

- Go Lang

BRUNO FREIRE
VINÍCIUS TEIXEIRA



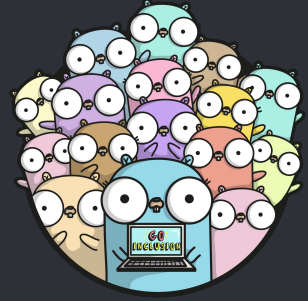
ORIGENS

“

Com problemas para desenvolver sistemas para trabalhar com sua infraestrutura, o Google criou em 2007 a linguagem Go!



PROBLEMAS



- Tecnologia avançada porém complexa
- Programas com milhares de linhas
- Milhares de engenheiros trabalhando simultaneamente
- Processos de Merge e Compilação tornavam-se muito demorados

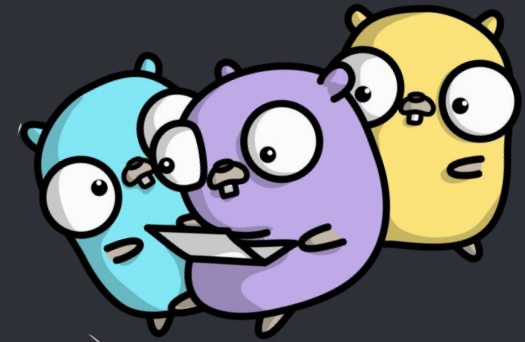
Precisavam de uma linguagem que atendesse todas as necessidades



● CRIADORES

○ Assim nasceu a Go! em 2007 pelos engenheiros:

- Rob Pike
- Ken Thompson
- Robert Griessemer



Em 2009, Go se tornou Open Source e em 2010, começou a ser adotada por desenvolvedores de fora do Google.

INFLUÊNCIAS E CARACTERÍSTICAS

- Pascal
- C
- Oberon
- Newsqueak
- Limbo



- Imperativa
- Orientada a Objetos (Sem herança)
- Compilada (Goroutines)
- Estaticamente tipada
- Fortemente tipada
- Garbage-Collected
- Sintaxe simples
- Excelente pacote padrão
- Multiplataforma
- Concorrente
- Suporte a Closures
- Dependências explícitas
- Funções com múltiplos retornos
- Ponteiros



EXPRESSIVIDADE DE GO (GOROUTINE)

Go!

Foram criadas N goroutines, cada um aguarda um número em seu canal de entrada, adiciona 1 a ele e o envia para a saída.

O Programa em Go é 4.3x mais rápido que C#. A versão de C# que trabalha com tarefas é 2.05x mais rápida e ainda assim é 2x mais lenta que Go. .

```
File Edit Selection View Go Debug parallelism.go - Untitled (Workspace) - Visual St...
parallelism.go x parallelism.cs
1 package main
2
3 import (
4     "flag"
5     "fmt"
6     "time"
7 )
8
9 func measure(start time.Time, name string) {
10     elapsed := time.Since(start)
11     fmt.Printf("%s took %s", name, elapsed)
12     fmt.Println()
13 }
14
15 var maxCount = flag.Int("n", 500000, "how many")
16
17 func f(output, input chan int) {
18     output <- 1 + <-input
19 }
20
21 func test() {
22     fmt.Printf("Started, sending %d messages.", *maxCount)
23     fmt.Println()
24     flag.Parse()
25     defer measure(time.Now(), fmt.Sprintf("Sending %d messages", *maxCount))
26     finalOutput := make(chan int)
27     var left, right chan int = nil, finalOutput
28     for i := 0; i < *maxCount; i++ {
29         left, right = right, make(chan int)
30         go f(left, right)
31     }
32     right <- 0
33     x := <-finalOutput
34     fmt.Println(x)
35 }
36
37 func main() {
38     test()
39     test()
40 }
41
```

Retorno
Started, sending 1000000 messages.
1000000
Sending 1000000 messages took 3.5034779s
Started, sending 1000000 messages.
1000000
Sending 1000000 messages took 808.9572ms

CONCORRÊNCIA (THREAD)



C#

Esse teste “pré-criado” para o Go-in C# normalmente não precisa de canais para tarefas assíncronas se comunicarem. As tarefas normalmente chamam umas às outras e aguardam assincronamente por um resultado. No entanto, a única opção que o Go tem para a comunicação entre goroutines é o canal.

