param2 int)
{}
// parameter mit dem gleichen Typ
func functionName(param1, param2 int) {}
// Rückgabewert
func functionName() (int) {
    return 42
}
// Mehrere Rückgabewerte
func returnMulti() (int, string) {
    return 42, "foobar"
}
```

Coroutines

```
import(
    t "time"
    f "fmt"
)
func say(s string) {
    for i := 0; i < 5; i++ {
        t.Sleep(100 * t.Millisecond)
        f.Println(s)
    }
}
func main() {
    go say("world")
    say("hello")
}
```

Channels

```
func seq(num int, c chan int) {
    x, y := 0, 1
    for num != 0 {
        x, y = y, x + y
        num--
    }
    c <-x
}
func main() {
    //channels müssen vor ihrer Anwendung
    //deklariert werden
    ch := make(chan int)
    go seq(50, ch)
    go seq(20, ch)
    res1, res2 := <-ch, <-ch
    println(res1, res2) //12586269025 6765
}
```