Паттерны Go-программ

Автор: Денис Лимарев

ПРОЛОГ

Вопросы:

1. Зачем переходить на Go?
2. В чем польза использования Go? //
3. Почему программы на Go быстрее? // ГДЕ бенчмарки ??)
4. Зачем мне учить Go?

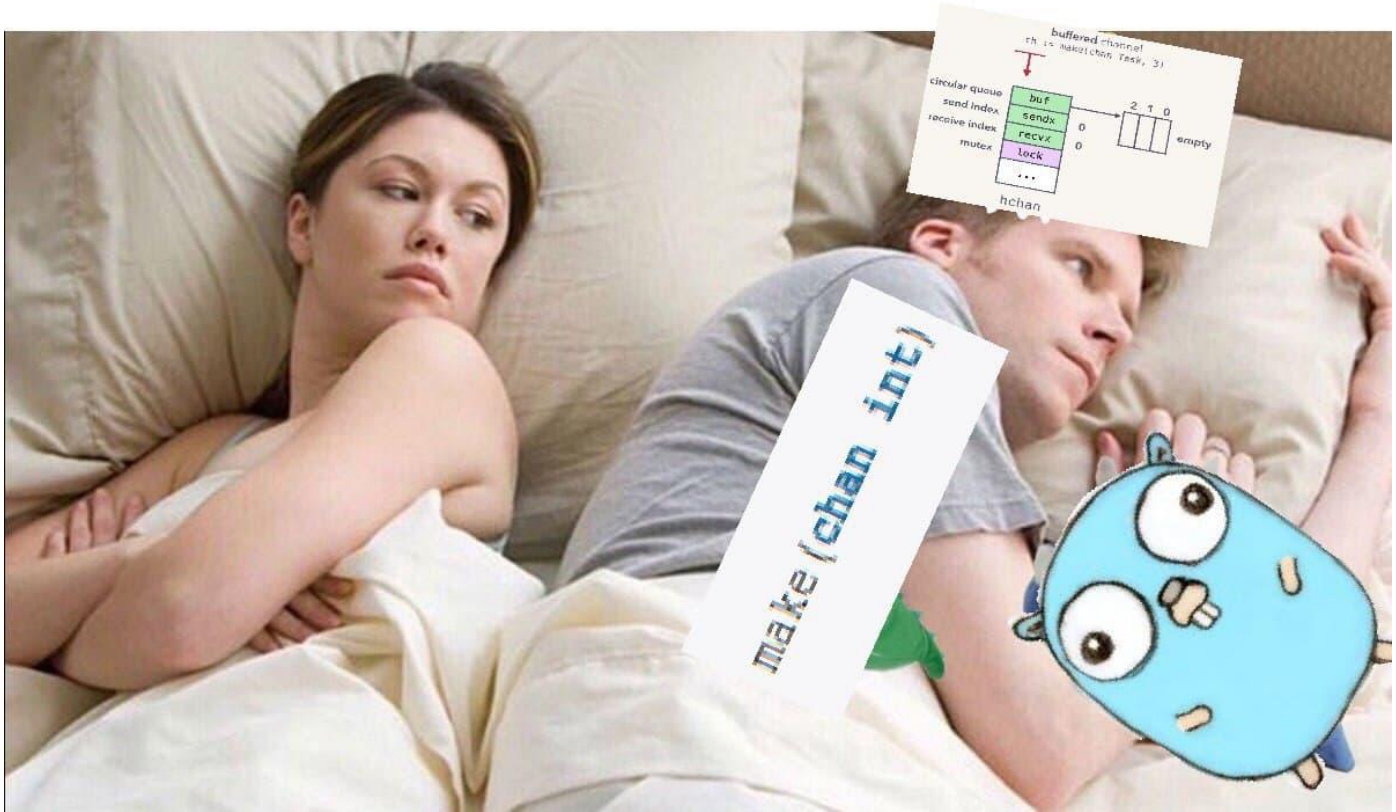
КАНАЛЫ

Канал - встроенная структура Go, с помощью которой можно получать и передавать значения.

```
package main

import "fmt"

func main() {

    var messageCh = make(chan string, 1)

    messageCh <- "42"

    fmt.Println(<-messageCh)

}
```

она: опять о своих каналах думает

он: каналы мои каналы



ГОРУТИНЫ

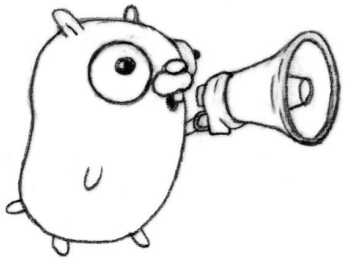
Горутина - легковесный тред управляемый окружением рантайма Go.

