Язык программирования



расскажет Михаил В. Шохирев

Клуб программистов Шадринск 2024-2025

О чём поGОворим



История: кем, когда, где и как создавался язык.

Цели: зачем создавался язык. Технология разработки.

Особенности: чем Go отличается от других языков. Корни языка.

Компиляция (для разных платформ) и выполнение.

Синтаксис: правописание и стиль. Управляющие конструкции. Данные.

Модульность: функции, методы. Пакеты. Объектное программирование.

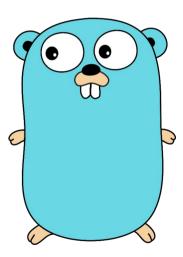
Интерфейсы: типы для действий (contracts), ограничения (constraints): generics.

Мно*Go*задачность: concurrency (goroutines, channels).

Инструменты: go command. Go tools. IDE и редакторы.

Применение: где, как и почему лучше использовать Go.

Критика: недостатки Go и альтернативы ему.



Шустрый суслик **Gopher**

(asmop: Renée French)

Go = "C for the 21st century"



Go — простой быстро компилируемый многопоточный язык программирования со статической типизацией, ориентированный на высокопроизводительную работу в сети и эффективное многопоточное выполнение (в виде "родных" исполняемых файлов), легко осваиваемый, с многочисленными надёжными *стандартными* библиотеками.

Проектировщики:

Robert Griesemer (разработчик V8 JS engine, Java HotSpot VM, Sawzall; его учитель – Niklaus Wirth)

Rob Pike (разработчик window system for Unix; Plan 9, Inferno, UTF-8, языка Sawzall)

Ken Thompson (разработчик Unix, C, grep, ed, Plan 9, Inferno, UTF-8)

Разработчики:

Команда в Google + Go community.

Цель: <u>система программирования</u> для разработки больших надёжных высоконагруженных быстро работающих серверных программных комплексов с распараллеливанием выполнения, которые будут развиваться в течение длительного времени большой командой разработчиков.

Ha проектирование Go **повлияли языки** C, Oberon-2, Active Oberon, Oberon, Modula-2, Modula, Pascal, Alef, Newsqueak, Squeak, CSP, Smalltalk, Limbo, APL, BCPL, occam.

Разработчиков первоначально объединило их общее недовольство языком С++. Кроме того, они хотели сделать язык с простым синтаксисом, но отвечающий современным требованиям к разработке программ.

Go: creators

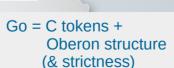




Ken Thompson, США («старая инженерная школа»): разработчик языка С, ОС Unix, Plan 9, Inferno, grep, ed, QED, UTF-8.



Rob Pike, Канада («следующее поколение», специалист по сопситепсу): разработчик window system for Unix; ОС Plan 9, Inferno, UTF-8, sam, acme, языков Sawzall, Limbo, Newsqueak.





Robert Griesemer,
Швейцария
(«ученик Никласа
Вирта (Pascal,
Modula, Oberon),
европейская
школа»):
разработчик V8 JS
engine, Java
HotSpot VM, языка
Sawzall, а
programming
language for vector
computers,
cucmeмы Strongtalk.



Rob Pike



Go: цели создания языка



Разработчики хотели не просто создать новый язык, но разработать лучшую <u>технологию</u> разработки программ (большой командой разработчиков в течение длительного времени):

- стабильный язык: совместимость с предыдущими и последующими версиями:
- язык спроектирован для надёжного программирования;
- краткий и логичный синтаксис: легко освоить и однозначно понимать в команде;
- строгая типизация, объявления, импорты: компилятор контролирует программистов;
- быстрый компилятор: минимизирует время сборки больших программных систем;
- интерфейсы с неявным соответствием: позволит расширять готовые системы;
- композиция вместо наследования: обеспечит независимое развитие компонентов;
- легковесные goroutine-ы: структурируют программу для параллельного выполнения;
- каналы обеспечат удобную синхронизацию и обмен данными между процессами;
- тип данных error: даёт все средства языка для явной обработки ошибок;
- единый стиль оформления исходников: задаётся утилитой **go fmt**;
- богатая и надёжно работающая стандартная библиотека: предоставит готовые компоненты;
- удобная система подключения внешних пакетов с идентификацией по URL;
- мощный набор стандартных инструментов: go command, go tool command;
- возможность автоматизации за счёт расширения набора инструментов;
- открытый исходный код: привлекает сообщество для развития системы программирования.

