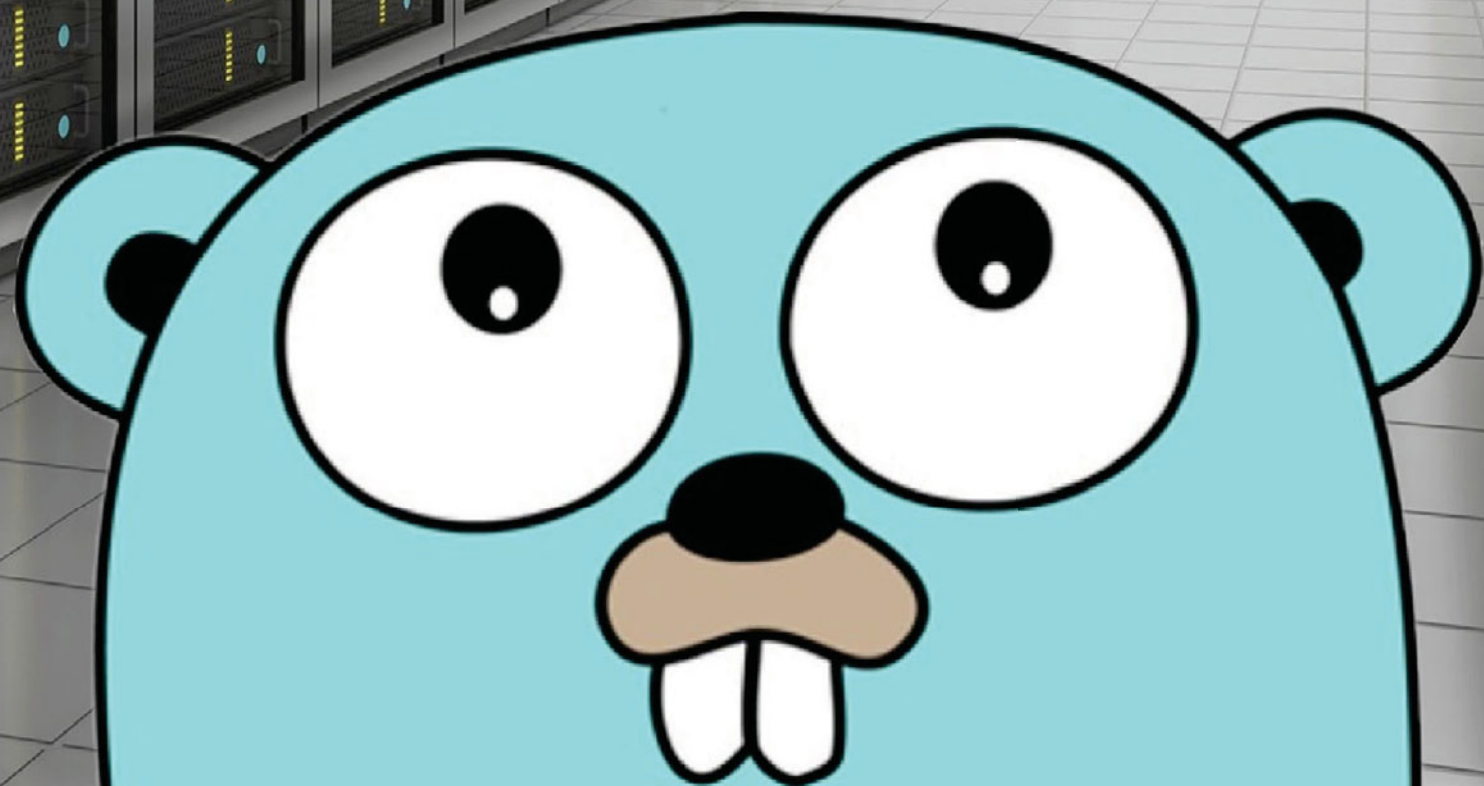
Крюков Е.А

ЯЗЫК ПРОГРАММИРОВАНИЯ Go

Руководство

2016



Содержание

Установка (обновление)

