Go Programming

Concepts of Programming Languages
11 October 2018

Johannes Weigend (QAware GmbH)
University of Applied Sciences Rosenheim

Golang

- Go is an open source programming language for distributed and parallel systems
- Go addresses the problems of C ++ backend development at Google
- The Go core team is prominent: Robert Griesheimer (Hotspot VM), Ken Thompson (UX / B) and Rob Pike (UTF-8)
- Go is actively developed since 2008 (current version 1.11)
- Go is **THE** language behind the Cloud Native Stack

www.cncf.io/ (https://www.cncf.io/)

- All essential components are written in Go: Docker, Kubernetes, Etcd, Prometheus, Grafana -> An important reason to take a closer look at Go

Characteristics

- Compiled (cross compiler for OS: Mac, Windows, Linux and CPU: ARM, x86 + Amd64)
- Static type system with runtime support (Reflection, Dynamic Types)
- Static linker (Single Binary) -> Ideal for Docker containers
- Focus on fast compile times (the entire Go codebase compiles in <10 seconds)
- Simple (less keywords like C - 25/32)
- Object-oriented, Functional, Parallel, Modular (vgo) and Versioned

Gopher - The Go Mascot

Hello World

```
// Copyright 2018 Johannes Weigend
// Licensed under the Apache License, Version 2.0

package main

import "fmt"

func main() {
    fmt.Printf("Hello %s", "Programming with Go \xE2\x98\xAF\n") // \xE2\x98\xAF -> ☯
    fmt.Printf("Hello %s", "Programming with Go ☯\n")
}
```

Run

- Go source files are UTF-8 encoded
- Package names are not hierarchical!
- Package import names are hierarchical!
- Public functions begin with a capital letter

5

Functions and Control Structures: Example Palindrome

```
// IsPalindrome implementation. Does only work for 1-Byte UTF-8 chars (ASCII).
func IsPalindrome(word string) bool {
    for pos := 0; pos < len(word)/2; pos++ {
        if word[pos] != word[len(word)-pos-1] {
            return false
        }
    }
    return true
}
```

- The type of a variable is behind the name!
- The return type of a function is behind the parameter list
- Conditions (if, for) are not clipped with ()
- if, for ... statements need curly braces {}
- semicolons are omitted

6

Go directly supports Unit Tests via "go test"

```
// palindrome_test.go
func TestPalindrome(t *testing.T) {
    if !IsPalindrome("") == true {
        t.Error("isPalindrome('' should be true. But is false.")
    }
    if !IsPalindrome("o") == true {
        t.Error("isPalindrome('o' should be true. But is false.")
    }
    if !IsPalindrome("oto") == true {
        t.Error("isPalindrome('oto' should be true. But is false.")
    }
    if !IsPalindrome("ottos") == false {
        t.Error("isPalindrome('ottos' should be false. But is true.")
    }
```

- The unit test for a file is located in a _test.go file of the same name
- TestXY functions are called automatically
- The test context testing.T controls the execution

Functions and Control Structures: Example Palindrome (UTF-8)

```
// IsPalindrome2 is using runes. This works for all UTF-8 chars (SBC, MBC).
func IsPalindrome2(word string) bool {
    var runes = []rune(word)
    for pos, ch := range runes {
        if ch != runes[len(runes)-pos-1] {
            return false
        }
    }
    return true
}
```

- The rune type is an alias for int32 and can store all UTF-8 characters
- rune (string) converts a string into a slice of runes
- The range operator has two return values: the position and the current value

```
pos, ch: = range runes
```

8

Functions and Control Structures: Example String Reverse (UTF-8)

```
// Reverse reverses an unicode string.
func Reverse(s string) string {
    r := []rune(s)
    for i, j := 0, len(r)-1; i < len(r)/2; i, j = i+1, j-1 {
        r[i], r[j] = r[j], r[i]
    }
    return string(r)
}
```

9

Arrays and Slices

- Arrays have a fixed length and can not resized
- Slices are views to underlying arrays
- Slices can be resized (append())
- The underlying array grows automatically (if needed)
- Both have a length and a capacity

```
var array = [3]int{1, 2, 3}
fmt.Println(array) // -> [1 2 3]

slice := array[:] // len(slice) == 3, cap(slice) == 4

slice = append(slice, 4) // len(slice) == 4, cap(slice) == 4
slice = append(slice, 5) // len(slice) == 5, cap(slice) == 8

fmt.Printf("%v\n", slice) // -> [1 2 3 4, 5]
fmt.Printf("%v\n", array) // -> [1 2 3]
```