O .NET emite código de método na invocação, ou seja, a primeira execução de qualquer função “pequena” leva muito mais tempo.

```
File Edit Selection View Go Debug parallelism.cs - Untitled (Workspace) - Visual Stu...
parallelism.go parallelism.cs x
1
2 using System;
3 using System.Diagnostics;
4 using System.Linq;
5 using System.Threading.Tasks;
6 using System.Threading.Tasks.Channels;
7
8 namespace ChannelsTest
9 {
10     class Program
11     {
12         public static void Measure(string title, Action<int, bool> test,
13         {
14             test(warmupCount, true); // Warmup
15             var sw = new Stopwatch();
16             GC.Collect();
17             sw.Start();
18             test(count, false);
19             sw.Stop();
20             Console.WriteLine($"{title}: {sw.Elapsed.TotalMilliseconds:0.0}");
21         }
22
23         static async void AddOne(WritableChannel<int> output, ReadableChannel<int> input)
24         {
25             await output.WriteAsync(1 + await input.ReadAsync());
26         }
27
28         static async Task<int> AddOne(Task<int> input)
29         {
30             var result = 1 + await input;
31             await Task.Yield();
32             return result;
33         }
34
35         static void Main(string[] args)
36         {
37             if (!int.TryParse(args.FirstOrDefault(), out var maxCount))
38                 maxCount = 1000000;
39             Measure($"Sending {maxCount} messages (channels)", (count, isWarmup) =>
40             {
41                 var firstChannel = Channel.CreateUnbuffered<int>();
42                 var output = firstChannel;
43                 for (var i = 0; i < count; i++) {
44                     var input = Channel.CreateUnbuffered<int>();
45                     AddOne(output.Out, input.In);
46                     output = input;
47                 }
48                 output.Out.WriteAsync(0);
49                 if (!isWarmup)
50                     Console.WriteLine(firstChannel.In.ReadAsync().Result);
51             }, maxCount);
52             Measure($"Sending {maxCount} messages (Task<int>)", (count, isWarmup) =>
53             {
54                 var tcs = new TaskCompletionSource<int>();
55                 var firstTask = AddOne(tcs.Task);
56                 var output = firstTask;
57                 for (var i = 0; i < count; i++) {
58                     var input = AddOne(output);
59                     output = input;
60                 }
61                 tcs.SetResult(-1);
62                 if (!isWarmup)
63                     Console.WriteLine(output.Result);
64             }, maxCount);
65         }
66     }
67 }
```

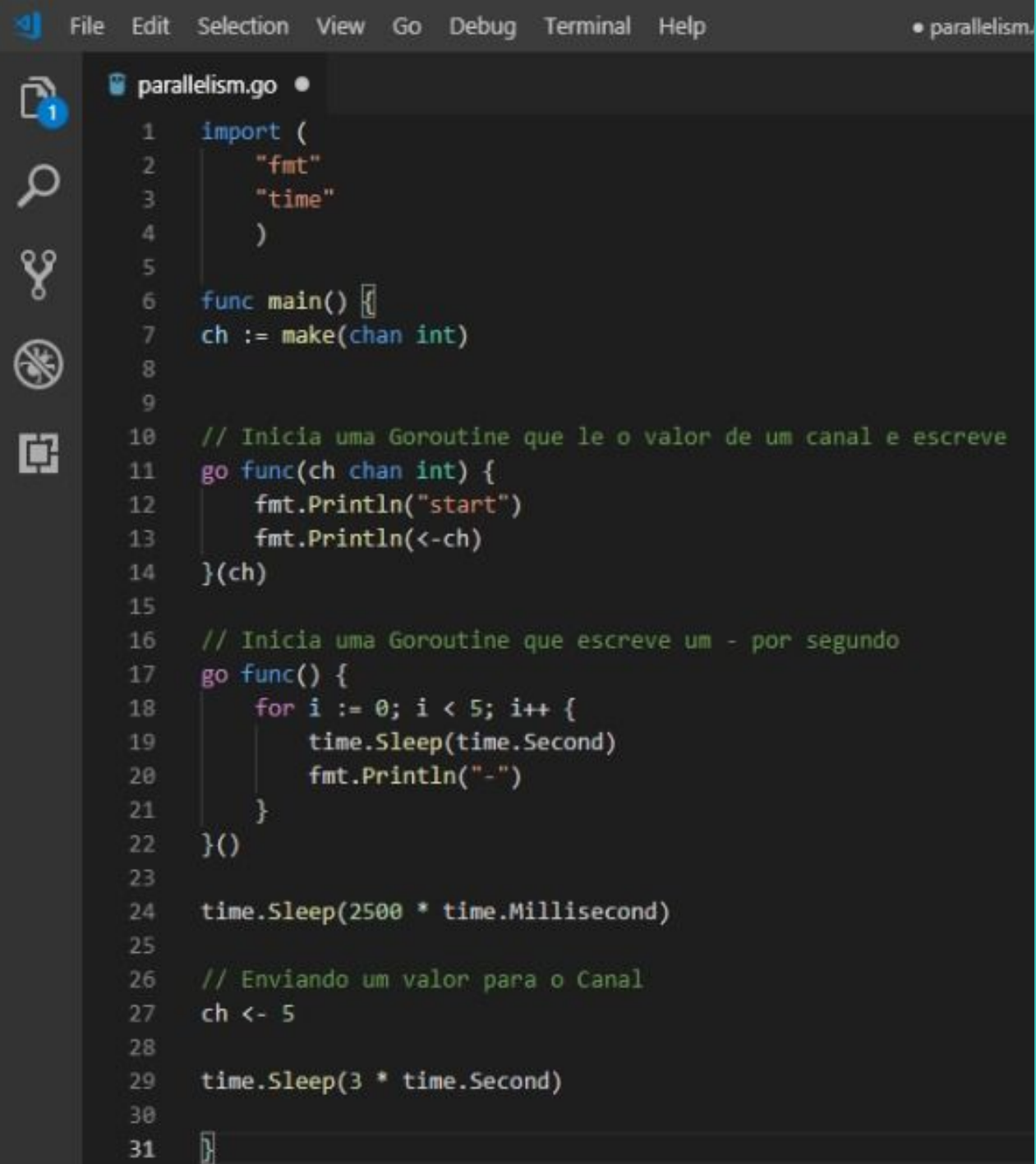
```
Retorno
1000000
Sending 1000000 messages (channels): 3545.006ms
1000000
Sending 1000000 messages (Task): 1693.675ms
```




