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. Улучшение *технологии 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) для generics.

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

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

Применение: где, как и почему лучше использовать Go.

Критика: недостатки Go и альтернативы ему.



Шустрый суслик **Gopher**

(автор: 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, удобный для сопровождения.

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; OC Plan 9, Inferno, UTF-8, sam, acme, языков Sawzall, Limbo, Newsqueak.





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

«Our original goal was not to create a new programming language, it was to create a better way to write programs.»

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):

- язык спроектирован для надёжного программирования больших программных комплексов;
- стабильная спецификация языка: совместимость с предыдущими и последующими версиями;
- 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;
- легко компилировать для разных ОС и архитектур одни и те же исходники;
- интерфейсы с неявным соответствием: позволят 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 распределённая система управления внешними пакетами с идентификацией по URL;
- мощный набор стандартных инструментов: go command, go tool command;
- широкие возможности для 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).
2008 -03	1-й проект (draft) спецификации языка.
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 improvement of compiled code.
2015 -08-19	go1.5: compiler & runtime written in Go (+ a little assembler); dynamic libraries.
2017-08-24	go1.9: type aliases.
2020-02-25	go1.14: Go modules.
2022- 03-15	go1.18: generics. Built-in fuzz testing.
2023-08-08	go1.21: min, max, clear built-in functions.
2024-02-06	go1.22: math/rand/v2 package; PGO (Profile-guided Optimization) in compiler.
2024 -08-13	go1. 23 : range over function types; <i>iter, unique, struct</i> s packages.
2025-02-11	go1. 24 : generic type aliases; weak pointers; post-quantum cryptography; FIPS mode.
2025 -08-12	go1. 25 : experimental GC (10-40% faster), many changes in the standard library.

Спецификация языка и стандартной библиотеки обратно совместимы с версиями Go 1.х.
Поэтому многие крупные компании выждав время убедились в долговременной поддержке язы

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

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 практически все **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

Обновление:

go get go@1.24.3 # или go get go@latest

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 ОС
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
                                      импортировать пакеты:
import (
    "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, записать в протокол
                                        // и завершить бесконечный цикл
                                        // nil означает: использовать HandleFunc
                                        // pkg.go.dev/net/http#ListenAndServe
```

Синтаксис: 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, реализуя их.

- ; 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 с выводом типа из значения (inferred type) и инициализация переменной в функции. Все объявленные переменные получают начальное zero value (0, false, "", nil для интерфейсов и ссылочных типов).

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. 1 пакет = 1 каталог.

Функции — полноценные типы: multiple return values, named return values, bare return, variadic functions, anonymous functions.

func init() { } // инициализирующие функции в файлах пакета.

defer f() // отложенное исполнение функции перед завершением функции: появилось в Go.

go f() // запустить функцию как goroutine.

Kaнaлы: channel <- value; value <- channel; select / case / default // переключение каналов

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.

import "package"; var declared // если не используются — программа не скомпилируется!

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
... список параметров переменной длины
                                                 func sum(numbers ...int) (sum int) { /* range numbers */ }
... список аргументов переменной длины
                                                 integers := []int\{1,2,3,4,5\}; sum(integers...)
; разделитель выражений в for и if
                                                 for i := 0; i < n; i++ \{ if y := f(i); y > 0 \{ println(y) \} \}
; перевод строки (разделитель утверждений)
                                                 version := 1.0; released := 2012; fmt.Println(version)
:= объявление и присваивание (в функции)
                                                 site := "https://go.dev/"
                                                 place = findLocation(latitude, longitude)
, разделитель в списке
. разделитель объекта и метода
                                                 result = object.method()
                                                 channel <- value; value = <-channel</pre>
<- запись в канал и чтение из канала
```

Управление выполнением: ветвление



```
if условие {
                                    if выражение; условие {
                                        действие1()
   действие1()
} else {
                                    } else {
   действие2()
                                        действие2()
switch выражение {
                                    switch {
   case значение1, значение2:
                                        case условие1:
       действие1()
                                            действие1()
        fallthrough
                                        case условиеN:
                                            действие1()
   case значениеN:
       действие N()
        break
   default:
                                    switch выражение; значение := интерфейс(type) {
       действиеПоУмолчанию()
                                        case Tun1:
                                            действие1()
                                        case ТипN:
                                            действие1()
```

