OTUS

Golang Developer. Professional

otus.ru

REC **Проверить, идет ли запись**

Меня хорошо видно && слышно?

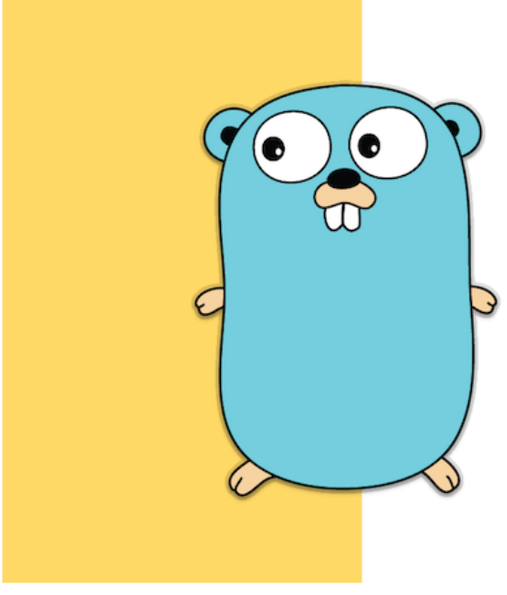
Ставим “+”, если все хорошо
“-”, если есть проблемы

Тема вебинара

Знакомство с курсом

Олег Венгер

Руководитель группы Защиты профилей в Wildberries



Правила вебинара

Активно
участвуем

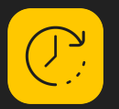
Off-topic обсуждаем
в учебной группе

Задаем вопрос
в чат или голосом

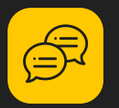
Вопросы вижу в чате,
могу ответить не сразу

Условные обозначения

Индивидуально



Время, необходимое
на активность



Пишем в чат

Говорим голосом

Документ

Ответьте себе или
задайте вопрос

О чем будем говорить:

- Почему появился Go?
- Начало работы с Go
- Как сдавать домашние задания

Немного истории

1956-1958 LISP
1959 Cobol
1964 Basic
1970 Pascal
1970 C
1978 SQL
1983 C++
1991 Python
1995 Java
1995 PHP
2009 Go

Golang

Go (часто также Golang) — компилируемый многопоточный язык программирования, разработанный внутри компании Google.

Разработка Go началась в сентябре 2007 года, его непосредственным проектированием занимались **Роберт Гризмер**, **Роб Пайк** и **Кен Томпсон**, занимавшиеся до этого проектом разработки операционной системы Inferno.

Официально язык был представлен в ноябре 2009 года.

https://github.com/golang/go

Проблемы Google, подтолкнувшие к Go

- Медленная сборка (вплоть до часа)
- Неконтролируемые зависимости
- Каждый программист использует свое подмножество языка
- Трудность в чтении чужого кода
- Сложности деплоя (инструменты автоматизации, межъязыковые сборки и пр.)

https://talks.golang.org/2012/splash.article

Требования к Go

- Возможность работы на больших масштабах: крупные команды разработчиков, большое количество зависимостей
- Должен быть знакомым программистам Google, а значит - Си-подобным
- Должен быть современным:
  - использование возможностей многоядерных машин "из коробки"
  - встроенные библиотеки для работы с сетью и пр.

https://talks.golang.org/2012/splash.article

Характеристики Go

-- Императивный
-- Компилируемый
-- Статически типизируемый
-- Нет классов, но есть структуры с методами
-- Есть интерфейсы
-- Нет наследования, но есть встраивание
-- Функции - объекты первого класса
-- Есть замыкания
-- Функции могут возвращать больше 1 значения
-- Есть указатели, но нет адресной арифметики
-- Обширные возможности для конкурентности
-- Сборка в 1 бинарный файл
-- Набор стандартных инструментов

Сообщество не дремлет:

-- https://github.com/avelino/awesome-go
-- https://go.dev/

Playground:

-- https://play.golang.org
-- https://goplay.space

Постулаты Go (Заповеди Роб Пайка)

https://go-proverbs.github.io/
https://habr.com/ru/post/272383/

Где используется Go

- Веб (backend)
- Системные утилиты
- Devops
- Сетевое программирование

Что написано на Go

- Grafana
- Docker
- Consul
- Kubernetes
- Prometheus

Кто использует Go?
https://github.com/golang/go/wiki/GoUsers

Первые шаги

https://go.dev/tour

Список литературы

Установка Go

Getting Started
https://go.dev/doc/install

GOROOT

`GOROOT` — переменная, которая указывает где лежит ваш дистрибутив Go, т.е. компилятор, утилиты и стандартная библиотека. В новых версия Go (> 1.0) утилиты сами определяют расположение Go.

Однако, вы можете узнать `GOROOT`

```
$ go env | grep ROOT
GOROOT="/usr/local/go"
```

И можете посмотреть исходный код Go =)

```
vim /usr/local/go/src/runtime/slice.go
```

Программа на Go

```go
package main // Имя текущего пакета

// Импорты других пакетов
import "fmt"

// Функция main как точка входа
func main() {
    fmt.Println("Hello!")
}
```

```
$ go build -o prog prog.go

$ file prog
prog: Mach-O 64-bit executable x86_64

$ ./prog
Hello!
```

```
$ go run prog.go
Hello!
```

Кросс-компиляция

Go позволяет легко собирать программы для других архитектур и операционных систем.