Хронология и совместимость



2007 .09	началась разработка Go в компании Google; проектированием занимались: Robert Griesemer, Rob Pike и Ken Thompson (~ в течение 1 года).
2009 .11.10	был официально представлен язык Go.
2011.03.16	go r56: based on release weekly.2011-03-07.1
2012 .03.28	go1.0: language & a set of core libraries.
2013.05.13	go1.1: ~30%-40% performance improvementof compiled code.
2015 .08.19	go1.5: compiler & runtime written entirely in Go (with a little assembler).
2017.08.24	go1.9: type aliases.
2020.02.25	go1.14: Go modules.
2022 .03.15	go1.18: generics.
2023.08.08	go1.21: min, max, clear built-in functions.
2024 .10.01	go1. 23 .2: range over function types; <i>iter, unique</i> packages.
2025.02.11	go1.24.0: generic type aliases; runtime have decreased CPU overheads by 2–3%.
2025.04.01	go1.24.2: security & bug fixes.
2025 .08	go1. 25 (upcoming release).

Спецификация языка и стандартной библиотеки обратно совместимы с версиями Go 1.х.

Поэтому многие крупные компании, выждав время, убедились в долговременной поддержке языка и стали применять его в своих важных проектах.

Распространение



Go применяется в (> 40% IT technology companies worldwide):

Alibaba, Amazon, American Express, **Apple**, Armut $(C\# \to Go)$, **Baidu**, BBC, bitly, Canonical, Capital One, CERN, Cloudflare, Cockroach, DataDog, Docker, DropBox $(Python \to Go)$, GitHub, **Google**, gov.uk, **IBM**, InfluxDB, Intel, K8s, Kubernetes, **Meta**, **Microsoft**, Monzo, **Mozilla** (Rust & Go), Netflix $(Java \to Go)$, New York Times, **Oracle**, PayPal, Pinterest, Reddit, Slack, Salesforce $(Python, C \to Go)$, SendGrid, Stream $(Python \to Go)$, SoundCloud, Terraform, The Economist, Twitch, **Twitter**, Uber, Walmart, YouTube, *многих других организациях и проектах open-source*.

В России (всеми крупными компаниями):

Яндекс, ЦУМ, УГМК-Телеком, Точка, Т-Банк, Совкомбанк Технологии, СберТех, Ростелеком, Онлайн-кинотеатр Иви, МТС, Магнит. Тесh, Лаборатория Касперского, ИТ-Холдинг Т1, ИнГосСтрах Банк, Домклик, Группа Астра, ГНИВЦ, АйТи Инновация, YADRO, X5 Digital, VK, Viasat Tech, Tutu, Tele2, Selectel, S8. Capital, Ozon, Okko, Mail.ru, Lamoda Tech, iSpring, IBS, Cloud.ru, Boxberry: IT, Beeline, Avito, 2GIS и многие другие...

Популярность: рейтинги



```
TIOBE index ▲ (since 2009):
                                                                       Is Golang Still Growing?
Go Language Popularity
Trends in 2024
    Now: #7 (Apr 2025) ← #13 (Nov 2023)
    Highest Position (before): #7 (Apr 2024)
                                                                        @ JetBrains
    Lowest Position: #122 (May 2015)
    Language of the Year: 2009, 2016
Cloudflare Radar ▲ API Client Language popularity: #1 (2024)
GitHub Octoverse ▲ Top 10 fastest growing languages in 2024: #3
JetBrains ▲ Language Promise Index: #4 (2024)
IEEE Spectrum ▲ Top Programming Languages: #8 (2024)
Crossover ▲ Top 10 In-Demand Programming Languages for 2025: #10
ZDnet ▲ The most popular programming languages in 2025: #10
StackOverflow 411 Most popular techs: language (professionals) (2024)
PYPL ▲: #12 (May 2025)
RedMonk ▲ Programming Language Rankings: #12 (Jun 2024)
Statista ▲ Most used programming languages among developers (2024): #12
GeeksForGeeks ▲ 20 Best Programming Languages to Learn in 2025: #13
```

^{*} я не учитывал в индексах позицию HTML/CSS

Установка



Реализации:

- 1. Официальный компилятор (Google) для ОС AIX, Android, *BSD, iOS, Linux, macOS, Plan 9, Solaris, Windows (на разных аппаратных архитектурах) и для WebAssembly (WASM).
 - 2. gofrontend + libgo для GCC и других компиляторов.
 - 3. TinyGo для embedded systems и WebAssembly.
 - 4. GopherJS кросс-компилятор из Go в JavaScript.

Поддерживаются **архитектуры**: i386, amd64, ARM, RISC-V, MIPS, ppc64, ... **go tool dist list**Лёгкая кросс-компиляция!

