GO

A new refreshing language
22 October 2016

Cristian Pavan
(some kind of) Web Developer

Why learn a new programming language

Learn a new programming language let you to solve problem in a better way
and you can see how different programming languages treat common situations.

Common situation in life:

- have a problem or an idea
- think a solution
- choose the right **tool** for the job

In IT problems, our tools are programming languages.

Go is an open source programming language that makes it easy to build simple, reliable, and efficient software.

What's go ?

Go is a *statically-typed* language program with syntax loosely derived from that of C that include:

- garbage collection
- type safety
- dynamic-typing capabilities
- large standard library
- CSP-style concurrent programming features

Why was it made?

The Go programming language is an open source project to make programmers more productive.

Story about Go

Conceived around 2007, announced late 2009 by:

- Robert Griesemer (*Research at Google*) @robertgriesemer (https://twitter.com/robertgriesemer)
- Rob Pike (*Unix*, *Plan 9*, *UTF-8*) @rob_pike (https://twitter.com/rob_pike)
- Ken Thompson (*designed orginal Unix*, *invented B*, *UTF-8*)

A slide about history is here: How go was made (https://talks.golang.org/2015/how-go-was-made.slide) .

Other active member

- Andrew Gerrand @enneff (https://twitter.com/enneff)
- Brad Fitzpatrick @bradfitz (https://twitter.com/bradfitz)
- Rick Hudson

Key features (1/5)

Learning Curve

Syntax like C with a small number of reserved words.

Generally there is only one way to do things.

Concurrency

Built-in *concurrency primitives*.

Use of *channels* to sincronize lightweight "thread" called *goroutine*, with `select` statement.

> Don't communicate by sharing memory, share memory by communicating.

Use all the cores of your system.

Key features (2/5)

Batteries Included

Large standar library (especially in network area), and a simple way to add external packages using go get

```
import (
    "net/http"
    "fmt"

    "database/sql"
    _ "github.com/lib/pq"

    "github.com/gorilla/mux"
)
```

Key features (3/5)

Fast Compilation

Initially made in C, now the compiler is in Go and use Static Code Analysis in order to generate better and smaller code.

Deployment

Go generates statically linked native binaries, without external dependencies.

This suits well for container/vm images.

Easy cross compilation:

```
GOOS=darwin GOARCH=amd64 go build main.go

GOOS=linux GOARCH=arm go build main.go
```

Key features (4/5)

Tools

Several tools to manipulate source code or get information

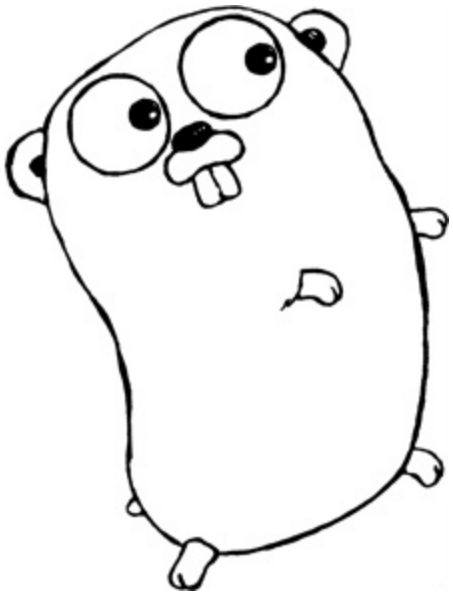
- gofmt

  > Gofmt's style is no one's favorite, yet gofmt is everyone's favorite.

- go vet
- godoc
- golint, goimports, gometalinter
- guru
- pprof
- delve

Key features (5/5)

a nice mascot, a *gopher*



Critics

- Lacks of Generic (expecially for functions with basic type)
- Dependency/Vendoring
- Too minimal
- Already stable

Critic: lacks of generics

Go does not support generics like other **strictly-typed** programming language (Java, C#, C++). You can't define function like that:

```
func sum(a Type, b Type) Type {
    return a + b
}
```

Yo have to use

```
interface{}
```

as generic type, and check the type inside a function:

```
func sum(a interface{}, b interface{}) interface{} {
    switch t := a.(type) {
        case int:
         ...
        case string:
        ...
    }
}
```

Critic: Dependency

Go default directory structure (exported via `$GOPATH`):

```
├── bin
├── pkg
└── src
    ├── github.com
        └── tux-eithel
        |    └── linuxday2016-golang
        └──gorilla
            └── mux
```

The `import` directive doesn't allow to specify a version, tag or hash, so in order to make **reproducibily builds** you need to use other tools like godep (https://github.com/tools/godep), gb (http://getgb.io/), gom (https://github.com/mattn/gom) or serivce like gopkg (http://labix.org/gopkg.in) because **src** folder is shared between projects.

The official solution is create a **vendor** folder inside your project and copy the dependencies.

There is a committee discussing a Package Management Proposal (https://groups.google.com/forum/#!msg/go-package-management/P8TehVoFLjg/Ni6VRyOjEAAJ) with the goal to implement it for Go 1.8 (january 2017).