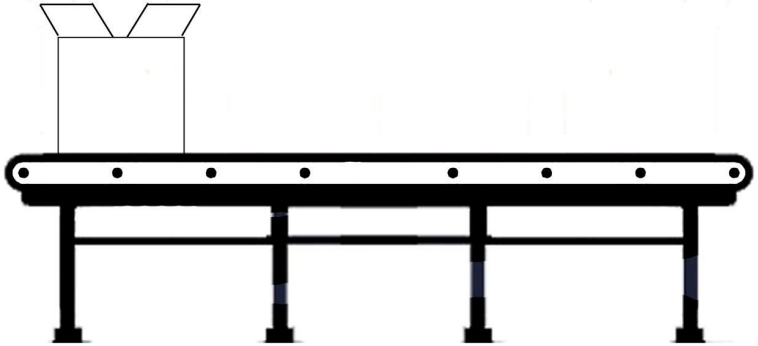
Для выполнения функции в горутине достаточно в начале вызова написать ключевое слово go.

```
package main

import "fmt"

func say(s string) { fmt.Println(s) }

func main() {

    go say("world")

    say("hello")

}
```

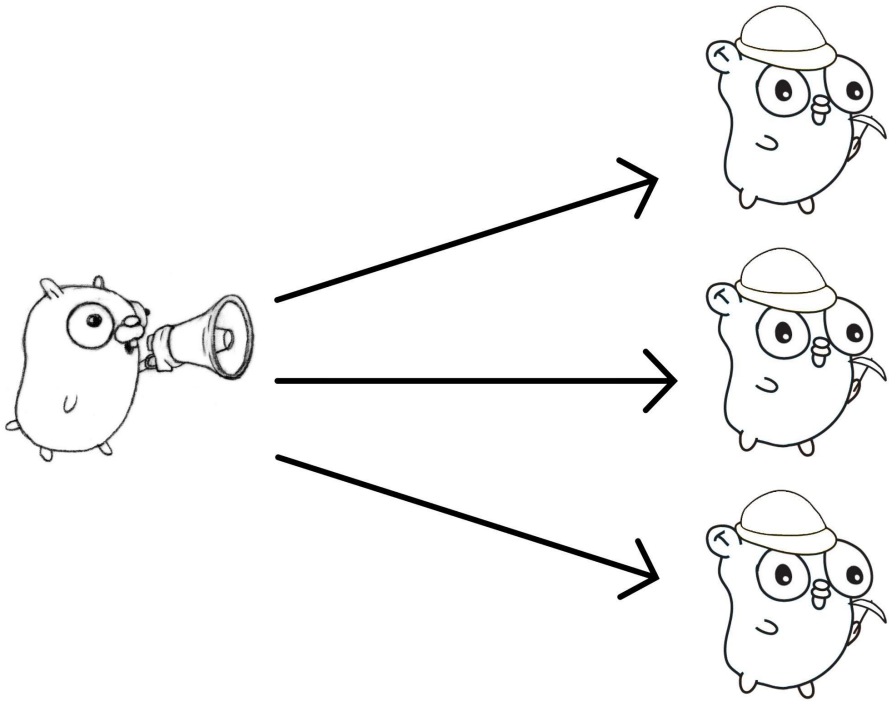
```go
package main

import (
	"fmt"
	"sync"
)

func main() {
	var messageCh = make(chan string, 42)
	var wg = &sync.WaitGroup{}

	wg.Add(1)
	go func() {
		defer wg.Done()
		messageCh <- "ping"
	}()

	wg.Add(1)
	go func() {
		defer wg.Done()
		messageCh <- "pong"
	}()

	wg.Wait()
	close(messageCh)

	for message := range messageCh {
		fmt.Println(message)
	}
}
```