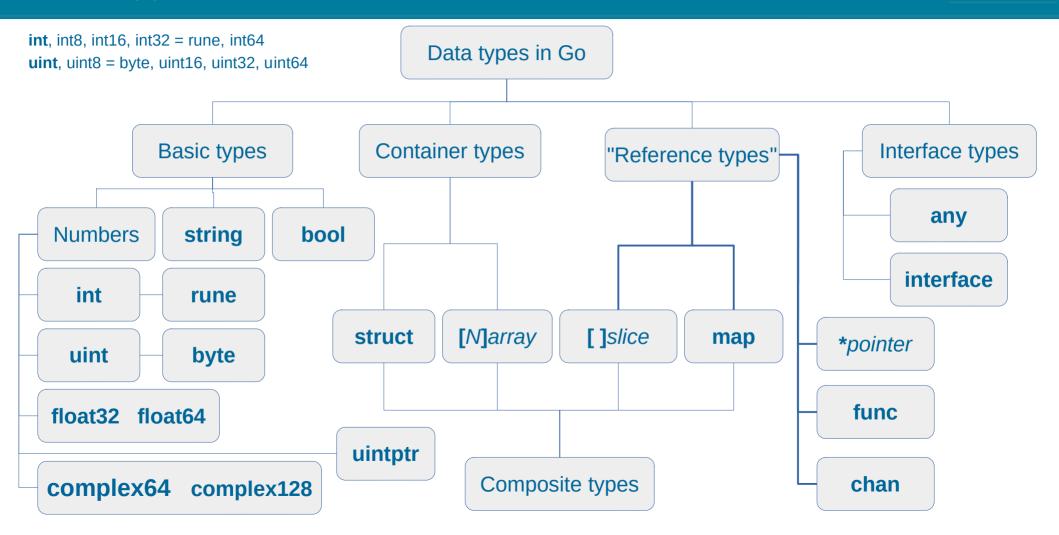
Управление выполнением: циклы



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

Типы данных





Константы: const



```
Предопределённые константы (predefined constants):
    true false nil iota
Безтиповые константы (untyped constants): // mathematically precise, no need for qualifiers (-42LL, 7UL, etc.)
    const constant = expression
    const constant1, constant2 = value1, value2
    const ( constant1 = value1; constant2 = value2 )
    const (
         e = 2.71828182845904523536028747135266249775724709369995957496696763
         \pi = 3.14159265358979323846264338327950288419716939937510582097494459
         \pi^2 = \pi * \pi
    var pi2 float32 = \pi2 // Only when used, constants are truncated to size
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
                                           // с плавающей точкой: 1 со 100 нулями
    const i int32 = -273
                                               // целое
    const Go rune = ' 碁'
                                             // символ
    const language string = "Go"
                                               // строка
    type Weekday int
    const ( Sun Weekday = iota; Mon; Tue; Wed; Thu; Fri; Sat ) // 0,1,2,3,4,5,6
```

Объявление и присваивание: var



```
var переменная Type
При объявлении новой переменной всегда есть начальное значение (zero value).
   var i int // 0
   var G, o rune // 0, 0
   var s string // ""
   var (
      tube chan string // nil
      ok bool // false
      x float64 // 0.0
   var answer = 42 // тип int выведен из присвоенного значения
// объявление новой переменной и присваивание значения (в функции)
                       // тип выводится из присваиваемого значения
   j := 25
   t, f := true, false // параллельное присваивание
// присваивание значения уже объявленным переменным
   s = "Go"
                       //
   G, O = 'G', 'O' // параллельное присваивание (tuple assignment)
   i, j = j, i // обмен значений i и j
```

Типы данных: struct



Структура (struct) — набор разнотипных полей

```
// объявление типа Structure1
   type Structure1 struct {
       field1 Type1
                                     // с полями
       field2, field3 Type2
                                       // разных типов
                              // объявление переменной и
   s := Structure1 {
       field1: initial_value1_of_Type1, // инициализация полей значениями
       field3: initial value3 of Type2, // по именам (не всех) полей
                                       // объявление переменной
   var s2 Structure1
   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



Когда переменные-указатели передаются в функции, указанные значения могут быть изменены.

