

Trabalho de EDL

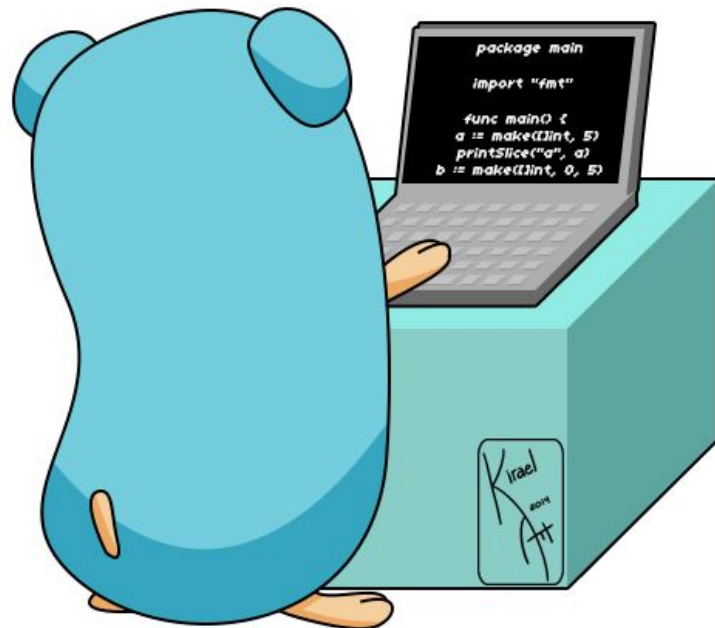
Linguagem Go/GoLang

Integrantes:

Bruno Fontes

Breno Brandão

Ingryd Moura

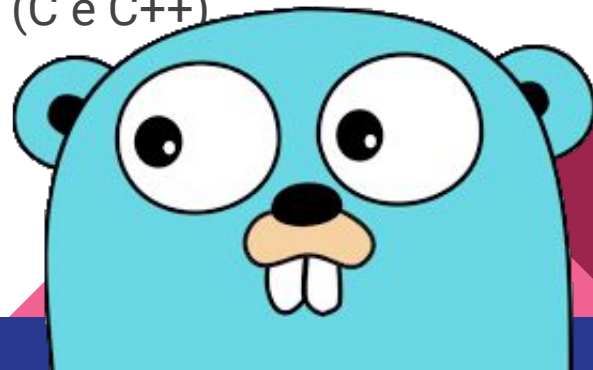


Origens/Motivação

- Criado em 2007 como um projeto interno da Google por Rob Pike, Ken Thompson e Robert Griesemer.
 - *Processadores multinúcleos*
 - *Sistemas distribuídos em redes*
 - *Modelo de Computação web*
- Open Source em 2009.

Influências

- Performance e Segurança de Linguagem Compilada (C e C++)
- Velocidade de desenvolvimento e Simplicidade (Python)
- Eficiência e Confiabilidade (Java)

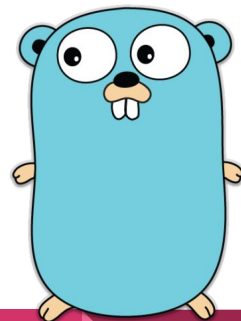


Avaliação comparar com outras linguagens

Critérios	GO	C		Critérios	GO	C
Legibilidade	Mediana	Ruim		Reusabilidade	Mediana	Mediana
Redigibilidade	Boa	Ruim		Concorrência	Sim	Não
Confiabilidade	Mediana	Ruim		Portabilidade	Sim	Não
Eficiência	Boa	Boa		Verificação de tipos	Estático/ Dinâmico	Estático
Facilidade de Aprendizado	Boa	Ruim		Método de projeto	Estruturado	Estruturado

Classificação

- Imperativa : mudança de estado
- Estática e fortemente tipada
 - Possui Mecanismo que lembra a tipagem dinâmica(!plus)
- Orientada a Objetos
 - *//do jeito Go :D*



Primitivas de Concorrência



- Goroutines

É uma função que é capaz de ser executada simultaneamente com outras funções. Os Goroutines são similares as threads em C.

- Channels

Channels em Go são utilizados para fazer com que duas Goroutines se comuniquem e para sincronizar a execução de seus códigos.