Functions and Control Structures: Example Palindrome (Reverse)

```
// IsPalindrome3 is implemented by reusing Reverse(). Reverse works for UTF-8 chars.
func IsPalindrome3(word string) bool {
    return strings.Reverse(word) == word
}
```

- Strings, arrays are compared in Go with ==
- Slices, Maps are compared with cmp.Equal

Special comparison with the cmp package

```
opt: = cmp.Comparer (func (x, y float64) bool {
  delta: = math.Abs (x - y)
  mean: = math.Abs (x + y) / 2.0
  return delta / mean <0.00001
})

x: = [] float64 {1.0, 1.1, 1.2, math.Pi}
y: = [] float64 {1.0, 1.1, 1.2, 3.14159265359} // Accurate enough to Pi
z: = [] float64 {1.0, 1.1, 1.2, 3.1415} // Diverges too far from Pi

fmt.Println (cmp.Equal (x, y, opt)) // -> true
fmt.Println (cmp.Equal (y, z, opt)) // -> false
fmt.Println (cmp.Equal (z, x, opt)) // -> false
```

- Comparator (Comparer) is optional (as in Java)

Pointer I

```
func swap1(x, y int) {
    x, y = y, x
}
```

```
func swap2(x *int, y *int) {
    *x, *y = *y, *x
}
```

```
func swap3(x **int, y **int) {
    *x, *y = *y, *x
}
```

- The double assignment saves a variable (tmp)!
- Pointers are transferred as well as values by copy
- There is no pointer arithmetic (p ++) as in C / C ++

Pointer II

```
func main() {
    var a, b = 1, 2
    fmt.Printf("Initial : a=%d, b=%d\n", a, b)
    a, b = b, a
    fmt.Printf("After a,b = b,a : a=%d, b=%d\n", a, b)
    swap0 := func(x, y int) (int, int) {
        return y, x
    }
    a, b = swap0(a, b)
    fmt.Printf("After a,b = swap0(a,b) : a=%d, b=%d\n", a, b)
    swap1(a, b)
    fmt.Printf("After swap1(a,b) : a=%d, b=%d\n", a, b)
    swap2(&a, &b)
    fmt.Printf("After swap2(&a,&b) : a=%d, b=%d\n", a, b)
    pa, pb := &a, &b
    swap3(&pa, &pb)
    fmt.Printf("After swap3(&pa, &pb): a=%d, b=%d, pa=%p, pb, %p\n", a, b, pa, pb)
}
```

Run

14

The Flag API simplifies Command Line Utilities

```
func main() {
    // construct a string flag with a default ip address and a description.
    ip := flag.String("ip", "192.168.1.1", "Overrides the default IP address.")
    port := flag.String("port", "8080", "Overrides the default listen port.")

    // flag.Args() parses the arg of our program.
    if len(flag.Args()) == 0 {
        fmt.Printf("Program Usage:\n")
        // PrintDefaults() prints a description and the default values to stdout.
        flag.PrintDefaults()
    }

    flag.Parse()

    fmt.Println("\nDefault value for IP: " + *ip)
    fmt.Println("\nDefault value for port: " + *port)
}
```

Run

15

Maps and Slices - Example Book Index

```
// Page contains an array of words.
type Page []string

// Book is an array of pages.
type Book []Page

// Index contains a list of pages for each word in a book.
type Index map[string][]int

// MakeIndex generates an index structure
func MakeIndex(book Book) Index {
    idx := make(Index)
    for i, page := range book {
        for _, word := range page {
            pages := idx[word]
            idx[word] = append(pages, i)
        }
    }
    return idx
}

// Stringer support
```