Установка (ОПИСАНИЕ https://golang.org/doc/install): sudo apt-get install golang

The Go **Playground** ~ интерактивное выполнение программ в браузере: https://go.dev/play/

Пример с приветом



```
package main
                                  // все программы принадлежат к своему пакету
import (
                                  // подключить...
  "fmt"
                                  // ... пакет форматированного вывода
  "0S"
                                  // ... и взаимодействия с OS
const world = "世界"
                                  // UTF-8 для исходного кода и литералов
func main() {
                                  // c main() начинается выполнение программы
  var s string = world
                                  // var переменная тип = значение
                                  // len() - встроенная функция определения размера
  if len(os.Args) > 1 {
                                  // в os.Args[0] – имя программы
                                  // имена с заглавной буквы экспортируются
    s = os.Args[1]
  fmt.Printf("Привет, %s!\n", s) // вызов функции из импортированного пакета
$ go run helloWorld.go
Привет, 世界!
$ go build helloWorld.go
$ ./helloWorld мир
Привет, мир!
$ GOOS=windows GOARCH=amd64 go build helloworld.go
$ ls helloWorld*
helloWorld helloWorld.exe helloWorld.go
```

Пример: web-server



```
package main
                                   // на основе примера, который привёл Rob Pike:
                                      полноценный многопоточный web-cepвep
import (
                                      импортировать пакеты:
   "fmt"
                                   // форматированная печать
   "loa"
                                   // протоколирование
   "net/http"
                                   // сетевая магия - в этом пакете
                      // во всех идентификаторах можно использовать символы UTF-8
func こんにちは Mup(w http. ResponseWriter, req *http. Request) {
   fmt.Fprintf(w, "Привет, мир!!!\n") // вывод ответа прямо в сеть 8-)
                                                  // точнее: в любой Writer
func main() {
   http.HandleFunc("/", こんにちは Мир)
                                                 // подключить функцию-обработчик
   // слушать порт на хосте
    log.Fatal(http.ListenAndServe("localhost:12345", nil))
   // обработать ошибку и// записать в протокол
                                                         // бесконечный цикл
   аргумент nil означает, что надо использовать HandleFunc
  https://pkg.go.dev/net/http#ListenAndServe
```

Особенности



Простой синтаксис. Минимум синтаксических конструкций. Однозначное выражение действий (по TIMTOWTDI). Каждое утверждение (statement) начинается с ключевого слова. Исходники в UTF-8.

Ошибки – это тип данных **error**. Нет исключений (exceptions). Есть panic() и recover(). Нет классов, но в struct можно описывать поля, и для всех типов данных можно определять методы. Есть интерфейсы (абстрактные наборы действий), и типы могут неявно соответствовать этим интерфейсам.

- ; служит переводом строки (line feed) автоматически вставляется компилятором, где необходимо.
- , запятая обязательна в конце строки в списке, если нет) как завершителя списка.
- _ "пустая переменная" (blank identifier) для игнорирования значения.
- := простое объявление (with inferred type) и инициализация переменной.

Все объявленные переменные получают начальное zero value (0, false, "", nil for interfaces and reference types).

Имена с заглавной буквы (Capitalized) экспортируются (видны вне пакета). Область видимости имён — пакет (package). Все имена со строчной буквы видны во всех файлах внутри одного пакета.

Функции: multiple return values, named return values, bare return, variadic functions (с переменным списком параметров). **func init() {** } // инициализирующие функции в файлах пакета. **defer** ~ отложенное исполнение функции перед завершением функции: появилось в Go.

Безтиповые константы (untyped constants) в языке со строгой типизацией! **rune** // тип данных для "символа" (code point) в кодировке UTF-8. **iota** // перечисление (enumerator) именованных целых значений.

variable declared / package imported — but not used: программа не скомпилируется!

Синтаксис: go.dev/ref/spec



```
const START, STOP = 10, 16
                                               Синтаксис похож на язык С
type PointerToInt *int
                                              тип *int – указатель на int
var i int = START
                                              &і – получить адрес і
var p1 *int = &i
var p2 PointerToInt
p2 = p1
loop:
                                              for – единственная форма цикла
for i := i; i < STOP; i++ {
                                              ";" == перевод строки
  *p1++; *p2 += 2
                                              вif, for, switch — нет скобок при условии
  if i < 0 { break loop }
  fmt.Printf("i=%d outer i=%d \n", i, *p1)
                                              в Printf – много удобных %verb и #adverb
```

Почти всё в синтаксисе стало понятно, когда прочитал статью про Go в Википедии. 8-)

i=10 outer i=13 i=11 outer i=16 i=12 outer i=19 i=13 outer i=22 i=14 outer i=25 i=15 outer i=28

Пунктуация



```
import ( "fmt" ); const ( answer = 42 ); var ( five = 42 )
() список: импортов, констант, параметров,
                                                 func f(x float64) float64 { return 0.0 }
  возвращаемых значений, ...
                                                 array [size]int; slice []string
[] размер массива, показатель среза
                                                 array[index]; slice[index]; value = map[key]
  элемент массива, среза, словаря
                                                 func f[T any](a []T) T { return a[0] }; f[int](someSlice))
[] тип параметра в generics
                                                 type struct Point { x, y int32 }
{} блок определения
{} блок начальных значений
                                                 ipAddress = [4]int{127, 0, 0, 1}
{} блок кода
                                                 func answer() int { return 42 }
                                                 gender := [2]string{ 0: "Female", 1: "Male" }
: отделяет индекс или ключ от значения
                                                 language := map[stringlint { "Go": 2007 }
                                                 label:
: ставится после метки
                                                 shadrinsk := [...] float32 { 56.05, 63.38 }
... размер массива вычисляется по значениям
                                                 func sum(numbers ...int) (sum int) { /* range numbers */ }
... список параметров переменной длины
                                                 integers := []int\{1,2,3,4,5\}; sum(integers...)
... список аргументов переменной длины
                                                 version := 1.0; released := 2012; fmt.Println(version)
; перевод строки (разделитель утверждений)
:= объявление и присваивание
                                                 site := "https://go.dev/"
                                                 place = findLocation(latitude, longitude)
, разделитель в списке
                                                 result = object.method()
. разделитель объекта и метода
                                                 channel <- value; value := <-channel
<- запись в канал и чтение из канала
```

