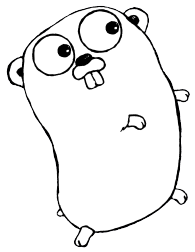
Go Lang
A linguagem do Google



Suelen Goularte Carvalho
Instituto de Matemática e Estatística
Universidade de São Paulo

Junho/2015

Contents

1 Introdução

1.1 Por que uma nova linguagem?

Em 2007 foi criada a linguagem de programação Go em resposta a alguns problemas enfrentados por ele ao desenvolver sistemas que trabalham com a sua infraestrutura. Estes problemas surgiram a partir do uso de processadores multinúcleos, sistemas distribuídos em redes, computação massiva e modelo de computação web, que apesar de tornar a computação muito mais potente, também agregou uma série de complexidades.

Sistemas como estes possuem milhares de linhas de código, com milhares de engenheiros de software trabalhando todos os dias, o que resulta em processos de merge e compilação muito complexos e demorados.

Houveram tentativas frustradas de resolver alguns destes problemas com sistemas desenvolvidos em linguagens já existentes, a maior parte deles em C++, Java e Python. E apesar de cada uma dessas linguagens possuírem características desejáveis, o mais desejável de fato, era encontrar todas estas características em apenas uma linguagem.

Características como linguagem compilada, garbage-collector, estaticamente tipada, lidar com concorrência, síntaxe simples dentre outras, foram reúnidas nesta nova linguagem denominada Go!

> Go foi desenhada para e por pessoas que escrevem, leêm, debugam e mantém grandes sistemas de software.

1.2 Go e sua história

Figure 1: Fundadores de Go. Robert Griesemer, Rob Pike e Ken Thompson

Go é uma linguagem de programação compilada, concorrente, forte e estaticamente tipada.

Foi desenvolvida por engenheiros do Google em 2007 como um projeto de tempo parcial (figura 1). Seus fundadores foram Ken Thompson, que participou de projetos como da linguagem B, C, Unix e UTF-8; Rob Pike, que participou de projetos como Unix e UTF-8 e Robert Griesemer, que participou de projetos como do Hotspot e JVM; além de outros engenheiros do Google.

Em 2008, Go passou de um projeto de tempo parcial para um projeto de tempo integral no Google, e com isso, muito outros engenheiros participaram, além claro, dos fundadores. Em 2009, Go se tornou Open Source e em 2010, começou a ser adotada por desenvolvedores de fora do Google.

As principais características encontradas em Go são:

- Compilada
- Garbage-collected
- Tem seu próprio Go Runtime
- Síntaxe simples
- Exclente pacote padrão
- Multiplataforma
- Orientada a Objetos (sem herança)
- Estaticamente e fortemente tipada
- Concorrente
- Compilada (goroutines)
- Suporte a clossures
- Dependências explícitas
- Funções com múltiplos retornos
- Ponteiros
- E muito mais.

Porém, algumas funcionalidades encontradas em outras linguagens, não serão vistas em Go, como por exemplo:

- Tratamento de erro
- Herança
- Generics
- Assertions
- Sobrecarga de métodos

1.3 Versões e ferramentas

Go teve sua versão 1 lançada em 2012. Em 2013 foram lançadas as versões 1.1 e 1.2. No meio de 2014 foi lançada a versão 1.3 e ao final, foi lançada a versão 1.4. Seu pacote padrão é considerado muito bom e completo por seus desenvolvedores, tornando desnecessário o uso de dependências externas e facilitando o gerenciamento das mesmas.

Go também é uma conjundo de ferramentas para o gerenciamento de códigos desenvolvidos com ela. Junto com a instalação de Go vem algumas ferramentas, dentre elas, as principais são:

- `build`: compilar pacotes e suas dependências;
- `run`: compila e executa programas Go;
- `clean`: remove arquivos extras de compilação e execução;
- `env`: imprime informações do ambiente de desenvolvimento Go;
- `test`: executa testes e benchmarks realizados com Go.