Для этого при сборке нужно переопределить переменные `GOARCH` и `GOOS`:

```
$ GOOS=windows go build -o /tmp/prog prog.go

$ file /tmp/prog
prog: PE32+ executable (console) x86-64 (stripped to external PDB), for MS Windows

$ GOARCH=386 GOOS=darwin go build -o /tmp/prog prog.go

$ file /tmp/prog
prog: Mach-O i386 executable
```

Возможные значения `GOOS` и `GOARCH`

- `go tool dist list`
- https://gist.github.com/asukakenji/f15ba7e588ac42795f421b48b8aede63

Go Modules

Начиная с Go 1.11 появилась поддержка модулей — системы версионирования и зависимостей, а также разработки вне `GOPATH`.

Стандартные команды (`go get`, `go install`, `go test` и т.д.) работают по-разному внутри модуля и внутри `GOPATH`.

Модуль — любая директория вне `GOPATH`, содержащая файл `go.mod`

Начиная с версии go 1.13 `GOPATH` не упоминается в Руководстве по началу разработки

Создание Go модуля

- (Опционально) создайте и склонируйте (в любое место) репозиторий с проектом

```
git clone https://github.com/user/otus-go.git /home/user/otus-go
```

- Создайте внутри репозитория нужные вам директории

```
mkdir /home/user/otus-go/hw-1
```

- Зайдите в директорию и инициализируйте Go модуль

```
cd /home/user/otus-go/hw-1
go mod init github.com/user/otus-go/hw-1
```

Теперь `/home/user/otus-go/hw-1` — это Go модуль.

https://blog.golang.org/using-go-modules

Добавление зависимостей

Внутри модуля, вы можете добавить зависимость от пакета с помощью

```
$ go get github.com/beevik/ntp
go: finding golang.org/x/net latest
```

```
$ cat go.mod
module github.com/mialinx/foobar

go 1.22

require (
    github.com/beevik/ntp v0.2.0 // indirect
    golang.org/x/net v0.0.0-20190827160401-ba9fcec4b297 // indirect
)
```

Внимание: в момент добавления зависимостей их версии фиксируются в `go.sum`.

Авто-добавление

Также можно просто редактировать код

```
package main

import (
    "fmt"

    "github.com/go-loremipsum/loremipsum"
)

func main() {
    fmt.Println(loremipsum.New().Word())
}
```

А потом запустить

```
$ go mod tidy
```

Это добавит новые и удалит неиспользуемые зависимости.

Базовые команды

`go get` — скачивает пакеты и его исходники в `$GOPATH/pkg/mod/` .

`go install` собирает и устанавливает указанные пакеты в `$GOBIN` (по умолчанию `$GOPATH/bin`).

One-shot запуск

Запустить файл "как скрипт".

```
go run ./path/to/your/snippet.go
```

Удобно для проверки кода и синтаксиса.

Так же можно использовать Go PlayGround: https://go.dev/play/p/Fz3j-hbcocv

Получение справки

```
$ go help
Go is a tool for managing Go source code.

Usage:

    go <command> [arguments]

The commands are:
    bug         start a bug report
    build       compile packages and dependencies
    clean       remove object files and cached files
    doc         show documentation for package or symbol
...

$ go help build
usage: go build [-o output] [-i] [build flags] [packages]

Build compiles the packages named by the import paths,
along with their dependencies, but it does not install the results.
...
```

Форматирование кода

В Go нет style guide, зато есть `go fmt path/to/code.go`

Было:

```go
package main
import "fmt"



const msg = "%d students in chat\n"
type Student struct{
    Name string
    Age int
}
func main() {
    for i:=99;i>0;i-- {
        fmt.Printf(msg, i)
        if i<10{
            break
        }

    }
}
```

Стало:

```go
package main

import "fmt"

const msg = "%d students in chat\n"

type Student struct {
    Name string
    Age  int
}

func main() {
    for i := 99; i > 0; i-- {
        fmt.Printf(msg, i)
        if i < 10 {
            break
        }

    }
}
```

Обновление и сортировка импортов

```
$ go get golang.org/x/tools/cmd/goimports
$ goimports -local my/module/name -w path/to/code.go
```

```
import (
    "strings"
)

func main() {
    fmt.Println(loremipsum.New().Word())
}
```

```
import (
    "fmt"

    "github.com/go-loremipsum/loremipsum"
)

func main() {
    fmt.Println(loremipsum.New().Word())
}
```

Линтеры

Линтер — программа, анализирующая код и сообщающая о потенциальных проблемах.

`go vet` — встроенный линтер

```
$ go vet ./run.go
# command-line-arguments
./run.go:14:3: Printf call needs 1 arg but has 2 args

$ echo $?
2
```

`golint` — популярный сторонний линтер

```
$ go get -u golang.org/x/lint/golint

$ ~/go/bin/golint -set_exit_status ./run.go
run.go:7:6: exported type Student should have comment or be unexported
Found 1 lint suggestions; failing.

$ echo $?
1
```

Металинтеры

Металинтеры — обертка, запускающая несколько линтеров за один проход.

https://github.com/golangci/golangci-lint/

```
$ ~/go/bin/golangci-lint run ./run.go
run.go:14:3: printf: Printf call needs 1 arg but has 2 args (govet)
        fmt.Printf(msg, i, i)
        ^
run.go:7:6: `Student` is unused (deadcode)
type Student struct {
     ^

$ echo $?
1
```

Работа с golangci-lint

https://gist.github.com/kulti/25f9243939e699428f7e14c5a3c8c32c
https://www.youtube.com/watch?v=QnG8z-JWfEY - Демо урок

Как сдавать домашние задания?

Инструкция

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Заполните, пожалуйста, опрос о занятии по ссылке в чате

Следующий вебинар

5 февраля

Синтаксис языка

Ссылка на вебинар будет в ЛК за 15 минут

Материалы к занятию в ЛК — можно изучать

Обязательный материал обозначен красной лентой

Спасибо за внимание!

Приходите на следующие вебинары

Олег Венгер

Руководитель группы Защиты профилей в Wildberries