Управление выполнением: ветвление



```
if условие {
                                           if выражение; условие {
   // код
                                               // код
} else {
                                            } else {
   // код
                                               // код
switch выражение {
                                           switch {
   case значение-1, значение-2:
                                               case условие-1:
       // код
                                                   // код
        fallthrough
                                               case условие-2:
   case значение-N:
                                                   // код
       // код
        break
   default:
       // код
```

«Всё уже придумано до нас!»

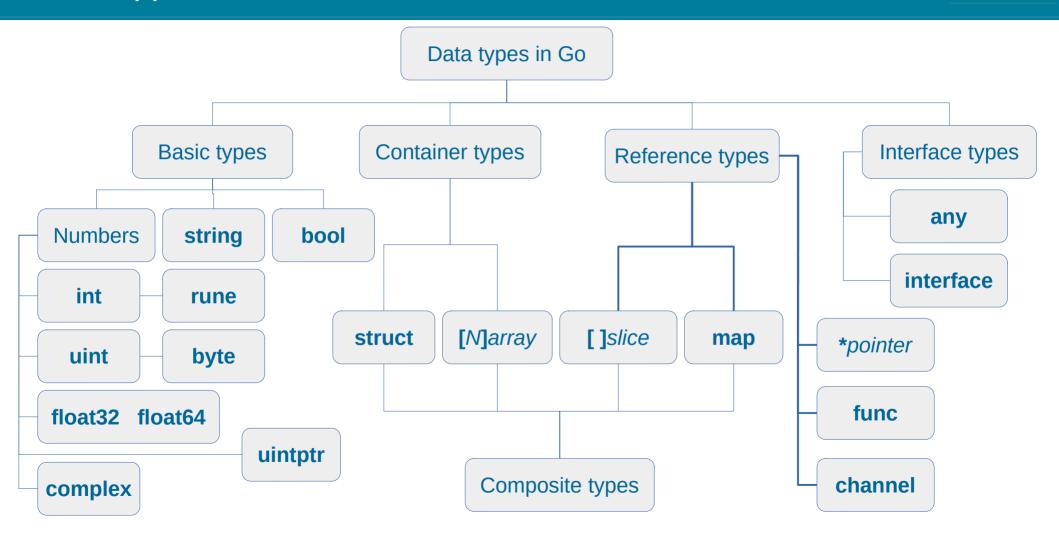
Управление выполнением: циклы



```
метка:
for инициализация; условие; изменение { // классический итерационный: (i := 0; i < n; i++)
 // break метка
 // continue
for условие {
                                              // == while
    // код
                                             // перебор целых чисел от 0 до < число
for значение := range число {
    // код
for индекс, элемент := range коллекция \{ // перебор array, slice или channel
    // код
for ключ, значение := range map {
                                             // перебор хэша
    // код
                                              // бесконечный цикл
for {
    // код
                                                                         «One for to rule them all.»
```

Типы данных





Константы: const



```
Предопределённые константы (predefined constants):
    true false nil iota
Безтиповые константы (untyped constants):
    const constant = expression
    const constant1, constant2 = value1, value2
    const ( constant1 = value1, constant2 = value2 )
   const (
        e = 2.71828182845904523536028747135266249775724709369995957496696763
        pi = 3.14159265358979323846264338327950288419716939937510582097494459
Константы с заданным типом (typed constants):
    const constant Type = expression
   const b byte = 0Xf
                                             // байт
    const x complex128 = 2+5i
                                             // комплексное число
    const Big float64 = 1 << 100
                                             // binary 1 with 100 zeros
   const i int32 = -273
                                             // целое
   const Go rune = ' 碁'
                                             // символ
    const language string = "Go"
                                             // строка
```

Операции: объявление и присваивание



```
// объявление новой переменной: всегда есть начальное значение (zero value)
// var variable Type
   var i int // 0
   var G, o rune // 0, 0
   var s string // ""
   var (
       tube chan string // nil
      ok bool // false
      f float64 // 0.0
// объявление новой переменной и присваивание значения (в функции)
              // тип выводится из присваиваемого значения
   i := 0
   t, f := true, false // параллельное присваивание
// присваивание значения уже объявленной переменной
   G, o = 'G', 'o' // параллельное присваивание (tuple assignment) i, j = j, i // обмен значений i и j
```