Além destas existem outras como: `fix`, `fmt`, `get`, `install`, `list`, `tool`, `version` e `vet`.

1.4 Quem está usando Go

Atualmente muitas empresas tem aderido ao uso da linguagem Go. Naturalmente, o Google foi o primeiro a usá-la em projetos produtivos. Com o passar dos anos, outras empresas que possuem um contexto similar, passaram a usar Go.

Na Figura 17 é possível ver algumas destas empresas. Uma lista completa pode ser encontrada em `https://github.com/golang/go/wiki/GoUsers`

Figure 2: Empresas que usam a linguagem de programação Go

2 Instalando e desenvolvendo com Go

Para instalarmos Go basta fazer o download do arquivo compactado da linguagem no site `https://golang.org/doc/install` e adicionar o diretório `bin` a variável de ambiente `PATH`.

Todo código Go possui como extensão `.go`. Para executá-lo podemos digitar no terminal `go run seuArquivo.go`, que irá compilar e executar o código, ou `go build seuArquivo.go` e em seguida `go run seuArquivo.go`, que irá respectivamente compilar e executar o programa Go.

Vamos agora conhecer algumas estruturas da linguagem de programação Go e entender um pouco sobre seu funcionamento.

2.1 Pacotes

- Cada programa em Go é contido em um pacote,
- Programas iniciam a partir do pacote main,
- O exemplo a seguir está contido no pacote `main` e usa dependências dos pacotes `fmt` e `math`.

```
$ go run packages.go
My favorite number is 1
```

packages.go

```go
package main

import (
    "fmt"
    "math/rand"
)

func main() {
    fmt.Println("My favorite number is", rand.Intn(10))
}

```

Figure 3: Exemplo do uso de pacotes em Go

2.2 Variáveis

- A instrução `var` é usada para declarar uma ou mais variáveis,
- O tipo deve ser informado após o nome,
- A instrução `var` pode ser usada no escopo de pacote ou de método/função,
- A instrução `var` pode incluir inicializadores, um por variável. Neste caso, não é necessário informar o tipo pois o mesmo será inferido.

```
$ go run variables.go
0 false false false
```

variables.go

```go
package main

import "fmt"

var c, python, java bool

func main() {
    var i int
    fmt.Println(i, c, python, java)
}
```

```
$ go run variables-with-initiali
1 2 true false no!
```

variables-with-initializers.go

```go
package main

import "fmt"

var i, j int = 1, 2

func main() {
    var c, python, java = true, false, "no!"
    fmt.Println(i, j, c, python, java)
}
```

Figure 4: Exemplo do uso de variáveis em Go

2.3 Declarações curtas

- Dentro de uma função, o operador de declaração curta `:=`, pode ser usado ao invés do operador `var`,
- Ao usar o operador de declaração curta, o mesmo irá criar e inicializar a variável,
- O tipo da variável será inferido e definido na atribuição.

```
$ go run short-variable-declarations.go
1 2 3 true false no!
```

short-variable-declarations.go

```go
package main

import "fmt"

func main() {
    var i, j int = 1, 2
    k := 3
    c, python, java := true, false, "no!"

    fmt.Println(i, j, k, c, python, java)
}
```

Figure 5: Exemplo do uso de declarações curtas em Go

2.4 Constantes

- Constantes seguem a mesma regra de declaração que as variáveis, porém é utilizada o operador `const`,
- Não pode ser usado o operador de declaração curta `:=`.

```
$ go run constants.go
Hello world! Happy 3.14 Day! Go rules?
```

constants.go

```go
package main

import "fmt"
const Pi = 3.14

func main() {
    const World = "world! "
    fmt.Print("Hello ", World)
    fmt.Print("Happy ", Pi, " Day! ")

    const Truth = true
    fmt.Print("Go rules? ", Truth)
}
```