Пакеты

Данные

Управляющие конструкции

Интерфейсы

Обработка ошибок

Параллельное программирование

Пакет sync

Ввод/Вывод

Пакет io

Пакет io/ioutil

GoDoc

Cgo - C + Go

Время

Пакет time

Структуры данных

Пакет bytes

Пакет containter/heap

Пакет container/list

Пакет container/ring

Форматы данных

Пакет encoding/csv

Пакет encoding/json

Пакет encoding/xml

Пакет encoding/gob

Пакет encoding/ascii85

Пакет encoding

Стек веб-технологий

Пакет net

Пакет net/http

Пакет cgi

Пакет fcgi
Пакет sql
Пакет redis
Пакет rpc
Пакет WebSocket
Создание пользовательского интерфейса
Пакет text/template
Пакет flag
Пакет html/template
Пакет gtk
Взаимодействие с ОС
Пакет os
Пакет os/user
Пакет os/exec
Пакет os/signal
Архивирование
Пакет archive/tar
Пакет archive/zip
Пакет bufio
Пакет fmt
Хеширование
Пакет hash
Пакет crypto/md5
Криптография
Пакет crypto
Пакет crypto/tls
Пакет crypto/rand
Регулярные выражения
Пакет regexp
Тестирование и отладка
Пакет log
Пакет testing
Пакет testing/quik

Пакет testing/iotest

Установка (обновление)

Поддерживаются все распространенные ОС и процессоры. Если вашей комбинации нет в списке —

устанавливаем через исходники.

FreeBSD 8 и ст.	amd 64 (64 бит)	Кроме Debian GNU/kFreeBSD
Linux 2.6.23 и старше	amd64, 386 (32 бит), arm	Для arm установка из исходников. CentOS/RHEL 5.x не поддерживается
Mac OS X 10.7 и ст.	amd64	Используйте clang или gcc*, которые идут с Xcode**
Windows XP и ст.	amd64, 386	Исп. MinGW gcc*

*gcc нужен если вы планируете использовать cgo(совместимость с языком C)

** Вам нужно только установить инструменты командной строки для Xcode. Если у вас Xcode старше версии 4.3, вы можете установить их через Downloads – Components

Если вы обновляетесь - удалите весь прежний каталог. Из переменных окружения удалите путь к папке bin.

В Linux и FreeBSD редактируете /etc/profile или $HOME/.profile.

В Mac OS X удаляете файл /etc/paths.d/go.

Заходим на оф. сайт golang.org/dl/ Язык развивается очень быстро, книга для версии 1.6.1, так что качаем её.

Windows

Лучше скачать msi. Если устанавливаете из zip-архива, придется устанавливать переменные окружения(см ниже).

- MSI: после установки в переменную среды PATH должен добавиться путь к каталогу bin в папке где вы установили Go.
- ZIP: .В переменную GOROOT записываем имя каталога в который извлекли Go, например c:/go. Так же в переменную PATH необходимо приписать путь к папке bin, которая находится в том же каталоге, например, без кавычек: "c:\go\bin;"

Добраться до переменных среды Windows можно так: Мой компьютер,

Пр. кн. мыши, Свойства, Изменить параметры (в правом нижнем углу), Дополнительно.

Mac OS X installer

Скачиваем файл, запускаем. Установка будет произведена в usr/local/go.

В переменную среды PATH должен добавиться путь usr/local/go/bin. Если этого не произошло, перезапустите рабочую сессию терминала.

Из исходников (Linux, Mac OS X, FreeBSD и пр.):

Скачиваем архив tar.gz и извлекаем в usr/local, создаем древо каталогов Go в /usr/local/go. Например:

```
tar -C /usr/local -xzf go$VERSION.$OS-$ARCH.tar.gz
```

Выбирайте файл архива подходящий для вашей версии. Например, если вы устанавливаете Go версии 1.2.1 для Linux на процессоре x86-64, архив должен называться go.1.2.1.linux-amd64.tar.gz

(Обычно эти команды должны выполняться root-пользователем или через sudo)

Добавляем /usr/local/go/bin в переменную среды PATH. Добавьте эту строчку:

```
export PATH=$PATH:/usr/local/go/bin
```

в ваш etc/profile (при установке для всех пользователей) или в $HOME/.profile.