ГЕНЕРАТОР

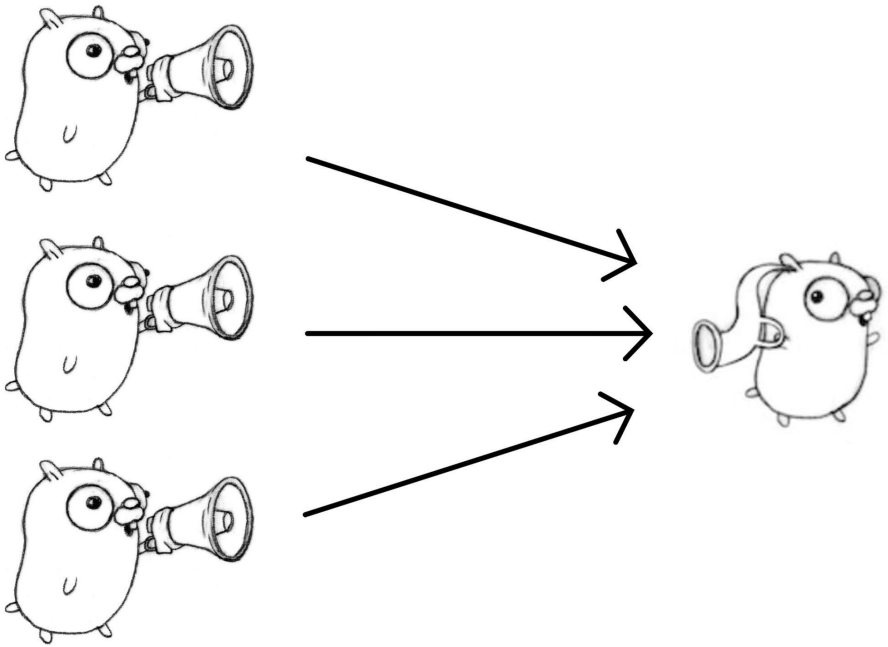




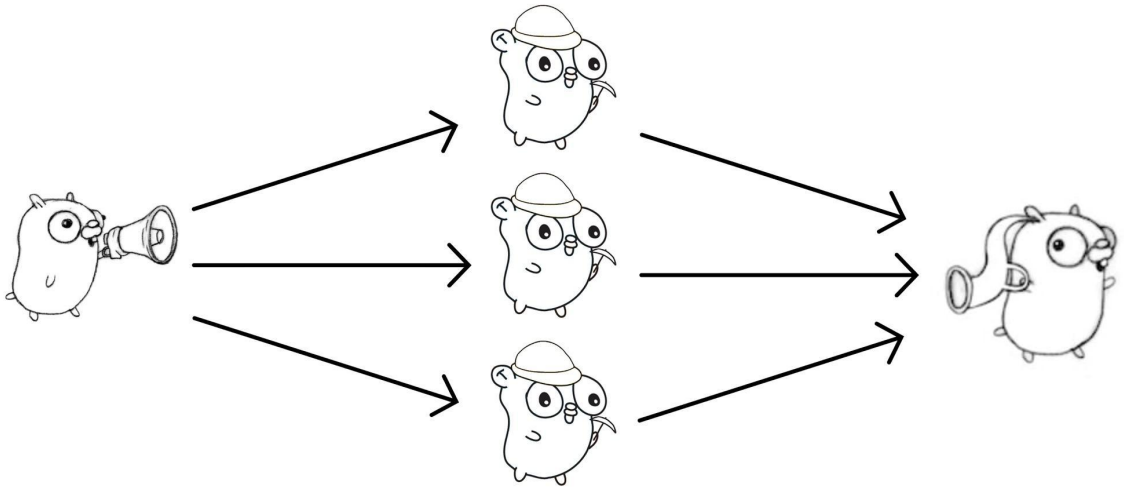
FAN-OUT



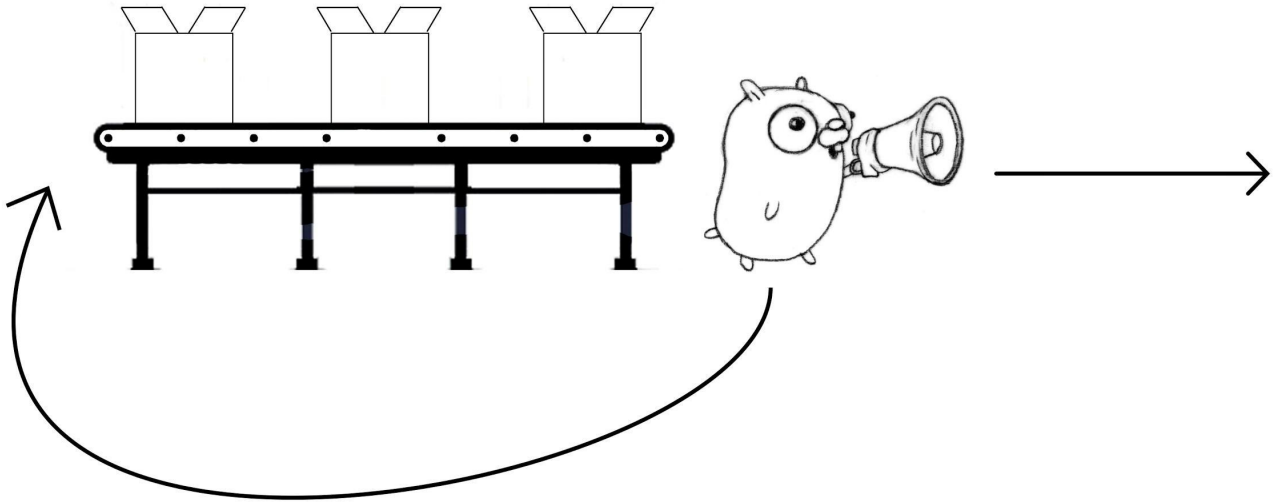
FAN-IN



КОНВЕЙЕР



OBJECT POOL



Примеры действующих Go-репозиториев

1. https://github.com/peakle/goszakupki-parser (консольная команда)
2. https://github.com/wakeapp/go-asf (Go библиотека)
3. https://github.com/wakeapp/go-sql-generator (Go библиотека)

СПАСИБО ЗА ВНИМАНИЕ!

```go
// Handle - entry point for commission_export command
func Handle(_ *cli.Context) error {
    log.Println("Start time: ", time.Now().Format("2006-01-02 15:04"))

    var commissionCh = make(chan *commission, 1000)
    var commissionIDCh = make(chan string, 1000)
    var doneCh = make(chan struct{}, 1)

    var wg = &sync.WaitGroup{}
    var workerWg = &sync.WaitGroup{}

    var cookies, err = getAffiliatePlayerCookie()
    if err != nil {
        return fmt.Errorf("on Handle commissions-export: %s", err.Error())
    }

    wg.Add(1)
    go fillCommissionUpsert(commissionCh, doneCh, wg)
    go commissionIDGenerator(cookies, commissionIDCh)

    for i := 0; i < consts.DefaultWorkerCount; i++ {
        workerWg.Add(1)
        go parseCommissions(cookies, commissionIDCh, commissionCh, workerWg)
    }

    workerWg.Wait()
    close(commissionCh)

    doneCh <- struct{}{} // release fillCommissionUpsert
    wg.Wait()

    log.Println("End time: ", time.Now().Format("2006-01-02 15:04"))

    return nil
}
```