In Deep (1/7)

Zero value

```
var i int
var ii = 0
iii := 0
```

all varaibles are of type **int** and have value **0**.

Array, **slice** and **map**:

```
array := [3]int{1, 2, 3}

slice := []int{5, 6, 7} // slice := make([]int, 3, 5)

collection := map[int]string{
    8:  "eight",
    9:  "nine",
    10: "ten",
}
```

In Deep (2/7)

If

```
if err := file.Chmod(0664); err != nil {
    log.Print(err)
}
```

For

```
// Like a C for
for init; condition; post { }

// Like a C while
for condition { }

// Like a C for(;;)
for { }


for key, value := range collection {
    ...
}
```

In Deep (3/7)

Multiple return values

```
func Write(b []byte) (n int, err error)

num, err := Write([]byte{})

value, ok := collection[10]
```

Blank identifier

```
if _, ok:= collection[11]; !ok {
    log.Print("not found")
}
```

In Depp (4/7)

Structs

```
package human // package name

type Human struct {
    Name     string // exported field
    sentence string // unexported field
}

func (h *Human) SetSentence (s string) { // method attached to
    h.sentence = s
}

func (h *Human) Speak() string {
    return h.Name + " says: " + h.sentence
}
```

In Depp (5/7)

Interfaces

Interfaces describe objects behaviour.

Interfaces are atuomatically implemented.

```
package main

type Speaker interface {
    Speak() string
}

func SpeakAll(speakers []Speaker) {
    for _, value := range speakers {
        speakers.Speak()
    }
}
```

The bigger the interface, the weaker the abstraction.

In Depp (6/7)

Embedding

```
type Reader interface {
    Read(p []byte) (n int, err error)
}

type Writer interface {
    Write(p []byte) (n int, err error)
}

// ReadWriter is the interface that combines the Reader and Wri
type ReadWriter interface {
    Reader
    Writer
}
```

In Deep (7/7)

Defer

```
f, err := os.Open(filename)

if err != nil {
    return err
}
defer f.Close()  // f.Close will run when function has finished

// rest of the code
```

Documentation

```
// funcName sees people and does stuff
func funcName(people []person) {

}
```

an example for the oauth2 package (https://godoc.org/golang.org/x/oauth2)

In Deep: goroutines

A goroutine is a lightweight thread managed by the Go runtime.

Concurrency is about dealing with lots of things at once.

Parallelism is about doing lots of things at once.

```
package main

import "fmt"

func main() {

    go func() {
        fmt.Println("first Goroutine")
    }()

    go func() {
        fmt.Println("second Goroutine")
    }()

    fmt.Println("Hi!")
}
```

Run

In Deep: channel

Channels are a typed conduit through which you can send and receive values

> Don't communicate by sharing memory, share memory by communicating.

```
package main

import "fmt"

func sum(a []int, c chan int) {
    sum := 0
    for _, v := range a {
        sum += v
    }
    c <- sum // send sum to c
}

func main() {
    a := []int{7, 2, 8, -9, 4, 0}

    c := make(chan int)
    go sum(a[:len(a)/2], c)
    go sum(a[len(a)/2:], c)
    x, y := <-c, <-c // receive from c

    fmt.Println(x, y, x+y)
}
```

Run

Channel with Select

```
package main

import (
    "fmt"
    "time"
)

func main() {
    tick := time.Tick(100 * time.Millisecond)
    boom := time.After(500 * time.Millisecond)
    for {
        select {
        case <-tick:
            fmt.Println("tick.")
        case <-boom:
            fmt.Println("BOOM!")
            return
        default:
            fmt.Println("    .")
            time.Sleep(50 * time.Millisecond)
        }
    }
}
```

Run

Go on Web

Example of a web application

```
package main

import (
    "flag"
    "fmt"
    "net/http"
)

func main() {
    flag.Parse()
    http.HandleFunc("/", func(h http.ResponseWriter, r *http.Re
        fmt.Fprint(h, "hello from server")

    })
    http.ListenAndServe(":8080", nil)
}
```

Run

Go to http://localhost:8080 (http://localhost:8080)

What's go place ?

Go, has been announced as a system language (used in server side).

Its scope right now is between microservices environment, useful command line utilities and web api.

It's going to be used to replace the slowest parts of monolithic applications.

Resources

Go Tour (https://tour.golang.org/)

Effective Go (https://golang.org/doc/effective_go.html)

Tips Links

Stupid Gopher Tricks (https://talks.golang.org/2015/tricks.slide)

Go Traps (https://go-traps.appspot.com/)

50 Shades of Go (http://devs.cloudimmunity.com/gotchas-and-common-mistakes-in-go-golang/index.html)

What Could Go Wrong? (http://slides.com/kevrone/what-could-go-wrong?token=HZlPo4g-#/)

Question ?

Thank you

Cristian Pavan
(some kind of) Web Developer
tux.eithel@gmail.com (mailto:tux.eithel@gmail.com)
@tux_eithel (http://twitter.com/tux_eithel)