Установка в произвольный каталог:

Путь должен быть без пробелов.

Если вы установили Go не в каталог по умолчанию, добавьте следующие команды к $HOME/.profile:

```
export GOROOT=$HOME/go
export PATH=$PATH:$GOROOT/bin
```

Примечание: переменная GOROOT устанавливается только если Go установлен не в стандартный каталог.

Проверка

Создадим рабочий каталог (workspace) - папка где будут храниться наши проекты на Go, например в Windows: C:\Users\VasyaPupkin\Desktop\work или в Linux: $HOME/work и допишем его в переменную среды GOPATH:

В Unix

$ export GOPATH=$HOME/work

Вы должны добавить эту команду в скрипт который выполняется при запуске оболочки (start-up script), например, `$HOME/.profile`.

В Windows По той же инструкции.

Размещение кода в workspace

Go удобен при работе с репозиториями.

Код размещается в рабочей папке. Обязательно, в ней должны быть 3 папки:

- src - содержит исходники на Go разбиты по пакетам (один пакет на директорию)
- pkg - сюда автоматически кладутся скомпилированные пакеты
- bin - а сюда программы

Так же обязательно чтобы имя каталога и имена исходников не содержали пробелов.

src обычно хранит репозитории контроля версий (Git, Mercurial и т.п), что позволяет параллельно вести разработку нескольких проектов. Пример:

bin/

hello # исполняемая программа

outyet # исполняемая программа

pkg/

linux_amd64/

github.com/golang/example/

stringutil.a # объект пакета

src/

github.com/golang/example/

.git/ # метаданные репозитория Git

hello/

hello.go # исходный код

outyet/

main.go # исходный код

main_test.go # исходный код с тестами

stringutil/

reverse.go # исходник пакета

reverse_test.go # исходник с тестами

Hello World: создаем каталог src/github.com/user/hello, где вместо user - ваше имя пользователя на github.com, если вы там зарегистрированы, если нет — оставьте так.

Внутри создаем файл hello.go:

```
package main

import "fmt"


func main() {
 fmt.Printf("hello, world\n")
}
```

Компилируем с помощью go tool:

Unix:

```
$ go install github.com/user/hello
```

Windows: в командной строке (Меню Пуск — Выполнить — cmd):

Переходим в папку с исходником. Например:

```
chdir C:\Users\VasyaPupkin\desktop\work\src\github.com\user\hello\
```

И компилируем: go install

Исполняемый файл появится в директории bin

На время разработки используйте go build - исполняемый файл появится в директории исходника.

Если после запуска файла, консоль лишь мелькнула на экране и закрылась - зациклите программу добавив в конец:

```
again:
goto again
```

При этом программу можно не компилировать. Находясь в каталоге исходника:

```
go run hello.go
```

После установки

Заходим на golang.org — там есть песочница для выполнения кода и примеры программ.

Пакеты

В Go нет препроцессора и нет особого разделения на библиотеки и модули программы. Их заменяет концепция пакетов.

Вы пишете один файл формата .go:

Если это пакет (библиотека) — начинаете его с объявления package name, где name – название вашего пакета.

Если это программа — начинаете её с объявления package main.

После директивы package всегда следуют директивы import. И только потом объявляются глобальные переменные, функции и т.д.

```
package main
import (
 "fmt"
 "math"
)
import "os"
import "html/template"
import "database" ; "syscall"
```

Пакеты линкуются поочередно — в указанном вами порядке. Импортирование друг в друга запрещено.

Пакеты-библиотеки (все что не package main) могут содержать только объявления. Но если вы хотите исполнить в них какой-либо код, размещайте его в особом блоке init():

```
var Happy bool
func init() {
 Happy = true // Инициализация только в разделе init()
}
```

Пакеты-программы исполняют код в функции main(), но тоже могут иметь блоки init() которые выполнятся перед функцией main(), даже если они располагаются после.