- Select

O select é similar a um switch-case, sua função é receber ou enviar dados com múltiplos canais. Combinar goroutines e canais com o select é um dos poderosos recursos da linguagem Go. O bloco select aguarda até que um de seus cases possam executar, então ele executa esse case. Ele escolhe um ao acaso se vários estiverem prontos.

Goroutines

```
import "fmt"

func f(from string) {
    for i := 0; i < 3; i++ {
        fmt.Println(from, ":", i)
    }
}

func main() {
    // Suponhamos que temos uma função de chamada `f(s)`. Aqui está
    // como nós chamaríamos ela de maneira usual, executando
    // sincronicamente.
    f("direto")

    // Para chamar essa função na goroutine, use
    // `go f(s)`. Essa nova goroutine executará
    // simultaneamente com a que foi chamada.
    go f("goroutine")

    // Você também pode iniciar uma goroutine para uma
    // função de chamada anônima.
    go func(msg string) {
        fmt.Println(msg)
    }("indo")

    // Nossas duas goroutines estão rodando de forma assíncrona
    // em goroutines separadas agora, assim a execução cai até
    // aqui. Este código `Scanln` exige que pressionamos uma
    // tecla antes de sair do programa.
    var input string
    fmt.Scanln(&input)
    fmt.Println("pronto")
}
```

```
$ go run goroutines.go
direto : 0
direto : 1
direto : 2
goroutine : 0
indo
goroutine : 1
goroutine : 2
<enter>
pronto
```

Channels

```
package main

import "fmt"

func sum(s []int, c chan int) {
    sum := 0
    for _, v := range s {
        sum += v
    }
    c <- sum // sum envia para o canal c
}

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

    c := make(chan int) //canal criado
    go sum(s[:len(s)/2], c)
    go sum(s[len(s)/2:], c)
    x, y := <-c, <-c // Recebem do c
                    //e atribui o valor para x,y

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

```
$go run channels.go
-5 17 12
```

Select

```
package main

import "fmt"

func fibonacci(c, quit chan int) {
    x, y := 0, 1
    for {
        select {
            case c <- x:
                x, y = y, x+y
            case <-quit:
                fmt.Println("quit")
                return
        }
    }
}

func main() {
    c := make(chan int)
    quit := make(chan int)
    go func() {
        for i := 0; i < 10; i++ {
            fmt.Println(<-c)
        }
        quit <- 0
    }()
    fibonacci(c, quit)
}
```

```
$go run select.go
0
1
1
2
3
5
8
13
21
34
quit
```

Módulos e Pacotes



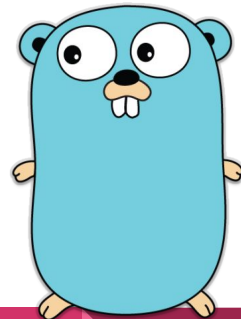
- Estrutura de pastas
 - *src* - Pastas dos projetos
 - *pkg* - Contém os arquivos objetos dos pacotes
 - *bin* - Executáveis
- Gerenciador de pacote *dep*
 - Se tornará oficial na versão 1.10
 - Resolve dependências de *imports* automaticamente
 - Comandos
 - `go get -u github.com/golang/dep/cmd/dep` - baixa e instala *dep*
 - `dep init` - inicia *dep* no projeto
 - `dep ensure` - atualiza dependências do projeto

Interface - POO e Herança

- Marca principal de orientação a objetos em go
- Para um método assinar uma interface, basta implementar todos os seus métodos, sendo possível a partir da mesma, simular o conceito de herança

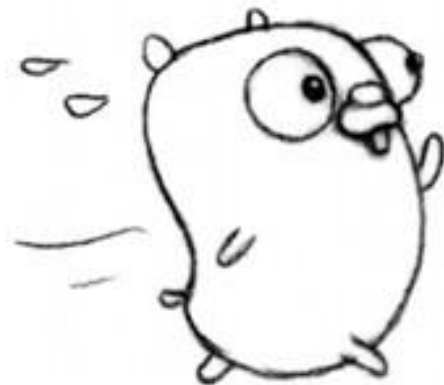
```
type familia interface {  
    dados() string  
}  
  