Go!

Canais possuem, por padrão um estado bloqueante. Isso significa que se enviarmos um valor a um Canal, ele será bloqueado até que o canal seja recebido, assim como será bloqueado quando o Canal for recebido, até que alguém envie um novo valor para o Canal.

Esse comportamento liga fortemente remetente e receptor, isso pode ser um comportamento indesejado, algumas vezes. A linguagem do Google apresenta algumas alternativas.



```
File Edit Selection View Go Debug Terminal Help • parallelism.

parallelism.go •
1 import (
2     "fmt"
3     "time"
4 )
5
6 func main() {
7     ch := make(chan int)
8
9
10    // Inicia uma Goroutine que le o valor de um canal e escreve
11    go func(ch chan int) {
12        fmt.Println("start")
13        fmt.Println(<-ch)
14    }(ch)
15
16    // Inicia uma Goroutine que escreve um - por segundo
17    go func() {
18        for i := 0; i < 5; i++ {
19            time.Sleep(time.Second)
20            fmt.Println("-")
21        }
22    }()
23
24    time.Sleep(2500 * time.Millisecond)
25
26    // Enviando um valor para o Canal
27    ch <- 5
28
29    time.Sleep(3 * time.Second)
30
31 }
```




Go!

EXPRESSIVIDADE DE GO (SELECT E BUFFER)

```
Selection View Go Debug Terminal Help • parallelism.go - Untitled

parallelism.go •
func main() {
    r := rand.New(rand.NewSource(time.Now().UnixNano()))

    sum := func(a int, b int) <-chan int {
        ch := make(chan int)
        go func() {
            // Random time up to one second
            delay := time.Duration(r.Int()%1000) * time.Millisecond
            time.Sleep(delay)
            ch <- a + b
            close(ch)
        }()
        return ch
    }

    // Call sum 4 times with the same parameters
    ch1 := sum(3, 5)
    ch2 := sum(3, 5)
    ch3 := sum(3, 5)
    ch4 := sum(3, 5)

    // wait for the first goroutine to write to its channel
    select {
    case result := <-ch1:
        fmt.Printf("ch1: 3 + 5 = %d", result)
    case result := <-ch2:
        fmt.Printf("ch2: 3 + 5 = %d", result)
    case result := <-ch3:
        fmt.Printf("ch3: 3 + 5 = %d", result)
    case result := <-ch4:
        fmt.Printf("ch4: 3 + 5 = %d", result)
    }
}
```

```
File Edit Selection View Go Debug Terminal Help • parallelism.go

parallelism.go •
2  "fmt"
3  "time"
4  )
5
6  func main() {
7      ch := make(chan int, 3)
8
9      // Start a goroutine that reads a value from the channel
10     go func(ch chan int) {
11         for {
12             time.Sleep(time.Second)
13             fmt.Printf("Goroutine received: %d\n", <-ch)
14         }
15     }(ch)
16
17     // Start a goroutine that prints a dash every second
18     go func() {
19         for i := 0; i < 5; i++ {
20             time.Sleep(time.Second)
21             fmt.Println("-")
22         }
23     }()
24
25     // Push values to the channel as fast as possible
26     for i := 0; i < 5; i++ {
27         ch <- i
28         fmt.Printf("main() pushed: %d\n", i)
29     }
30
31     // Sleep five more seconds to let all goroutines finish
32     time.Sleep(5 * time.Second)
33
34
35
36 }
```



● ALGUMAS EMPRESAS QUE UTILIZAM GO!



- Amazon
- Google
- Globo.com
- Mozilla Foundation
- YouTube
- Vimeo
- Soundcloud
- Docker
- Nokia
- Heroku
- Bitly