Размещение исполняемого кода вне этих блоков приведет к ошибке.

Импортировать можно только пакеты находящиеся в директориях %GOPATH%/bin и

%GOROOT%/bin. Их желательно размещать в первой, т. к. с нее начинается поиск.

Абсолютный путь указать нельзя.

Например мы создали пакет UserPackage.go и разместили его в папке

C:\Users\VasyaPupkin\Desktop\work\src\github.com\user\UserPackage

Допустим, переменные: %GOROOT% = C:\Go и %GOPATH% = C:\Users\VasyaPupkin\Desktop\work

В используемой программе, (слеши не зависят от операционной системы) мы пишем, к примеру:

import "github.com/user/UserPackage"

Содержимое пакетов может быть видимым только внутри пакета или доступным другим пакетам — экспортируемым. Чтобы сделать что-либо экспортируемым, присваивайте имя которое начинается с большой буквы:

math.Pi //Обращение к константе из пакета math

math.pi //Ошибка

import m “math” // Теперь нельзя обратиться к math.Pi, только m.Pi

import . “math” // Теперь обращаемся просто к Pi

Идеология Go не приемлет импорта неиспользуемых пакетов (и вообще вычислений результат которых не используется). Такая программа просто не скомпилируется. Чтобы это обойти:

import _ “math” // При этом исполняется раздел init() пакета math

Сейчас мы рассмотрели статическую линковку: грубо говоря, когда код из библиотеки копируется в программу и программа становится независимой от библиотеки.

Начиная с версии Go 1.5, появилась возможность использования динамических библиотек языка C и кода на нем, с помощью утилиты Cgo, о которой будет рассказано.

Данные

Встроенные типы

uint беззнаковое целое, 32 или 64 бита в зависимости от ОС

int размер как у uint

uintptr uint для хранения указателя

uint8 0..255	0..18446744073709551615	-2147483648..2147483647
uint16 0..65535	int8 -128..127	int64
uint32 0..4294967295	int16 -32768..32767	
uint64	int32	
-9223372036854775808..9223372036854775807		

float32 32-битное число с плавающей точкой

float64 64-битное

complex64 комплексное число, действительное и мнимое числа формата float32

complex128 float64

Для работы с большими числами или в области финансов (дабы избежать потери точности при операциях), пользуйтесь пакетом math/big.

byte - псевдоним для uint8.

Символьные типы

rune

string

Коллекции: массивы, слайсы, каналы.

Указатели и структуры.

Объявления

var someNumber int

var thisIsFalse bool = true

var (foo float32

bar

)

По умолчанию, после объявления без инициализации переменные все-равно инициализируются нулем, false и пустой строкой "" в соответствии с типом. Так же есть значение nil — not initialised (аналог NULL), которые получают, к примеру, пустые указатели.

Оператор := автоматически задает тип переменной. Свой тип с этим оператором нельзя назначить.

myGab := 5 // int

byMag := 65.5 // float32

Числовые типы

Литералы целых чисел задаются как и в других языках:

42 0600(восьмеричный) 0xBadA55 0XdeadBEEF(шестнадцатеричный) Двоичного формата нет.

Числа с плавающей точкой задаются только в десятичном формате:

0.72.40 072.40 (72.40) 2.71828 1.e+0 6.67428e-11 1E6 .25 .12345E+5

Комплексные числа выражаются десятичными цифрами. i всегда маленькая:

0i 011i 11i 0.i 2.71828i 1.e+0i 6.67428e-11i1E6i

.25i .12345E+5i

Для доступа к их частям есть две встроенные функции:

imag() // Возвращает float32 или float64

real() // То же самое

Создание комплексного числа:

complex(65, 42) Вернет complex64 или 128 для 2-х float32 или 64

complex(_, 7) //Применение пустого идентификатора

Буквенные типы

Тип rune - псевдоним для int32, аналог типа char. Хранит символ в кодировке Unicode.

a := '\n' // Апостроф ([Э] на клавиатуре) используется только с типом rune

// (и rune только с апострофом)

b := "\"" // С типом string используются двойные и обратные

d := `"` // кавычки ([Ё] на клавиатуре) если строка

// неинтепретируемая

В неинтерпретируемых строках не надо экранировать символы (использовать саму обратную кавычку нельзя) и можно переносить строку.