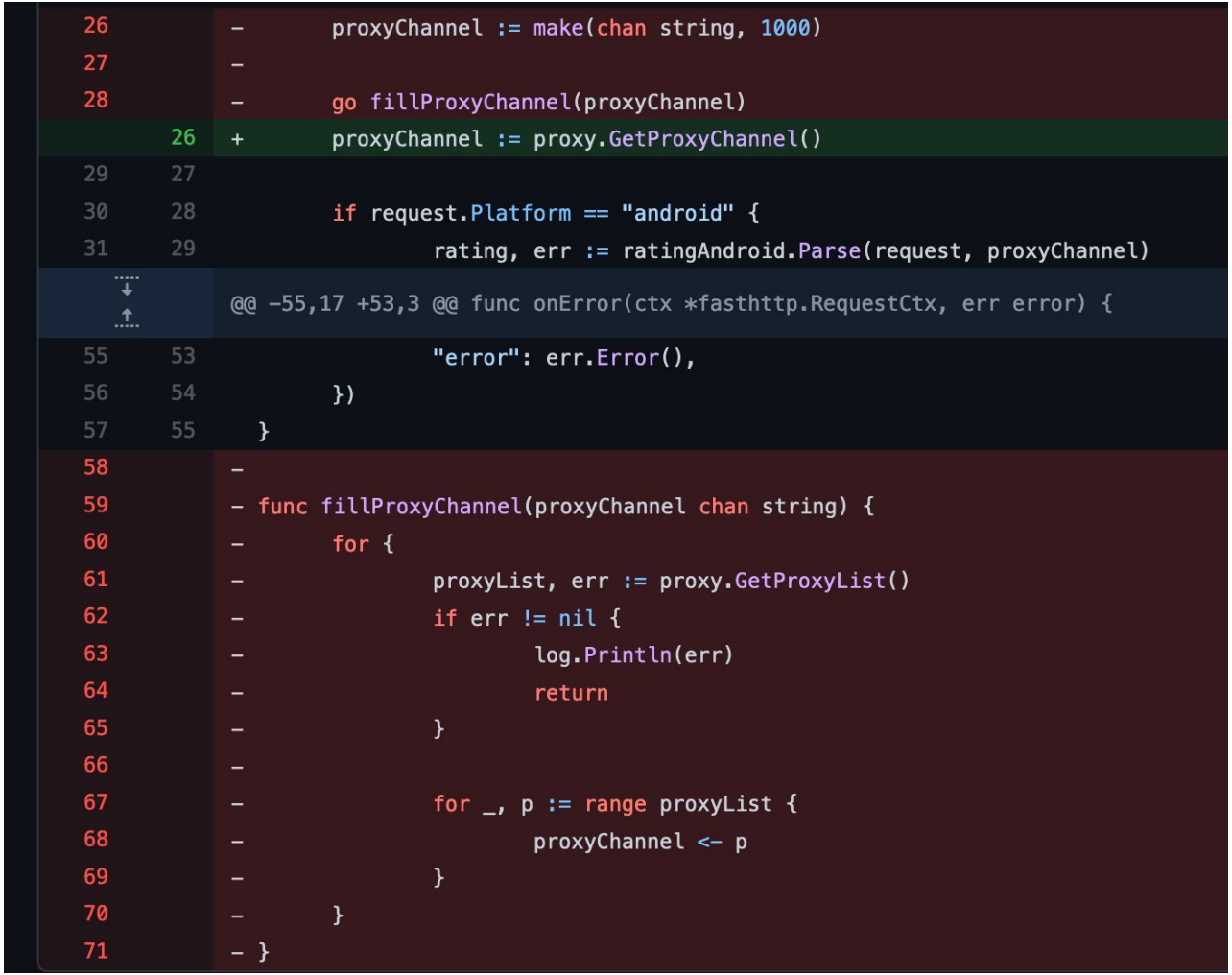
ОШИБКИ НЕДЕЛИ

```
var resultSaverWg sync.WaitGroup
var saverWg sync.WaitGroup
var workerWg sync.WaitGroup
var resultSaverWg = sync.WaitGroup{}
var saverWg = sync.WaitGroup{}
var workerWg = sync.WaitGroup{}
```

```
var wg *sync.WaitGroup

wg.Add(1)
go fillCommissionUpsert(commissionCh, doneCh, wg)
```

```
var idList map[string]string
idList["1"] = "1"
```

```
var idList = make([]string, 10)
idList = append(idList, "1")
```