```
pointer ~ указатель:
   var v BaseType // переменная типа BaseType
   var p *BaseType // указатель на переменную типа BaseType
            // ссылка на значение переменной типа BaseType
   p = &v
                     // значение переменной типа BaseType по ссылке на v
slice ~ динамический массив: может изменять свой размер:
   s := []Type
мар ~ хэш | ассоциативный массив | словарь | отображение | «мапа»:
   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
   places := make(Coordinates)
   places[location] = "Шадринск"
   places[[2]float32{33.54, -118.05}] = "Norwalk"
```

Функции: func main(); func init()



```
// Главная функция в пакете main, с которой начинается выполнение программы.
package main
func main() {
   обработка (данных)
   в каждом пакете может быть несколько «инициализирующих» функций,
func init() {
   инициализация ( &данных )
// которые выполняются при загрузке пакета в порядке их описания
func init() {
       инициализация ( &данных )
   и могут располагаться в разных файлах этого пакета
```

Функции: func



```
// функция без возвращаемого значения = процедура
func f1(param1 Type1) { /* ... */ }
    f1(arg1)
// функция с одним возвращаемым значением типа ReturnType1
func f2(p1, p2 Type1, p3 Type2) ReturnType1 { var r ReturnType1; /* ... */ return r }
    result := f2(a1, a2, a3)
// функция с одним именованным возвращаемым значением
func f3(p1, p2 Type3) (r ReturnType1) { /* ... */ return }
    result := f3(a1, a2)
// функция с несколькими возвращаемыми значениями: возможно, именованными
func f4(p1 Type1, p2 Type2) (r1 ReturnType1, r2 ReturnType2) {
    /* ... */ return r1, r2
    result1, result2 := f4(a1, a2)
// функция с переменным списком параметров = variadic function
func f5(p ...Type1) ReturnType1 { r ReturnType1; /* ... */ return r }
    arguments := []siceOfType1Values{v1, v2, v3, v4, v5}
    result1 = f5(arguments...)
    result2 = f5(a1, a2, a3)
```

Типы-функции: func type



```
// функция как тип
// type FuncType func(ParamerType1, PT2) ReturnValueType
    type F1 func(int, int) int
// у любой функции есть тип, например: func(int, int) int
    func add(x, y int) int { return x+y }
// функция как значение переменной
    var f0 F1 = add
    fn := func() { fmt.Println("anonymous") } // анонимная функция типа func()
// функция как возвращаемое значение
    func returnsFunc() F1 { return add }
    f1 := returnsFunc()
    y := f1(4, 2)
// функция как параметр
    func receivesFunc(a, b int, f F1) (r ReturnType) { r:= f(a, b); return r }
    sum = receivesFunc(21, 21, add)
    production = receivesFunc(21, 21, func(x, y int) int { return x*y } )
// определение и вызов анонимной функции
    func() { fmt.Println("lambda") }() // lambda типа func()
```

Meтоды: func (t T) f()



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; media string } a := Album{"Pink Floyd", "Dark Side of the Moon", 1973, 44, "катушка 18 см"} type **TapeRecorder** struct { Model string 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*{model: "Hota 203-1 ctepeo"} recorder.play(a)

Объектное программирование: type struct + func



```
«Object- but not type-oriented» -- D.Griesmer
// 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") // static dispatch of methods
   fmt.Printf("'%v' '%v'\n", mike.Login(), mike.Email())
```

Интерфейсные типы: interface types



```
interface - это абстрактный тип данных для описания поведения: описывает
набор функций (set of method signatures):
    type Messenger interface { // Basic interface
        Send(message string) error
        Receive() string
    var icq, skype, whatsapp, viber, signal, discord Messenger
// Объявлены переменные абстрактного типа, которым можно присваивать значения типов,
// соответствующих контракту (интерфейсу);
// но пока у них нет конкретного типа (реализации), а значение = nil
// Конкретный тип будет неявно соответствовать ранее описанному интерфейсу,
// если реализует все методы этого интерфейса
    type Telegram struct { }
    func (t Telegram) Send(m string) (e error) { e = telegram.send(m); return err }
    func (t Telegram) Receive() (m string) { m = telegram.receive(); return m }
// у абстрактного типа динамически появляется конкретный тип (underlying type) и значение
    var telegram Messenger = Telegram{}
    telegram.Send("Интерфейсы в Go") // вызывается реализованный метод
```

Интерфейсные типы: interface types