Запомните что в начало интерпретируемых пустых строк автоматически добавляется символ возврата каретки.

s := "abc" // s[0] = "\r"

Это все одна и та же строка:

"日本語"

`日本語`

"\u65e5\U672c\u8a9e" // Unicode

"\U000065e5\U0000672c\U00008a9e" // Unicode

"\xe6\x97\xa5\xe6\x9c\xac\xe8\xaa\x9e" // UTF-8

'a' 'ä' '本' '\t' '\000' //После \ ровно три восьмеричных цифры

'\007'

'\377'

'\x07' //После \x ровно две шестнадцатеричных цифры

'\xff'

'\u12e4'

'\U00101234' //После \U ровно восемь шестнадцатеричных цифр

'\'' // rune содержащий символ апострофа

'aa' // нельзя: много символов

'\xa' // нельзя: мало цифр

'\0' // нельзя: мало цифр

'\uDFFF' // нельзя: лишь половина так называемой суррогатной

// пары — двусоставного символа

'\U00110000' // нельзя: неверный символ Unicode

Список escape-последовательностей:

\a U+0007 предупреждение(alert), он же колокольчик(bell)

\b U+0008 backspace (забой)

\f U+000C перевод страницы

\n U+000A новая строка или перевод строки

\r U+000D возврат каретки

\t U+0009 горизонтальный отступ

\v U+000b вертикальный отступ

\\ U+005c обратный слэш

\' U+0027 одинарная кавычка

\" U+0022 двойная кавычка

Обработка текста с символами Unicode довольно трудная задача, т.к одному смысловому символу соответствует множество кодов, к примеру: тире(3 вида), дефис, минус; римские цифры и буквы I V X; отступы и пробелы. Лучше всего использовать готовые библиотеки.

При присваивании переменных типа string переменным rune, делается привидение типа и индексация:

var a rune = rune(“DABO”[1])

Так же учтите что из-за синтаксиса Go интерпретируемые строки не могут содержать переноса строки, поэтому их надо конкатенировать(объединять):

var a = “Two ” +

“rows”

var b = `Two

rows`

Прочие типы

Тип bool может иметь только значения true/false (вместо них нельзя использовать другие значения, например 0 как false)

Тип byte используется во множественных типах — каналах и строках.

Константы

можно объявлять в любом месте программы — при этом у них будет соответствующая зона видимости.

Константы вычисляются на этапе компиляции.

Константы могут быть любого типа. Вы должны сами присвоить нужный вам тип, если вы этого не делаете, по ситуации константы получают тип bool, rune, int, float64, complex128 или string.

const g int = 9.8

const a = 9 // оператор := не работает с константами

const af = float32(g)

const b = 2 + 2 /* 4 Выражения просчитываются на этапе

компиляции */

При объявлении констант доступна особая переменная iota:

type ByteSize float64 //Объявление нового типа.

const (

zero ByteSize = iota // 0

first // 1

_ // 2, пустой идентификатор

third // 3

agbe = iota*2 // 8

)

const faba = iota // 0

Константы не могут иметь значения бесконечность, отрицательный ноль и не-число (NaN).

Операторы

Приоритет Операторы

5 * / % << >> & &^

4 + - | ^

3 == != < <= > >=

2 &&

1 ||

% - остаток от деления & - побитовое И | - побитовое ИЛИ

+ - сложение чисел; объединение строк; унарный + делает число положительным.

- разность; смена знака числа на противоположный

&^ - сброс бита (И НЕ) >> - сдвиг битов вправо << - сдвиг битов влево

&& - И || - ИЛИ

^ - XOR (исключающее ИЛИ)

Побитовые операторы работают только с целыми числами.