Figure 6: Exemplo do uso de constantes em Go

2.5 Funções

- Para criar uma função utilize a palavra chave `func`,
- Os parâmetros devem estar contidos entre parênteses, mesmo caso não haja parâmetros,
- As instruções da função devem estar dentro de abre e fecha chaves,

- Funções podem receber nenhum ou mais parâmetros,
- Repare que o tipo do parâmetro também vem após o nome, como na declaração de variáveis.

```
$ go run functions.go
55
```

functions.go

```
1  package main
2
3  import "fmt"
4
5  func add(x int, y int) int {
6      return x + y
7  }
8
9  func main() {
10     fmt.Println(add(42, 13))
11 }
12
```

Figure 7: Exemplo do uso de funções em Go

2.6 Múltiplos retornos de funções

- Uma função pode ter múltiplos retornos,
- Ao retornar o valor da função, devem ser informados todos os retornos,
- Os retornos e seus tipos são informados após os parênteses dos parâmetros e antes da chave de abertura das instruções. Caso haja somente um retorno, o parânteses pode ser omitido.

```
$ go run multiple-results.go
world hello
```

multiple-results.go

```
1  package main
2
3  import "fmt"
4
5  func swap(x, y string) (string, string) {
6      return y, x
7  }
8
9  func main() {
10     a, b := swap("hello", "world")
11     fmt.Println(a, b)
12 }
```

Figure 8: Exemplo do uso de funções com múltiplos retornos em Go

2.7 Estrutura de laço for

- Go só possui uma estrutura de laço, o `for`,
- Sua síntaxe é muito similar as linguagens C ou Java, exceto por não ter obrigatoriamente os parênteses e ter obrigatoriamente as chaves,
- A variável de controle do laço e seu incremento podem ser omitidos, ficando apenas a condição.

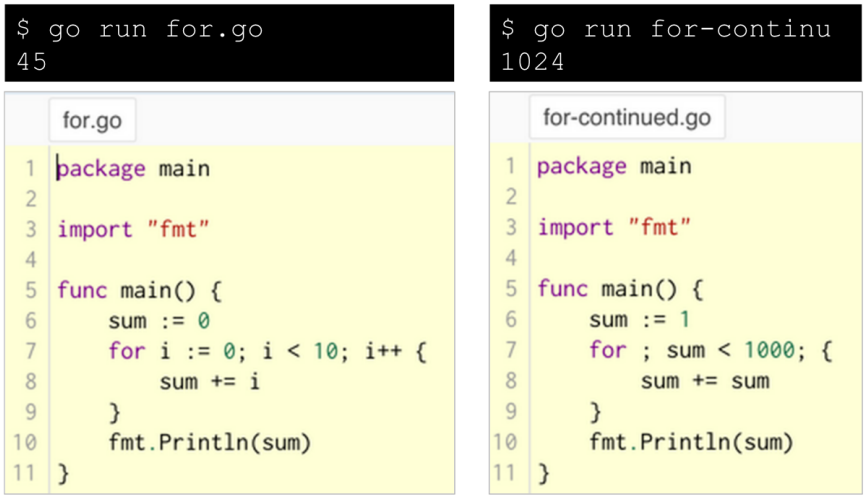


Figure 9: Exemplo do uso de laço for em Go

2.8 Estrutura de laço while

- Os pontos e vírgulas podem ser removidos juntos com as condições de início e incremento, restando apenas a condição e desta forma, temos o `while` em Go,
- Também é possível omitir todas as partes e ter um laço infinito.