Имена: области видимости



```
package packgeName
                                 // имена с Заглавной буквы экспортируются вовне
const (
    packageConstant = 1
                                 // видна во всех файлах этого пакета
    GlobalConstant = 2
                                 // видна вне этого пакета: packgeName.packageConstant
                                 // видна вне этого пакета: packgeName.GLOBAL CONSTANT
    GLOBAL CONSTANT = 3
type
    packageType map[string]int
    GlobalType struct { localField int; GlobalField string } // packgeName.GlobalType
var
    packageVariable int = 4
    GlobalVariable int = 5
func packageFunction(functionParameter int) (functionResult int) {
    functionVariable := 6
                                      // видна только внутри функции
    functionResult = functionParameter+functionVariable + packageVariable+GlobalVariable
                                      // вернёт functionResult
    return
func GlobalFunction(i int, s string) GlobalType {// packgeName.GlobalFunction(2009, "Go")
    return GlobalType{localField: i, GlobalField: s}
```

Типы данных: struct



```
struct // структура: набор разнотипных полей
                                    // объявление типа
type Structure1 struct {
   field1 Type1
   field2, field3 Type2
                          // объявление переменной и
s := Structure1 {
   field1: initial_value1_of_Type1, // инициализация полей значениями
   field2: initial value2 of Type2, // по именам (не всех) полей
var s2 Structure1
                                    // объявление переменной
s2 = s1
// объявление переменной и инициализация полей значениями по порядку
s3 := Structure1{s2.field1, s2.field2, s2.field3} // следования полей
var s4 Structure1 = Structure1\{\} // инициализация пустой структурой
```

Ссылочные типы: reference types



```
pointer ~ указатель:
   p := *Type
   p = &v // value of Type
slice ~ динамический массив может изменять свой размер:
   s := []Type
мар ~ хэш | ассоциативный массив | словарь:
   m := map[KeyType]ValueType{ key: value }
function ~ функция:
   func f(parameter Type) returnValueType { /* код */ }
channel ~ канал:
   ch chan Type
```

Контейнерные типы: container types



```
[pasmep] Type // массив (array) определённой длины:
   var punchCard [80]rune // 80 * 0
   localhost := [4]int \{127, 0, 0, 1\}
   gender := [2]string { 0:"Female", 1:"Male" }
   location := [...]float32 { 56.05, 63.38 }
[] Туре // срез (slice) — динамический массив:
   primes := []int{2, 3, 5, 7, 11, 13, 17, 19, 23}
   messages := make([]string, 0, 1024)
   messages = append(messages, "OK")
map[KeyType]ValueType // map = хэш = ассоциативный массив = словарь:
    languages := map[string]int { "Go": 2007 }
   languages["Kotlin"] = 2011
   type Coordinates map[[2]float32]string
   shadrinsk := make(Coordinates)
   shadrinsk[location] = "Шадринск"
   norwalk := map[[2]float32]string {[2]float32{41.05,73.25}:"Norwalk"}
```

Функции: func



```
// функция без возвращаемого значения = процедура
func f1(param1 Type1) { /* ... */ } // no return value
    f1(arg1)
// функция с одним возвращаемым значением
func f2(p1, p2 Type1, p3 Type3) ReturnType1 { /* ... */ return res }
    result = f2(a1, a2, a3)
// функция с одним именованным возвращаемым значением
func f3(p1, p2, p3 T3) (returnValue1 RT1) { /* ... */ return }
    result = f3(a1, a2, a3)
// функция с несколькими возвращаемыми значениями: возможно, именованными
func f4(p1 T1, p2 T2) (rV1 RT1, rV2 RT2) {
    /* ... */ return res1, res2
    result1, result2 = f4(a1, a2)
// функция с переменным списком параметров
func f5(p \dots T) ReturnType { /* \ldots */ } // variadic function
    arguments := []siceOfValues{v1, v2, v3, v4, v5}
    result = f5(arguments...)
```

Функции: func main(), func init()



```
// Главная функция в пакете, с которой начинается выполнение программы.
packge main
func main() {
   * /
  в каждом пакете может быть несколько «инициализирующих» функций,
func init() {
// которые выполняются при загрузке пакета в порядке их описания
func init() {
   * /
```

Mетоды: func (t T) f()

recorder.play(album)



К любому типу данных можно присоединить поведение с помощью методов: func (object Type) method(parameters) { /* body */ } type **Temperature** float32 func (t *Temperature*) Celsius() *Temperature* { return (t-32.0)*(5.0/9.0) } var f Temperature = 37.0 c := f.Celsius() // 2.777778 type Album struct { name, artist string; year, length int } a := Album{"Pink Floyd", "Dark Side of the Moon", 1973, 44} type **TapeRecorder** struct { // ... func (r *TapeRecorder*) play(a Album) { fmt.Printf("Playing album '%s' by '%s' for %d minutes...\n", a.name, a.artist, a.length) recorder := *TapeRecorder*{/* начальные значения */}