Так же доступны следующие операторы: += -= /= *= %= <<= >>= &= |= ^= &^= . Общий принцип:

var a int; a += 5 // a = a + 5

Так же есть операторы ++ (+1) и - - (-1). Они ставятся только после операнда (например a++). Их нельзя использовать в выражениях. Только как отдельно стоящую директиву или в управляющих конструкциях (например в циклах).

Синтаксис, форматирование, имена

Исходный код программ на Go кодируется в UTF-8 (символ Unicode представленный последовательностью от одного до шести байт), поэтому вы можете использовать в именах, литералах и комментариях любые национальные алфавиты. В этом случае тоже действует правило — если первая буква имени (типа, функции и т.п) большая — значит это видимо в других пакетах.

В Go вы можете не ставить точку с запятой после директивы: перенос строки считается за неё.

var a string = “Lorem ipsum” + // поэтому строки приходится “dolor sir amet” // конкатенировать

Точка с запятой ставится автоматически если в конце строчки кода находится:

идентификатор(имя), литерал(значение), ключевое слово break, continue, fallthrough или return,

) или] или } или операторы ++ и -- .

Точку с запятой надо ставить если вы хотите разместить несколько директив на одной строчке:

fmt.Println("a"); fmt.Println("b")

Имена состоящие из нескольких слов принято давать в верблюжьейНотации.

Пакетам, по соглашению даются имена в нижнем регистре без пробелов и _ (нижнее подчеркивание).

Первым символом в имени обязательно должна быть буква или _ .

var something int // Это разные переменные

var someThing int

Следующие ключевые слова зарезервированы:

break default func interface select

case defer go map struct

chan else goto package switch

const fallthrough if range type

continue for import return var

Так же в блоках (все что находится между фигурными скобками - { … }) предопределены следующие имена:

Типы: bool byte complex64 complex128 error float32 float64

int int8 int16 int32 int64 rune string

uint uint8 uint16 uint32 uint64 uintptr

Константы: true false iota

Нулевое значение: nil

Функции: append cap close complex copy delete imag len

make new panic print println real recover

Коллекции: массивы, слайсы, каналы.

В Go принят общий стандарт форматирования кода. Об этом позаботится утилита gofmt – она приведет ваш код к общепринятому виду. О ней рассказано в одном из разделов.

Чтобы код скомпилировался, все циклы и условные конструкции указываются с фигурными скобками ({}) - даже если внутри них ничего нет, при этом

открывающая скобка должна располагаться на той же строчке, иначе программа не скомпилируется.

Для выделения памяти служат два оператора — new() и make().

new() - возвращает указатель на обнуленную память размером в соответствии с заданным типом.

p := new(SyncedBuffer) // тип *SyncedBuffer

make() - функция с общим видом make(Тип, Аргументы). Она работает только со множественными типами (коллекциями) — слайсами, ассоциативными слайсами(map) и каналами. Например:

v := make([]int, 100) //Создание слайса (динамического массива)

Приведение типов

Go – язык со строгой типизацией. Единожды присвоив тип переменной, его нельзя сменить. Так же в Go нет неявного привидения типов. При операциях с переменными различных типов (даже при сложении int и int32) придется приводить тип вручную:

type(variable) // Функция возвращающая переменную нового типа,

//например string(amount) – вернет переменную amount типа string

var i = string(0x71) //буква q в Unicode

fmt.Println(i) // q

fmt.Println(int(i)) // 113

Классы/структуры и типы

Новым типом можно задать: встроенные типы, структуры, массивы, слайсы, ассоциативные массивы, каналы, указатели, функции и интерфейсы:

type T1 string

type T2 T1

type T3 []T1

type T4 T3

Каждый тип имеет основополагающий тип(underlying type) — для T1 и T2 это string. Для []T1, T3, T4 это []T1. Привидение можно делать сразу к underlying type, минуя промежуточные типы.