```
// Интерфейс также может включать в себя (embed) другие интерфейсы:
    type File interface{
       Reader
                               // Embedded interfaces
       Writer
       Seeker
       ReaderAt
       WriterAt
       Closer
   type Reader interface {
        Read(b []byte) (int, error)
    type Writer interface {
       Write([]byte) (int, error)
// Конкретный тип может соответсвовать (implement) нескольким интерфейсам
   type Telegram struct { }
   func (t Telegram) Read(b []byte) (int, error) { return 0, nil }
// any == пустой интерфейс
    type AnyTypeSatisfyMe interface{} // ему соответствует объект любого типа
```

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 set (union)
   ~int | ~int8 | ~int16 | ~int32 | ~int64 | ~uint | ~uint8 | ~uint16 | ~uint32 |
   ~uint64 | ~uintptr | ~float64 | ~float32 // General interface
// 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
```

Обобщённые типы: generics



```
// Generics описываются с помощью ограничений (constraints) на обобщённый тип в функции:
func First[T any](a []T) (result T, err error) {
    if len(a) == 0 {
         return result, errors.New("Slice is empty!") // пустой результат и ошибка
    return a[0], err // 1-й элемент и nil (ошибки нет)
func Last[T any](a []T) (result T, err error) {
    if len(a) == 0 {
         return result, errors. New("Slice is empty!")
    return a[len(a)-1], err // последнийй элемент и nil
    sliceOfIntegers := []int{1, 2, 3, 4, 5}
    // если тип параметра можно вывести из переменной, то его можно не указывать
    first, err := First[int](sliceOfIntegers)
    last, err := Last(sliceOfIntegers)
    sliceOfStrings := []string{"Вышел", "зайчик", "погулять"}
    fmt.Println(First(sliceOfStrings))
    fmt.Println(Last[string](sliceOfStrings))
```

Пакеты: package



Пакет — это набор (логически связанных) исходных файлов, расположенных в одном каталоге. В начале каждого файла должно описываться его принадлежность к пакету фразой

package packageName // site/path/packageName

Пакет — единица видимости имён (типов, констант, переменных, полей, функций):

- Все имена видны во всех файлах одного пакета.
- Имена в пакете, начинающиеся с Заглавной буквы экспортируются: они видны в программе, которая импортировала пакет фразой

import "*packageName*" // это строка

var result packageName.Type = packageName.Func(packageName.Const, packageName.Var)

Пакет **main** — это специальное имя пакета, которое означает, что этот пакет содержит код, который будет скомпилирован в двоичный исполняемый файл. В одном из файлов (обычно, в main.go) этого пакета должна быть функция **main()**.

Пакеты (не из стандартной библиотеки) могут располагаться где угодно, их полные адреса (локальные пути или URL) содержатся в файле **qo.mod**

Модули



Модуль — это набор пакетов, которые распространяются (с определённым номером версии) как единое целое. Модули могут загружаться прямо из систем управления версиями исходников или с общедоступных серверов.

Модуль идентифицируется путём до модуля (module path), который объявляется в файле **go.mod** вместе с информацией о зависимостях модуля.

создать файл go.mod c именем модуля go mod init path/to/module/moduleName

Например:

go mod init caiman-club.org/go/mshock/presentation

Главный каталог модуля (module root directory) — это каталог, содержащий файл **go.mod**. Когда модуль состоит из нескольких пакетов, они располагаются в подкаталогах главного каталога модуля.

Мно G озадачность: concurrency



• **concurrency** ~ одновременность = взаимодействие множества процессов, которые могут выполняются одновременно, если позволяет «железо» и ОС

• 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 параллельное выполнение нескольких (независимых, 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 параллельно («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 возможный параллелизм.

If you have only one processor, your program can still be concurrent but it cannot be parallel.

Мно*G*озадачность: средства



Многозадачность в Go реализована на основе CSP (communicating sequential processes, C. A. R. Hoare, 1978).