Объектное программирование: object programming



```
// Нет классов, но можно описывать типы объектов на основе struct:
package user
type User struct {
   login string
   email string
// К такому типу можно присоединить поведение с помощью методов:
func (u User) Login() string { return u.login }
func (u User) Email() string { return u.email }
func (u *User) SetEmail(mailbox string ) { u.email = mailbox }
   // Это не конструктор, а просто обычная функция, которую назвали New
func New(l, e string) (u User) { u = User{login: l, email: e}; return u }
package main
import "github.com/mike-shock/go/sample/oop/user"
func main() {
   mike := user.New("mshock", "mshock@caiman-club.org")
   mike.SetEmail("librarian@caiman-club.org")
   fmt.Printf("'%v' '%v'\n", mike.Login(), mike.Email())
```

Интерфейсные типы: interface types



```
interface - это абстрактный тип данных, у которого может быть набор методов:
   type Messenger interface{ // описывает только действия
       Send(message string) error
       Receive() string
// объявление корректно: это переменные абстрактного типа,
  которым можно присваивать
   var icg, skype, whatapp, viber, signal Messenger
// Конкретный тип будет соответствовать ранее описанному интерфейсу,
// если реализует все методы этого интерфейса
   type Telegram struct { }
   func (t Telegram) Send(m string) error { return nil }
   func (t Telegram) Receive() (m string) { return m }
// у абстрактного типа динамически появляется конкретный тип
   var telegram Messenger = Telegram{}
```

Интерфейсные типы: interface types



```
// интерфейс также может включать в себя (embed) другие интерфейсы:
    type File interface{
        Reader
        Writer
        Seeker
        ReaderAt
        WriterAt
        Closer
    }
// any == пустой интерфейс
    type AnyObjectSatisfyMe interface{} // ему удовлетворяет любой объект
```

Интерфейсы: пример



```
type Flyer interface { fly() string } // 1-й интерфейс с методом
// все типы, которые реализуют метод fly(), будут соответствовать типу Flyer
                                               // пользовательский тип Bird
type Bird struct { Name string }
func (b Bird)fly() string {
                                               // соответствует интерфейсу Flyer
   return "flying..."
type Swimmer interface { swim() string } // 2-й интерфейс с методом
type Penguin struct { Name string } // пользовательский тип Penguin
func (f Penguin) swim() string { return "swimming..." } // соответствует сразу
func (b Penguin)fly() string { return "I can fly under water!" } // двум интерфейсам
   var s = Bird{"Sparrow"}
   var p = Penguin{"Gentoo"}
   birds := []Flyer{s, p, Bird{"Dove"}}
                                               // Flyer — это тип данных
   for _, b := range birds {
                                               // polymorphism
       fmt.Println(b, b.fly())
```

Интерфейсные типы: constraints



```
Интерфейсные типы также могут описывать ограничения (constraints):
type Numeric interface { // non-basic: has a type element (union)
   ~int | ~int8 | ~int16 | ~int32 | ~int64 | ~uint | ~uint8 | ~uint16 | ~uint32 |
   ~uint64 | ~uintptr | ~float64 | ~float32
// Интерфейс может сочетать ограничения, основные (basic) и
// пользовательские (non-basic) типы и интерфейсы, а также методы
type SpecialNumber interface {
   Number
   IsSpecial() bool
type ComaprableTypes interface {
   comparable
   SomeOtherType
```

Мно G озадачность: concurrency



• **concurrency** ~ одновременность = взаимодействие множества одновременных процессов

• parallelism ~ параллелизм = параллельное выполнение множества процессов

"Concurrency is the *composition* of independently execution things." — *Rob Pike*

"Parallelism is the simultaneous execution of multiple things." - Rob Pike

Concurrency — это способ структурировать программу, согласовывая взаимодействие процессов (возможно, выполняющихся одновременно).

Parallelism — это одновременное выполнение нескольких (возможно, взаимосвязанных) процессов.

Concurrency — это о том, как *организовать* одновременную обработку многих вещей («*dealing* with a lot of things at once»).

Parallelism — это о том, как *выполнить* обработку многих вещей одновременно («*doing* a lot of things at once»).

В программе, спроектированной на основе **concurrency**, процессы не обязательно будут автоматически выполняться параллельно (например, из-за аппаратных ограничений).

Программа, спроектированная на основе concurrency, организует взаимодействие процессов, учитывая их **параллелизм**.

Мно G озадачность: concurrency



CSP (communicating sequential processes)

Goroutines: go f()

Channels

select

Мно*Gо*задачность: goroutines



```
func f() {}
```

go f()

Каналы: chan



```
ch chan Type // канал (channel) для обмена данными между процессами
   var ch chan int
                  // объявлен канал для целых, значение nil
   club := make(chan string) // выделена память каналу для строк
   club <- "Разговор о языке Go" // send a value to the channel
   received := <- tube // get a value from the channel
func sendMessage(ch chan<- string, s string) {</pre>
   ch <- s
   go sendMessage(club, "Разговор о языке Go состоялся.")
   m1, m2 := <-club // receive from channel
   // ...
   message, ok := <- club
   if !ok {
       fmt.Println("Разговор завершился.")
```

Модульность



Модульность: функции, методы.