В Go нет ключевого слова class, конструкторов и деструкторов. Классы объявляются через структуры:

type Vertex struct {

X, Y int //Неявно инициализированы нулем

```
}
var (
 v1 = Vertex{1, 2} // Тип Vertex
 v2 = Vertex{X: 1} // Y:0 неявно
 v3 = Vertex{} // X:0 и Y:0
 p = &Vertex{1, 2} // тип *Vertex (указатель)
)
```

X и Y являются полями(fields) класса/структуры. Классы могут иметь другие классы в качестве полей:

```
type Point3D struct { x, y, z float64 }
type Line struct { p, q Point3D }
origin := Point3D{} // Point3D с нулевыми значениями
line := Line{origin, Point3D{y: -4, z: 12.3}}
// line.q.x с нулевыми значениями
```

Методы объявляются вне классов**.** О них подробно в соответствующем разделе.

Еще существуют так называемые анонимные поля - указывается просто тип без имени:

```
type Another struct{
 *Vertex
}
```

В примере выше класс Another наследует поля и методы типа Vertex.

```
var a Another
a.Vertex.X //Можно обращаться так
a.Y //Или так
```

Поля анонимных классов-полей (например X и Y в примере выше) называются вдвинутыми (promoted), они ничем не отличаются от обычных полей, за тем исключением что их нельзя использовать в составных литералах (как в примере выше с типом Line).

Указывать наследуемый тип в структуре можно по разному:

```
... struct {
 T1 // обращаемся как T1
 *T2 // обращаемся как T2
```

P.T3 // обращаемся как T3

***P.T4 // обращаемся как T4**

}

При получении методов от анонимных полей действуют следующие правила:

Если S содержит анонимное поле класса T: *S получает методы классов T и *T

S получает методы только T

Если S содержит анонимное поле класса *T: *S получает методы классов *T и T

S получает методы классов *T и T

Все имена в структуре должны быть уникальны. Конфликт имен анонимных полей:

… struct {

T

***T**

***P.T**

}

Для работы с пакетом reflect (рефлексия - когда программа меняет сама себя):

после полей структур вы можете оставлять теги. Они представляют из себя строковый литерал (на него распространяются все правила для типов string и rune):

struct {

x, y float64 "" // Пустой тег приравнивается к его // отсутствию

name string "любая строка может являться тегом"

}

Как применяются теги вы узнаете из раздела посвященного пакету reflect.

Заметьте что на типы тоже распространяется правило — "если имя с большой буквы — видимо в других пакетах":

type Amount []int

В Go нет перегрузки операторов и функций(т.е нельзя использовать разные функции с одним именем или к примеру задать сложение разных типов через оператор +).

Так же тип может быть функцией (подробней в специальном разделе).

Указатели и ссылки

Любая переменная имеет значение и свой адрес в памяти. Указатель(pointer) имеет свой адрес и хранит адрес другой переменной, функции и т.п. & - операция взятия адреса.

```
var a int
fmt.Println(a) // 0
fmt.Println(&a) //Например 0x10434114
var p *int //Указатель на значение типа int
fmt.Println(p) // nil , еще ни на что не указывает

p = &a
//Теперь p хранит адрес переменной a
fmt.Println(p) // 0x10434114 (адрес a)
fmt.Println(&p) // 0x1dc39b15 (адрес самого p)

// * - операция обращения по адресу (оператор разыменования),
// т.е взять значение по хранимому адресу
fmt.Println(*p) // 0 (значение a)

a = 5
fmt.Println(*p) // 5
*p = 10
fmt.Println(a) // 10

 var a **int // Указатель на указатель
 var b *int
 var c int
 a = &b
 b = &c
 fmt.Println(**a) // 0
```

Адреса что вы видите на экране — виртуальные. Скорей всего они будут различаться каждый раз когда вы запускаете программу. На самом деле размещением ваших данных в реальной памяти занимается операционная система, а те адреса с которыми вы работаете — удобная абстракция.

Не инициализированные указатели принимают значение nil.

Функции и методы

При передаче аргумента в функцию создается его копия. Если мы хотим изменить значение какого либо аргумента — мы передаем его по ссылке (в функцию копируется адрес изменяемой переменной). Функции объявляются снаружи main():

```
func increment (number *int) { // Если без * - number не изменит
 *number = *number + 1 // своего значения
}
// …
var v int; increment(&v)
```

Функцию можно сначала вызывать, а затем давать объявление.