Object Oriented Programming

17

Go has no classes but types and functions

```
// Rational represents a rational number numerator/denominator.
type Rational struct {
    numerator   int
    denominator int
}
```

```
// Multiply method for rational numbers (x1/x2 * y1/y2)
func (x Rational) Multiply(y Rational) Rational {
    return NewRational(x.numerator*y.numerator, x.denominator*y.denominator)
}
```

```
// NewRational constructor function
func NewRational(numerator int, denominator int) Rational {
    if denominator == 0 {
        panic("division by zero")
    }
    r := Rational{}
    divisor := gcd(numerator, denominator)
    r.numerator = numerator / divisor
    r.denominator = denominator / divisor
    return r
}
```

The typical OO Syntax (object.method ()) is supported in Go

```
func TestRational(t *testing.T) {

    r1 := NewRational(1, 2)
    r2 := NewRational(2, 4)

    // test equal
    if r1 != r2 {
        t.Error("1/2 should be equal to 2/4 but is not.")
    }

    // test multiply
    r3 := r1.Multiply(r2)
    if r3 != NewRational(1, 4) {
        t.Error(fmt.Sprintf("1/2 * 1/2 should be 1/4 but ist %v", r3))
    }

    // test add
    r4 := r3.Add(r3)
    if r4 != NewRational(1, 2) {
        t.Error(fmt.Sprintf("1/4 + 1/4 should be 1/2 but ist %v", r4))
    }
}
```

"ToString" - The Stringer Interface

```
// Stringer
func (x Rational) String() string {
    return fmt.Sprintf("(%v/%v)", x.numerator, x.denominator)
}
```

```
r1: = NewRational (1, 2)
fmt.Println (r1) // -> (1/2)
```

- Works for any type

```
type number int64
func (n Number) String () string {
    return fmt.Sprintf (% x, n)
}
```

20

The Base Type for Container is the Empty Interface (interface {})

```
// Stack is a generic LIFO container for untyped object.
type Stack struct {
    data []interface{}
}

// NewStack constructs an empty stack.
func NewStack() *Stack {
    return new(Stack)
}

// Push pushes a value on the stack.
func (s *Stack) Push(value interface{}) {
    s.data = append(s.data, value)
}

// Pop pops a value from the stack. It returns an error if the stack is empty.
func (s *Stack) Pop() interface{} {
    if len(s.data) == 0 {
        panic("can not pop: empty stack")
    }
    var result = s.data[len(s.data)-1]
    s.data = s.data[0 : len(s.data)-1]
    return result
}

// Size returns the count of elements in the Stack
func (s *Stack) Size() int {
```

```
    return len(s.data)
}

// Get returns the n-th element in the Stack
func (s *Stack) Get(idx int) interface{} {
    return s.data[idx]
}
```