type pai struct {  
    nome string  
    idade int  
}  
  
func (p pai) dados() string {  
    return fmt.Sprintf("Nome: %s, Idade: %d", p.nome, p.idade)  
}  
  
type filho struct {  
    pai  
    email string  
}
```

```
func (f filho) dados() string {  
    return fmt.Sprintf("Nome: %s, Idade: %d, Email: %s", f.nome, f.idade,  
        f.email)  
}  
  
func mostraDados(membro familia) {  
    fmt.Println(membro.dados())  
}
```



Genéricos - Debate

- Em Go não existem genéricos.
- Dentre os motivos, podemos destacar o seguinte:
 - A recusa dos criadores de introduzir conceitos que compliquem a linguagem ou o compilador
- Existem inúmeros debates na comunidade, um deles, apresentado a seguir.



Genéricos - Debate

▲ zimbatm 528 days ago [-]

Generics introduce more complexity in the type system which in turn makes the compiler slower.

Generics introduce more complexity for the reader of the code because it's another abstraction to understand.

It's debatable but when your brain is thinking about generics or context-switching because it has to wait on the compiler to finish, it's less time making progress on the actual thing that needs to be done.

▲ d4rkph1b3r 528 days ago [-]

The whole point of abstractions is that you don't have to worry about as much. Generics take away complexity, that's the whole point.

▲ zimbatm 525 days ago [-]

I suppose it depends on the level of understanding that you want from your code. Generics introduce another dimension which you have to think about when you want a good level of control on allocations for example. Different data types also have different optimizations available which could be missed when blindly relying on the generic algorithm (think sorting on a fixed set for example)

- fonte do debate : *"Go should have generics"*
 - <https://news.ycombinator.com/item?id=11494181>

GO vs C - Goroutines

Em C

```
#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>

pthread_t tid[2];

void* notGoroutineClone(void* arg)
{
    char* str = arg;

    for(int i = 0; i < 3; i++)
        printf("%s %d\n", str, i);

    return NULL;
}

void* notNestedFunction(void* arg)
{
    char* str = arg;

    printf("%s\n", str);

    return NULL;
}
```

```
int main()
{
    int err;

    notGoroutineClone("direto ");

    // Temos que salvar o endereço de memória
    da thread
    pthread_create(&(tid[0]), NULL,
        &notGoroutineClone, "nao e goroutine ");
    if (err != 0)
        printf("\nThread 1 deu errado :[%s]",
            strerror(err));


    pthread_create(&(tid[1]), NULL,
        &notNestedFunction, "não é possível aninhar
        funcao em c \\_(ツ)_/");
    if (err != 0)
        printf("\nThread 2 deu errado :[%s]",
            strerror(err));

    sleep(5);
    return 0;
}
```

GO vs C - *Goroutines*

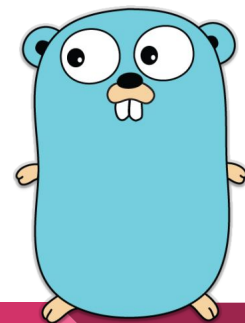
Em Go

```
package main
import "fmt"
func f(from string) {
    for i := 0; i < 3; i++ {
        fmt.Println(from, ":", i)
    }
}
func main() {
    f("direct")
    go f("goroutine")
    go func(msg string) {
        fmt.Println(msg)
    }("going")
    var input string
    fmt.Scanln(&input)
    fmt.Println("done")
}
```



Bibliografias

- <http://www.golangbr.org/doc/>
- <http://goporexemplo.golangbr.org/>
- <https://tableless.com.br/por-que-utilizar-golang-no-seu-backend/>
- <https://pt.slideshare.net/natavenancio/linguagem-go-12238181>
- <http://eltonminetto.net/post/2017-07-28-gerenciando-dependencias-golang/>
- <https://github.com/golang/dep/issues/847>
- <https://code.tutsplus.com/pt/tutorials/lets-go-object-oriented-programming-in-golang-cms-26540>
- <https://www.ime.usp.br/~gold/cursos/2015/MAC5742/reports/GoLang.pdf>
- https://medium.com/@ViniciusPach_97728/go-composi%C3%A7%C3%A3o-vs-heran%C3%A7a-2e8b78928c26



FIM



GO