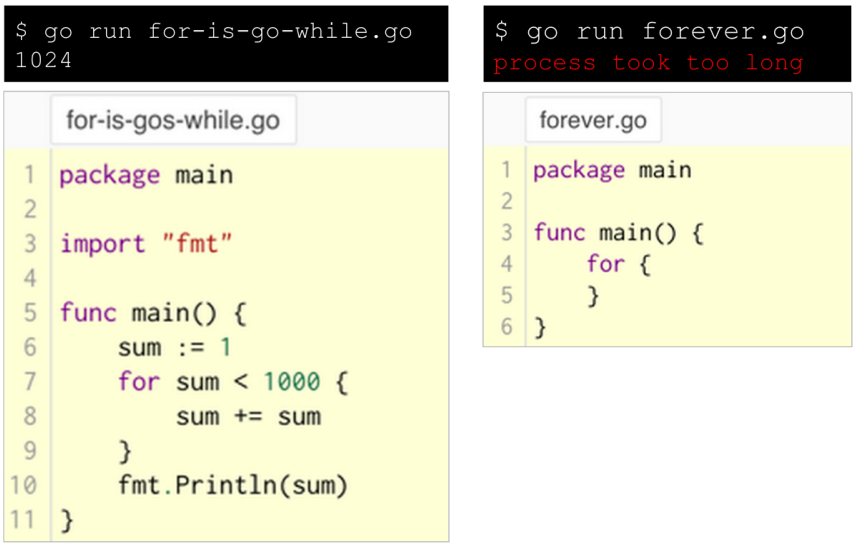


Figure 10: Exemplo do uso de laço for em Go simulando um while

2.9 Estrutura condicional if

- É muito similar com C ou Java exceto por obrigatóriamente não haver parênteses e obrigatóriamente haver chaves.

```
$ go run if.go
1.4142135623730951 2i
```

if.go

```go
package main

import (
    "fmt"
    "math"
)

func sqrt(x float64) string {
    if x < 0 {
        return sqrt(-x) + "i"
    }
    return fmt.Sprint(math.Sqrt(x))
}

func main() {
    fmt.Println(sqrt(2), sqrt(-4))
}
```

Figure 11: Exemplo do uso de if em Go

2.10 Estrutura condicional switch

- É muito similar com C ou Java exceto por obrigatóriamente não haver parênteses e obrigatóriamente haver chaves.

```
$ go run switch.go
Go runs on nacl.
```

switch.go

```go
package main

import (
    "fmt"
    "runtime"
)

func main() {
    fmt.Print("Go runs on ")
    switch os := runtime.GOOS; os {
    case "darwin":
        fmt.Println("OS X.")
    case "linux":
        fmt.Println("Linux.")
    default:
        // freebsd, openbsd,
        // plan9, windows...
        fmt.Printf("%s.", os)
    }
}
```

Figure 12: Exemplo do uso de switch em Go

2.11 Estrutura defer

- Posterga a execução de uma função, até que a função chamadora termine,
- Os argumentos passados para uma função deferida, são resolvidos no momento de sua chamada, apesar de a execução ser realizada apenas ao final.

```
$ go run defer.go
hello world
```

defer.go

```go
package main

import "fmt"

func main() {
    defer fmt.Println("world")

    fmt.Println("hello")
}
```

Figure 13: Exemplo do uso de switch em Go

2.12 Outros recursos de Go

Tivemos uma visão geral sobre Go passando pelas estruturas básicas, mas ainda há muito mais sobre esta linguagem que não iremos entrar em detalhes pois o objetivo deste trabalho não é ser um tutorial de Go e sim uma visão geral sobre a linguagem.

No entanto, segue uma lista sobre o que mais há para ver na linguagem:

- Ponteiros
- Struct
- Matrix
- Slice
- Range
- Map
- Função de valor
- Closures
- Métodos
- Interfaces
- Stringer
- Error
- E muito mais.

É disponibilizado online um passeio completo sobre cada item da linguagem em `http://go-tour-br.appspot.com`.

2.13 Web-service em Go

Para se ter ideia de como Go é simples de usar, a seguir podemos visualizar um exemplo de implementação de um **web servide** onde é disponibilizado o **endpoint** / apontando para a função `index`.