21

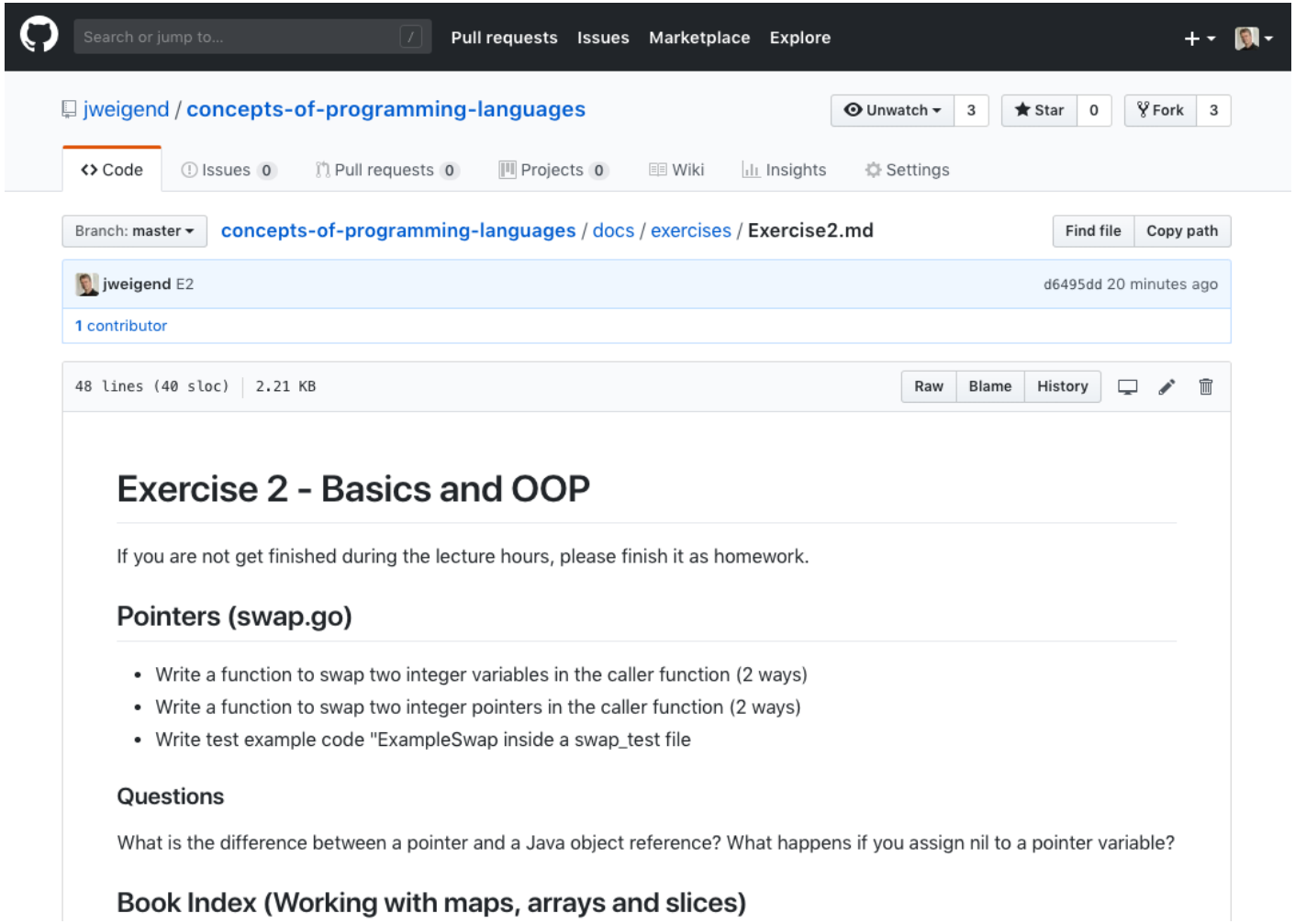
The Downcast Syntax is similar to a Function Call

```
func TestCasting(t *testing.T) {

    s := NewStack()
    s.Push(1)
    s.Push(2)

    sum := 0
    for i := 0; i < s.Size(); i++ {
        sum += s.Get(i).(int) // type assertion = cast from interface{} to int
    }

    if sum != 3 {
        t.Error(fmt.Sprintf("Sum result should be 3 but is %v", sum))
    }
}
```

- Object types (e.g., Containers) are used by pointers (no copies)
- Data types (e.g., Money, Date, ...) are used by value (copies)
- The downcast is safe (== dynamic_cast in C ++ or cast in Java)
- No dereference (*s) is needed to call a method

Exercise 2

github.com/jweigend/concepts-of-programming-languages/blob/master/docs/exercises/Exercise2.md (https://github.com/jweigend/concepts-of-programming-languages/blob/master/docs/exercises/Exercise2.md)


Search or jump to...
Pull requests
Issues
Marketplace
Explore
jweigend / concepts-of-programming-languages
Unwatch 3
Star 0
Fork 3
Code
Issues 0
Pull requests 0
Projects 0
Wiki
Insights
Settings
Branch: master
concepts-of-programming-languages / docs / exercises / Exercise2.md
Find file
Copy path
jweigend E2
d6495dd 20 minutes ago
1 contributor
48 lines (40 sloc) | 2.21 KB
Raw
Blame
History
Exercise 2 - Basics and OOP
If you are not get finished during the lecture hours, please finish it as homework.
Pointers (swap.go)
Write a function to swap two integer variables in the caller function (2 ways)
Write a function to swap two integer pointers in the caller function (2 ways)
Write test example code "ExampleSwap inside a swap_test file
Questions
What is the difference between a pointer and a Java object reference? What happens if you assign nil to a pointer variable?
Book Index (Working with maps, arrays and slices)

Thank you

Johannes Weigend (QAware GmbH)
University of Applied Sciences Rosenheim
johannes.weigend@qaware.de (mailto:johannes.weigend@qaware.de)
http://www.qaware.de (http://www.qaware.de)
@johannesweigend (http://twitter.com/johannesweigend)