В Go функции могут возвращать несколько значений:

```
//Функция возвращает инкрементированное значение и ошибку
func increment(number int8) (int8, int){
 var err bool
 if number==127 { //Переполнение типа (скобки в if не обязательны)
 err=true
 return number, err
 }else{
 return number+1, err
 }
}
myNum, err := increment(255)
myNum, _ = increment(254)
_, err := increment(255)
```

Пустой идентификатор _ (нижнее подчеркивание) играет роль черной дыры — он ничего не сохраняет. Количество аргументов слева должно равняться количеству возвращаемых значений.

first, second = second, first // Не надо писать функцию swap()

Так же в Go можно задать имена возвращаемых значений. Так называемый голый возврат (naked return):

func increment(number int8) (result int8, err int8){

//… Можно просто (result, err int8)

return

}

Метод класса:

func (someVert* Vertex) increment() { // Ключевого слова this

someVert.X++ // нет, обзывайте как хотите

someVert.Y++

}

var myVert Vertex

myVert.increment() j

Ресивер(receiver) - объект над которым вызывается метод.

Метод объявляется с ресивером-указателем если он изменяет значения ресивера.

Запомните что для стандартных типов методы создавать нельзя.

Чтобы не запутаться с оператором косвенного доступа (* - звездочка) -

как мы увидели выше метод вызывается без оператора взятия адреса (& - амперсанд), хоть он и объявлен с указателем. Go скрыто преобразует вызовы таких функций из class.func() в (&class).func().

Таким же образом Go преобразует вызовы методов над указателями:

var v Vertex

fmt.Println(v.do()) // OK

p := &v

fmt.Println(p.do()) // OK

Функцию можно передавать в качестве аргумента другой функции. Последняя называется коллбеком(callback, функция обратного вызова). Т.е функция может являться типом.

type doThat func // Функция без аргументов не возвращающая ничего

Анонимная функция — функция без имени. Их еще называют лямбда-функции:

```
package main
import "fmt"

func main() {
// Сохраняем анонимную функцию в переменную
MultiPrint := func(whatPrint string, times int) {
 for ; times != 0; times--{
 fmt.Println(whatPrint)
 }
}
// Теперь MultiPrint можно вызывать как обычную функцию:
MultiPrint("Lorem ipsum”, 3)
}
```

Пример передачи анонимной функции в качестве аргумента:

```
package main
import "fmt"

// Коллбек
func wrapFunc(f func(string) int, subst string){
 f(subst)
}

func main() {
 anon := func(who string) int { // Сохраняем анонимную функцию
 fmt.Println("Hello", who)
 return 0
 }
 wrapFunc(anon, "gad") //Передача аргумента-функции без скобок!
 anon("VOVAN OTVE4AET") //Вызов.
}
```

Вызов анонимной функции можно делать и при объявлении, поставив в конце скобки с аргументами:

```
func ...{
// ...
}("PO NOGY")
```

Можно объявить сигнатуру функции в качестве типа и затем подставлять его в аргументы:

```
type writeIt func (string) int
//Так бы выглядело объявление обертки для функции из примера выше:
func wrapFunc(f writeIt, subst string){
//...
}
```

Замыкание(closure) – анонимная функция которая обращается к переменными вне её контекста (грубо говоря снаружи). В данном примере два контекста — функции adder и возвращаемой функции:

```
//Функция возвращающая замыкание
func adder() func(int) int {
 sum := 0
 return func(x int) int {//Возврат замыкания
 sum += x //sum – переменная из внешнего контекста
 return sum //Если sum объявить заново в этой функции
 } //это будет другая переменная с тем же названием,
}
```

Если вы хотите чтобы функция принимала нефиксированное число аргументов (включая случай без аргументов), поставьте оператор ... (многоточие) перед типом последнего аргумента. В функции эти аргументы представляются в виде слайса (динамического массива):

```
func sum (num ...int) int{
if num == nil{ // Если не передали(слайс нулевой длины)
 fmt.Println("No arguments")
 return 0
}
var sum int
 for i:= range num{ // range - прохождение по
```