Em outras linguagens, como Java por exemplo, muito mais código, arquivos de configuração e demais seriam necessários.

```
$ go run http.go
```

```go
package main

import(
    "io"
    "net/http"
)

func index(w http.ResponseWriter, r *http.Request) {
    io.WriteString(w, "Hello world!")
}

func main() {
    http.HandleFunc("/", index)
    http.ListenAndServe(":8080", nil)
}
```

Figure 14: Exemplo de web-service em Go

3 Concorrência com Goroutines

Em Go, concorrência é feita através da execução de uma função precedida da palavra chave `go`. Neste momento, a Go Runtime se encarregará de criar processos e threads se achar necessário, além de usar a configuração da variável de ambiente `GOMAXPROCS` que informa a quantidade máxima de processos que Go pode trabalhar.

Quando executamos uma função precedido da palavra chave `go`, dizemos que temos uma `goroutine`. Pois não sabemos se será outra `thread` ou não, então apenas generalizamos como `goroutine`.

Um trunfo de Go com relação ao paralelismo e concorrência é que, a decisão por paralelizar ou não os trabalhos a serem realizados fica por conta dela, de forma que ela irá optar sempre pela melhor e mais rápida forma de execução, o que não quer dizer necessariamente que serão usados mais de um processo ou thread.

Isto pois, como é de conhecimento geral, não necessariamente o fato de se ter vários processos e threads, implica obrigatoriamente em uma execução mais rápida, isso por conta das sincronizações e escalonamentos necessários, dentre outras ações que podem ser custosas quando se fala em máxima performance.

goroutines.go

```go
package main

import (
    "fmt"
    "time"
)

func say(s string) {
    for i := 0; i < 5; i++ {
        time.Sleep(100 * time.Millisecond)
        fmt.Println(s)
    }
}

func main() {
    go say("world")
    say("hello")
}
```

```
$ go run goroutines.go
hello
world
hello
world
hello
world
hello
world
hello
```

Figure 15: Exemplo de Go routine

4 Channels

Os `channels` são como conduítes tipados pelo qual pode-se enviar e receber informações de uma goroutine a outra. Existem dois tipos de `channels`, os não buferizados (**unbuffered channels**) e os buferizados (**buffered channels**).

4.1 Unbuffered channels

`Channels` não buferizados são usados para sincronização entre duas `goroutines`. Uma vez que uma `goroutine` coloca uma informação em um channel não buferizado, ela só será liberada quando outra `goroutine` pegar esta informação. Ou seja, neste tipo de `channel`, só pode existir uma informação por vez e esta deve ser enviada e entregue em uma mesma transação.