26 - proxyChannel := make(chan string, 1000)
27 -
28 - go fillProxyChannel(proxyChannel)
26 + proxyChannel := proxy.GetProxyChannel()
29 27
30 28 if request.Platform == "android" {
31 29 rating, err := ratingAndroid.Parse(request, proxyChannel)
@@ -55,17 +53,3 @@ func onError(ctx *fasthttp.RequestCtx, err error) {
55 53 "error": err.Error(),
56 54 })
57 55 }
58 -
59 - func fillProxyChannel(proxyChannel chan string) {
60 - for {
61 - proxyList, err := proxy.GetProxyList()
62 - if err != nil {
63 - log.Println(err)
64 - return
65 - }
66 -
67 - for _, p := range proxyList {
68 - proxyChannel <- p
69 - }
70 - }
71 - }

ЭПОС ОДНОЙ ОШИБКИ

```go
func saveWorker(c <-chan parsedSearchText, country enum.Country, lang string, wg *sync.WaitGroup) {
    defer wg.Done()

    var (
        emptyParsedSearchText parsedSearchText
        pst                   parsedSearchText
    )
    var index int

    psts := make([]parsedSearchText, 0, suggestChunkSize)
    for pst = range c {
        psts = append(psts, pst)

        if len(psts) >= suggestChunkSize {
            insertSuggestBuffer(psts, lang)

            for index = range psts {
                psts[index] = emptyParsedSearchText
            }
            psts = psts[:0]
        }
    }

    if len(psts) > 0 {
        insertSuggestBuffer(psts, lang)
    }
}
```

```go
func saveWorker(c <-chan parsedSearchText, country enum.Country, lang string, wg *sync.WaitGroup) {
    defer wg.Done()

    var (
        emptyParsedSearchText parsedSearchText
        pst                   parsedSearchText
    )
    var index int

    var timer = time.NewTimer(1 * time.Minute)
    defer timer.Stop()

    psts := make([]parsedSearchText, 0, suggestChunkSize)
    select {
    case <-timer.C:
        if len(psts) > 0 {
            insertSuggestBuffer(psts, lang)

            for index = range psts {
                psts[index] = emptyParsedSearchText
            }
            psts = psts[:0]
        }
    case pst = <-c:
        psts = append(psts, pst)

        if len(psts) >= suggestChunkSize {
            insertSuggestBuffer(psts, lang)

            for index = range psts {
                psts[index] = emptyParsedSearchText
            }
            psts = psts[:0]
        }
    }

    if len(psts) > 0 {
        insertSuggestBuffer(psts, lang)
    }
}
```

```go
func saveWorker(c <-chan parsedSearchText, country enum.Country, lang string, wg *sync.WaitGroup, doneCh chan struct{}) {
    defer wg.Done()

    var pst parsedSearchText

    var timer = time.NewTimer(1 * time.Minute)
    defer timer.Stop()

    var psts = make([]parsedSearchText, 0, suggestChunkSize)

breakLoop:
    for {
        select {
        case <-timer.C:
            if len(psts) > 0 {
                insertSuggestBuffer(psts, lang)
                psts = psts[:0]
            }
        case pst = <-c:
            psts = append(psts, pst)

            if len(psts) >= suggestChunkSize {
                insertSuggestBuffer(psts, lang)
                psts = psts[:0]
            }
        case <-doneCh:
            break breakLoop
        }
    }

    if len(psts) > 0 {
        insertSuggestBuffer(psts, lang)
    }
}
```

```go
func saveWorker(c <-chan parsedSearchText, country enum.Country, lang string, wg *sync.WaitGroup, doneCh chan struct{}) {
    defer wg.Done()

    var pst parsedSearchText

    var timer = time.NewTimer(5 * time.Minute)
    defer timer.Stop()

    var psts = make([]parsedSearchText, 0, suggestChunkSize)

    for {
        select {
        case <-timer.C:
            if len(psts) > 0 {
                insertSuggestBuffer(psts, lang)
                psts = psts[:0]
            }
        case pst = <-c:
            psts = append(psts, pst)

            if len(psts) >= suggestChunkSize {
                insertSuggestBuffer(psts, lang)
                psts = psts[:0]
            }
        case <-doneCh:
            if len(psts) > 0 {
                insertSuggestBuffer(psts, lang)
            }

            return
        }
    }
}
```