Для управления многозадачностью в язык встроено несколько механизмов:

```
• Сопрограммы (goroutines) для распаралелливания выполнения:

go f() // запустить любую функцию как процесс
```

```
• Kaнaлы (channel) для обмена данными и синхронизации выполнения: channel <- value // отправить значение в канал и ждать value = <-channel // ждать и получить значение из канала
```

• Выбор (select) для обработки нескольких потоков данных череез каналы:

```
select {// ожидать, когда что-то появится в канале #1case v2 := <-ch2:</td>// ждать и получить значение из канала #2case ch3 <- v3:</td>// отправить в канал #3default:// обработать другое событие
```

МноGозадачность: goroutines



Coпрограммы (goroutines) — это легковесные потоки, которые выполняюся одновременно и управляются главным потоком (main go thread).

```
func f(n int) { println(n) }
func main() {
   for n := range 5 {
      qo f(n+1) // запустить 5 экземпляров f() параллельно c main()
      // можно запустить анонимные функции
      qo func () { println(n+1) }()
   time.Sleep(1 * time.Second)
   println("Вышел зайчик погулять.")
```

Мно*G*озадачность: каналы



```
// объявить переменную для канал обмена данными указанного типа
   var channel chan T
// создать канал
   channel = make(chan T, pasmepEy\phiepa)
// отправить значение в канал
   channel <- value
// и ждать, пока не будет прочитано значение из канала
// ждать, пока не будет записано значение в канал
// прочитать значение из канала в переменную
   value := <-channel
// прочитать из канала, игнорируя значение
   <-channel
close(channel) // закрыть канал
```

Каналы: пример



```
var club chan string // объявить канал для строк, значение nil
   club = make(chan string) // выделить память каналу для строк
// club <- "Разговор о Go" // будет deadlock !!!
// отправить в канал значение параллельно
   go func() { club <- "Предложен разговор о языке Go" }()
   received := <-club // получить значение из канала в переменную
   go sendMessage(club, "Разговор о языке Go запланирован.")
   go sendMessage(club, "Разговор о языке Go состоялся.")
   m2, m1 := <-club, <-club // получить 2 сообщения
   close(club)
   message, ok := <-club // проверить доступность канала
   if !ok {
                               // канал закрыт
       fmt.Println("Разговор завершился.")
                             func sendMessage(ch chan<- string, s string) {</pre>
                                 ch <-s
```

Инструменты: go command



```
go command [arguments...] // All integrated into the one go command.
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, ...

Распространение



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

Alibaba, Amazon, American Express, **Apple**, Armut ($C\# \to Go$), **Baidu**, BBC, bitly, ByteDance, Canonical, Capital One, CERN, **Cloudflare**, Cockroach Labs, Curve, DataDog, Dailymotion, **Docker**, DropBox (*Python* \to *Go*), GitHub, **Google**, gov.uk, Heroku, **IBM**, InfluxDB, Intel, K8s, Kubernetes, **Meta**, **Microsoft**, Monzo Bank, **Mozilla** (*Rust* & *Go*), Netflix (*Java* \to *Go*), New York Times, **Oracle**, PayPal, Pinterest, **Reddit**, RedHat, Riot Games, Slack, Salesforce (*Python*, $C \to Go$), SendGrid, Stream (*Python* \to *Go*), SoundCloud, Terraform, The Economist, The New York Times, **Twitch**, Uber, Walmart, YouTube, **X** *I* **Twitter**, *Mhozux* ∂*pyzux opzahusayux u проектах* **open-source**.

В России (всеми крупными компаниями):

Яндекс, ЦУМ, УГМК-Телеком, Точка, Т-Банк, Совкомбанк Технологии, СберТех, Ростелеком, Онлайн-кинотеатр Иви, МТС+МWS, 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...

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.)
- ... IoT & embedded systems (TinyGo)

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, Genkit, Flamingo (web framework), Gin (web framework), Gitlab Runner, Google Cloud, Gorilla (web toolkit), Grafana, Hugo (website engine), InfluxDB, JuiceFS, Kubernetes, LangChainGo, LocalAI, LXD @ Canonical, Mattermost (messaging platform), Monzo (banking app), Ollama (89%), PayPaI, Prometheus (monitoring & alerting toolkit), Rend (large scale data caching @ Netflix), SoundCloud, Soundscape (music streaming server), Terraform (IaC), Timesheets (project management), Twitch (live-streaming), VITESS @ YouTube, Zabbix agent2, ...

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

Популярность: рейтинги