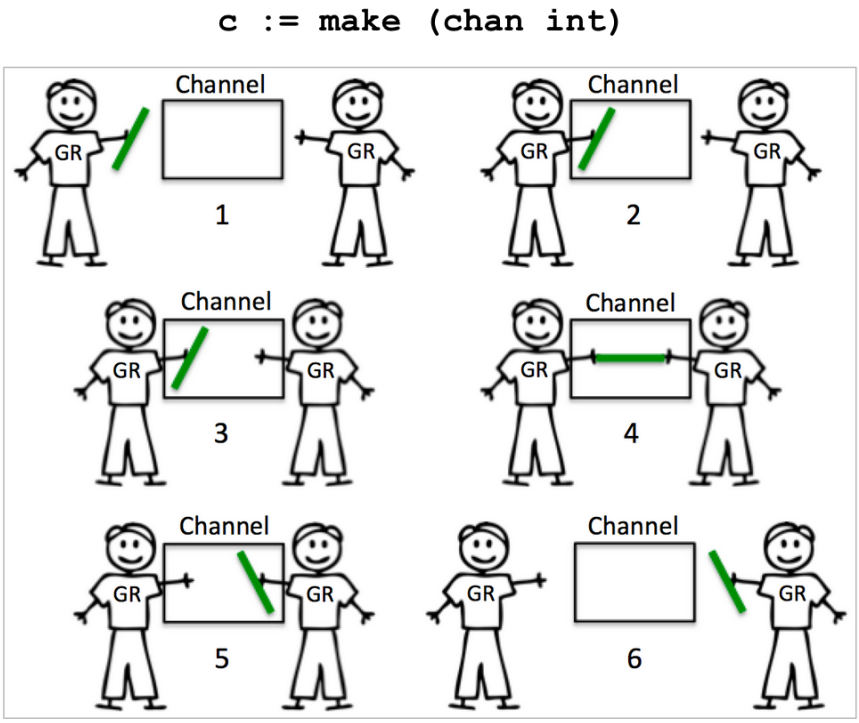


Figure 16: Exemplo de channel não buferizado em Go

4.2 Buffered channels

`Channels` buferizados são usados para envio e recebimento assíncrono de informações entre 2 ou mais `goroutines`. Buferizar um `channel` significa informar seu tamanho na sua declaração e desta forma ele poderá conter 0 ou até seu limite de informações.

Uma `goroutine` pode colocar informações em um `channel` até seu limite. Se um `channel` está cheio e uma `goroutine` tenta colocar informações nele, ela só será liberada quando liberar um espaço e puder completar seu envio.

Uma `goroutine` pode pegar informações de um `channel` enquanto ele for diferente houver, se uma `goroutine` tenta pegar uma informação porém o `channel` está vazio, ela estará bloqueada até que haja informação a ser recuperada, e então o resgate da informação é completo e a `goroutine` é liberada.

```
c := make (chan int, 10)
```

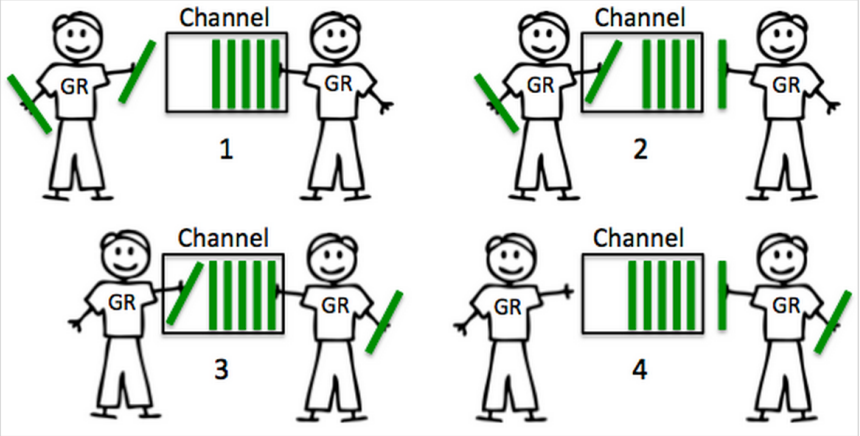


Figure 17: Exemplo de channel buferizado Go

5 Bibliografia

- http://golang.org
- http://go-tour-br.appspot.com
- https://tour.golang.org
- http://www.golangbr.org
- https://vimeo.com/49718712
- http://gophercon.com
- http://www.infoq.com/br/news/2014/09/go-1-3
- http://www.casadocodigo.com.br/products/livro-google-go
- https://pt.wikipedia.org/wiki/Inferno_(sistema_operacional)
- http://www.grokpodcast.com/series/a-linguagem-go
- https://pt.wikipedia.org/wiki/Go_(linguagem_de_programação)
- https://gobyexample.com
- http://www.goinggo.net/2014/02/the-nature-of-channels-in-go.html
- http://www.goinggo.net/2013/09/detecting-race-conditions-with-go.html?m=1
- https://en.wikipedia.org/wiki/Green_threads
- http://www.toptal.com/go/go-programming-a-step-by-step-introductory-tutorial