Пакеты.

Пакет — единица видимости имён

Инструменты: go command



```
go command [arguments...]
The commands are:
                start a bug report
   bug
   build
                compile packages and dependencies
                remove object files and cached files
   clean
                show documentation for package or symbol
   doc
                print Go environment information
   env
   fix
                update packages to use new APIs
   fmt
                gofmt (reformat) package sources
                generate Go files by processing source
   generate
   get
                add dependencies to current module and install them
   install
                compile and install packages and dependencies
   list
                list packages or modules
               module maintenance
   mod
   work
               workspace maintenance
                compile and run Go program
   run
   telemetry
               manage telemetry data and settings
   test
                test packages
                run specified go tool
   tool
   version
                print Go version
                report likely mistakes in packages
   vet
```

Инструменты: IDEs



- ▲ IDEs And Text Editor Plugins @ go.dev:
- Visual Studio Code + plug-in (Microsoft)
- **GoLand** (IDE by JetBrains)
- LiteIDE (open source and cross-platform Go IDE)
- **Komodo IDE** (cross-platform IDE with built-in Go support)
- Komodo Edit + plug-in (cross-platform text editor)
- **jEdit** (open-source, cross-platform text editor: Java)
- **Geany** (free cross-platform programmer's text editor)
- Notepad++ (text & source code editor: Windows)
- Kate (cross-platform text editor with Go support out-of-the-box: KDE)
- Sublime Text (commercial text editor: macOS, Windows, Linux)
- TextMate (commercial text editor: macOS)
- vim & Neovim+ vim-go plugin (open-source, cross-platform text editor)

... Atom, BBEdit, Chime, CodeLobster IDE, Coding Rooms, emacs, Gitpod, IDEone, Jdoodle, OneComplier, OnlineGDB, Micro, Nova, zed, Zeus IDE, ...

Применение



Области применения ▲ Go:

- DevOps & SRE ▲ (Development Operations & Site Reliability Engineering)
- Cloud & Distributed Network Services
- Web Development ▲ (frameworks, toolkits, engines, servers)
- System Automation & CLIs ▲, Utilities & Stand-Alone Tools
- ... multi-platform GUI apps (fyne.io)
- ... Al clients via API & libraries (TensorFlow in GO, etc.)

Software на Go:

Allegro (eCommerce), AmneziaWG, AKS [Azure Container Service] @ Microsoft, AresDB @ Uber, Buffalo (web framework), Caddy (web server), CockroachDB, Digger (IaC), Docker, Drone (CD), DropBox (backend), ent @ Meta, Flamingo (web framework), Gin (web framework), Gitlab Runner, Google Cloud, Gorilla (web toolkit), Grafana, Hugo (website engine), InfluxDB, JuiceFS, Kubernetes, LXD @ Canonical, Mattermost (messaging platform), Monzo (banking app), Ollama (89%), PayPal, Prometheus (monitoring & alerting toolkit), Rend (large scale data caching @ Netflix), SoundCloud, Soundscape (music streaming server), Terraform (IaC), Timesheets (project management), Twitch (livestreaming), VITESS @ YouTube, Zabbix agent2, ...

Top 60+ Open-source Apps Written with Golang in 2024 ▲

Go: критика



При разработке и развитии программных комплексов, написанных на Go, выделяют следующие недостатки:

- Синтаксис слишком простой, мало syntactic sugar.
- Синтаксис непривычный: использование [] в типах параметров и в интерфейсах при ограничениях снижает читабельность. Странный шаблон при форматировании даты и времени.
- Нет настоящего ООП.
- Ограниченный вывод типов (inference): явное указание типов параметров: reduces the simplicity and reduces boilerplate code benefits.
- Constraint Restrictions: Constraints are limited to interfaces, and the comparable constraint may not cover all use cases, limiting the flexibility of generics in certain scenarios.
- Implicit error handling: frustrating for developers who are used to exception-based error handling: Verbosity, Lack of Control Flow Interruptions
- Нет перегрузки функций (function overloading).
- Не хватает значений по умолчанию для параметров (default values for arguments).
- nil and weak type safety
- no support operator overloading or keyword extensibility, no support immutability declarations.

 Goroutines scheduler decides when and how goroutines are scheduled, which can lead to non-deterministic behavior.
- Сборщик мусора иногда вносит недопустимые задержки при выполнении программ;
- В некоторых случаях требуется более низкоуровневое управление распределением памяти, как в Rust.

Многие компании сочетают применение Go с использованием других новых языков, таких как Rust (Rust vs. Go: Why They're Better Together ▲).