```
TIOBE index ▲ (since 2009):
                                                                       Is Golang Still Growing?
Go Language Popularity
Trends in 2024
    Now: #8 (Aug 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

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.
- Нет настоящего ООП.
- Ограниченный вывод типов (inference): явное указание типов параметров: reduces the simplicity and reduces boilerplate code benefits.
- Ограничения (constraints) задаются только интерфейсами и могут ограничивать гибкость generics в определённых сценариях работы.
- Явная обработка ошибок: смущает разработчиков, кто привык к исключениям (многословность, нет прерывания потока выполнения).
- Нет перегрузки функций (function overloading).
- no support operator overloading or keyword extensibility, no support immutability declarations.
- Не хватает значений по умолчанию для параметров функций (default values for arguments).
- Использование nil и weak type safety?
- Диспетчер сопрограмм (goroutines scheduler) управляет выполнением goroutines, что может привести к недетерминированному поведению.
- Сборщик мусора иногда вносит недопустимые задержки при выполнении программ.
- Странный шаблон при форматировании даты и времени.
- Нет проверки значений на соответствие перечислению, объявленному через iota.
- В некоторых случаях требуется более низкоуровневое управление распределением памяти, как в Rust.

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

Мои впечатления от Go



Синтаксис простой, но с некоторыми непривычными конструкциями. Логично спроектирован, предсказуем. Исходники хорошо понимаются. Непривычно после динамического 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 программ. Явная работа с ошибками дисциплинирует программиста: о них надо думать постоянно. Убедился в преимуществах отказа от традиционного ООП в пользу объектного подхода в Go. Интерфейсы в Go — основа динамичности и гибкости при разработке.

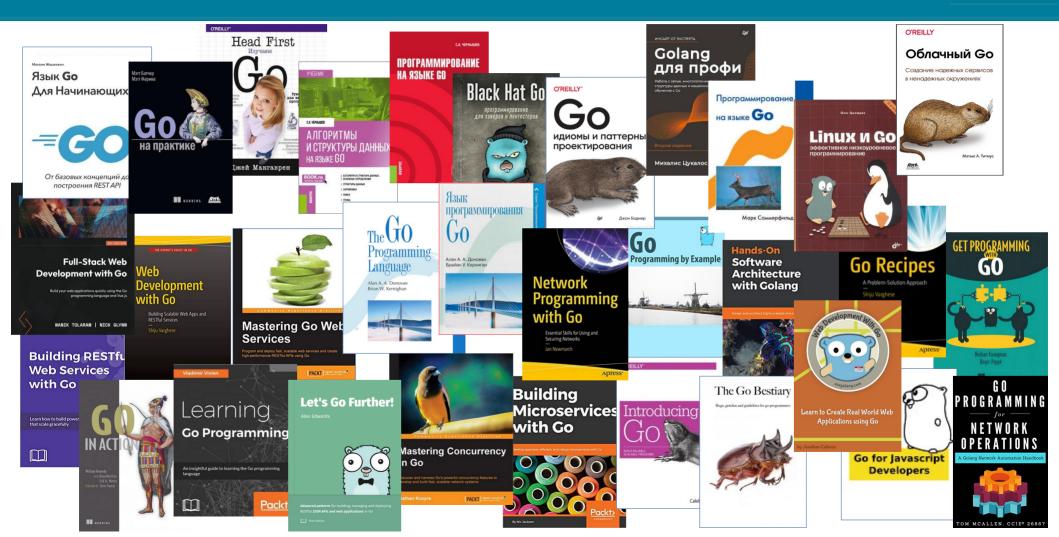
Довольно низкоуровневый: напоминает Си — вспомнил молодость! Очень быстро компилируется. Удобно сразу выполнить: go run program.go Легко скомпилировать исполняемую программу для другого «железа» и ОС. Действительно очень быстро выполняется: переписывал на Go с Python и Ruby.

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



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 task ~ задача