Мои впечатления от Go



Синтаксис простой, но с некоторыми непривычными конструкциями. Логично спроектирован, предсказуем. Исходники хорошо понимаются. Непривычно после Ruby: все объявления и преобразования надо делать явно.

Очень строгий: переменная не используется – код компилироваться не будет! Strong typing и другие строгости важны для надёжности больших приложений. Явная работа с ошибками дисциплинирует программиста. Убедился в преимуществах отказа от ООП в пользу объектного подхода в Go. Интерфейсы в Go — основа динамичности и гибкости при разработке.

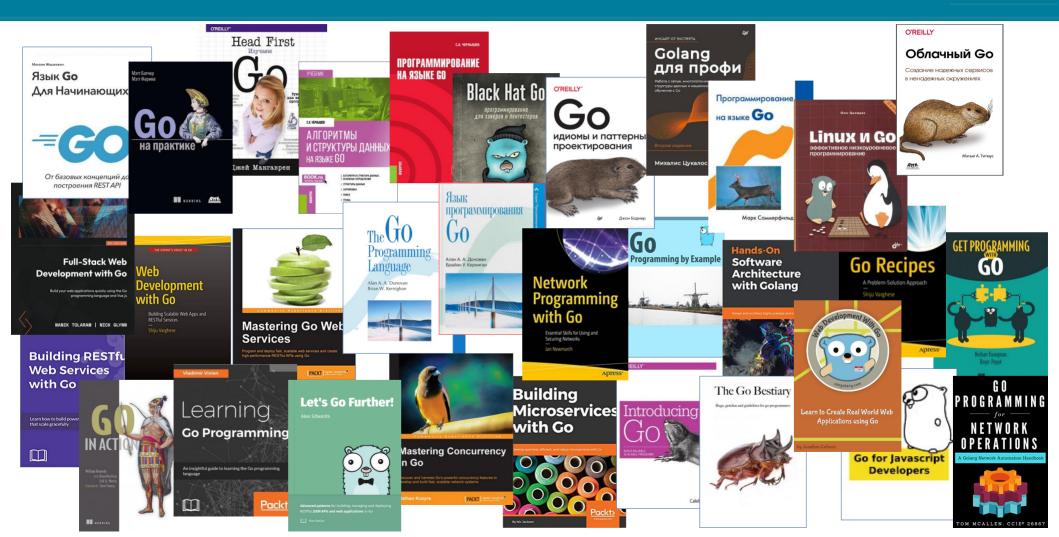
Довольно низкоуровневый: напоминает Си — вспомнил молодость! Очень быстро компилируется. Удобно сразу выполнить: go run program.go Легко скомпилировать исполняемую программу для другого «железа» и ОС. Действительно очень быстро выполняется.

Много *стандартных* библиотек — на все случаи жизни. Легко подключать сторонние модули. Хорошая документация на библиотеки (с исполняемыми примерами). Много сайтов с примерами — изучать легко.

Наверное, это последний язык, разработанный «классиками», которые создали Unix.

Книг много (лучше читать на английском)





Ссылки 🛦



```
qo.dev/
                                                 // Официальный сайт языка
go.dev/play/
                                                 // Go Playground ~ выполнение в браузере
 go.dev/ref/spec
                                                 // Спецификация языка (!!!)
 qo.dev/doc/
                                                 // Документация
  qo.dev/doc/code
                                                 // How to Write Go Code
 pkg.go.dev/std
                                                 // стандартная библиотека
  gobyexample.com
                                                 // Go в примерах
• go.dev/doc/modules/layout
                                                 // Структура каталогов

    github.com/golang-standards/project-layout

                                                 // Стандартный макет [большого] Go проекта
tour.golang.org
                                                 // Экскурсия по возможностям Go
• golangdocs.com
                                                 // примеры конструкций
 appliedgo.net/why-go/
                                                 // 15 Reasons I Love Go
 awesome-go
                                                 // libraries for everything...

    go.dev/doc/effective_go

                                                 // "Effecive Go" бесплатная web-книга
• gopl.io
                           // "The Go Programming Language" by A.A.A.Donovan & B.W.Kernighan

    w3schools.com/qo/

                                   @ w3schools // Справочник
 Самоучитель по Go для начинающих @ proglib.io // Самоучитель
 Дорожная карта Go-разработчика
                                   @ proglib.io // План изучения
  lyceum.yandex.ru/go
                                                 // Яндекс-лицей: Программирование на Go
• Книги по Go
                                    @codelibs.ru //
                                                 // TinyGo

    tinygo.org
```

Готов ответить на вопросы





???



Словарик

task ~ задача



communicating sequential processes ~ взаимодействие последовательных процессов concurrency ~ свойство программы, допускающее одновременное выполнение нескольких вычислительных процессов CSP = communicating sequential processes gopher ~ программист на Go goroutine ~ подпрограмма, возможно, выполняемая параллельно multitasking ~ многозадачность parallelism ~ параллелизм = параллельное выполнение вычислений process ~ процесс subprocess ~ подпроцесс subtask ~ подзадача