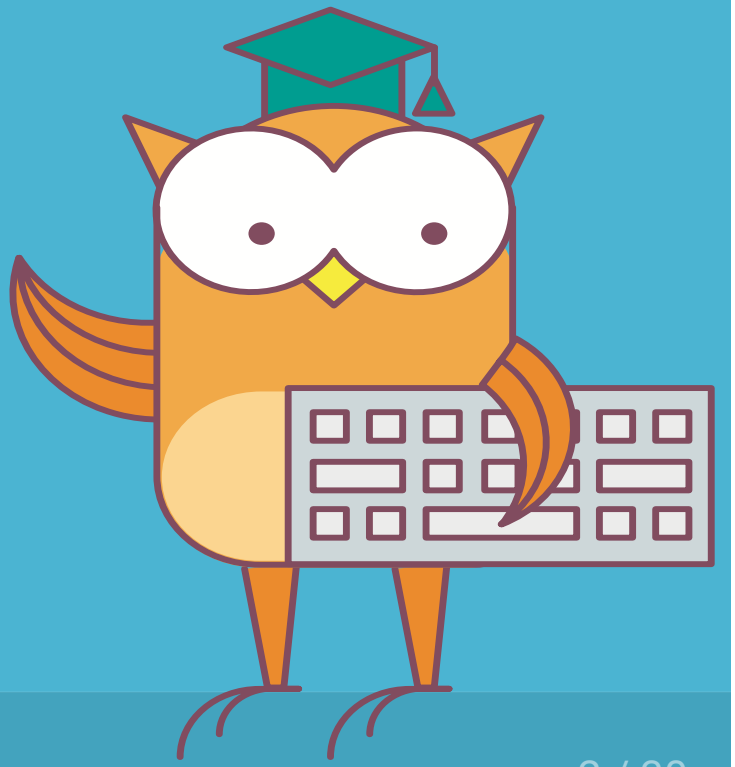
ОНЛАЙН-ОБРАЗОВАНИЕ

OTUS

Кодогенерация Go

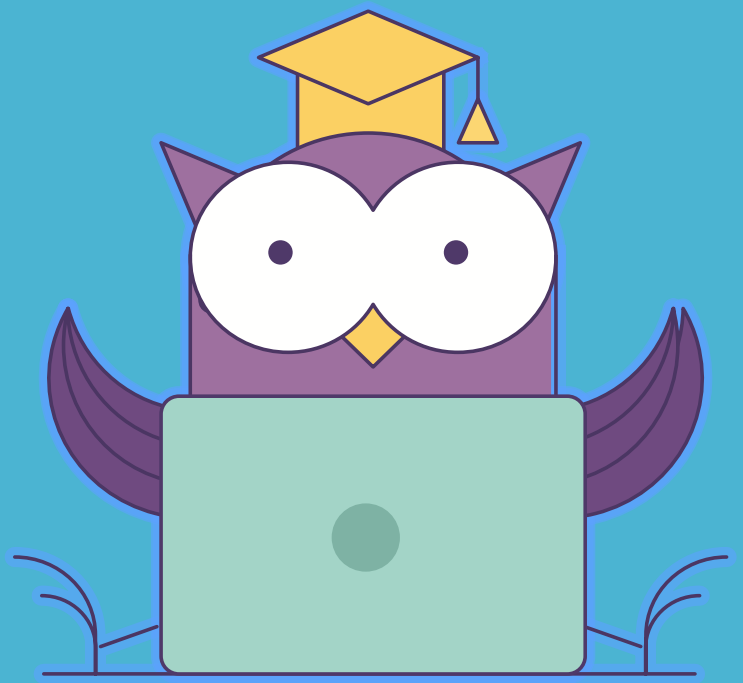
Александр Давыдов

Антон Телышев



OTUS

Как меня слышно и видно?



> Напишите в чат

\+ если все хорошо

\- если есть проблемы со звуком или с видео

!проверить запись!

План занятия

- Посмотрим, где нам может помочь генерация кода
- Посмотрим на Protocol Buffers

Go generate

https://golang.org/pkg/cmd/go/internal/generate/

```
//go:generate echo "Hello, world!"

package main

import (
    "fmt"
)

func main() {
    fmt.Println("run any unix command in go:generate")
}
```

```
> go generate
Hello, world!
```

Go generate

Псевдоним:

```
//go:generate -command foo go tool foo
```

Regexp:

```
go generate -run enums
```

Вывести команды:

```
go generate -x
```

Вывести список команд к выполнению:

```
go generate -n
```

Зачем?

- Генерировать структуры на основе JSON
- Генерировать заглушки для интерфейсов (моки для тестов)
- Protobuf: генерировать код из описания протокола (.proto)
- bindata: вставка бинарных данных (изображения, шаблоны и пр.) в код на Go в виде `[]byte`

Цикл разработки пакета с go generate

```
% edit …
% go generate
% go test

% git add *.go
% git commit   # Коммитим сгенерированный код
```

Принципы go generate

- `go generate` запускается разработчиком программы/пакета, а не пользователем
- инструментарий для `go generate` находится у создателя пакета
- генерация кода не должна происходить автоматически во время `go build`, `go get`, `go test` и пр.
- генерация кода должна вызываться явно
- инструменты генерации кода "невидимы" для пользователя и могут быть недоступны для него
- `go generate` работает только с .go-файлами, как часть тулкита go
- не забывайте добавлять disclaimer

```
^// Code generated .* DO NOT EDIT\.$
```

https://docs.google.com/document/d/1V03LUfjSADDooDMhe-_K59EgpTEm3V8uvQRuNMAEnjg/edit

Binary data

```
go get -u github.com/go-bindata/go-bindata/...
```

```
go-bindata -o myfile.go data/
```

```
//go:generate bindata -o jpegs.go pic1.jpg pic2.jpg pic3.jpg
```

```
    b, err := Asset("pic1.jpg")
    if err != nil {
        log.Fatalf("unable to get template: %v", err)
    }
```

Генерация go структур из JSON

https://mholt.github.io/json-to-go/

```
go get github.com/ChimeraCoder/gojson/gojson
```

```
{
  "name" : "Alex",
  "age": 24,
  "courses": ["go", "python"]
}
```

```
cat schema.json | gojson -name Person

package main

type Person struct {
        Age     int64    `json:"age"`
        Courses []string `json:"courses"`
        Name    string   `json:"name"`
}
```

impl: моки интерфейсов

```
go get -u github.com/josharian/impl
```

```
$ impl 'f *File' io.ReadWriteCloser
func (f *File) Read(p []byte) (n int, err error) {
    panic("not implemented")
}

func (f *File) Write(p []byte) (n int, err error) {
    panic("not implemented")
}

func (f *File) Close() error {
    panic("not implemented")
}
```

Stringer


```
go get golang.org/x/tools/cmd/stringer
```

```
func (t T) String() string
```

```
//go:generate stringer -type=MessageStatus
type MessageStatus int

const (
    Sent MessageStatus = iota
    Received
    Rejected
)
```

```
func main() {
    status := Sent
    fmt.Printf("Message is %s", status) // Message is Sent
}
```

JSON Enums

```
go get github.com/campoy/jsonenums
```

```
func (t T) MarshalJSON() ([]byte, error)
func (t *T) UnmarshalJSON([]byte) error
```

```
//go:generate jsonenums -type=Status
type Status int

const (
    Pending Status = iota
    Sent
    Received
    Rejected
)
```

easyjson

```
go get -u github.com/mailru/easyjson/...
```

```
easyjson -all <file>.go
```

генерирует MarshalEasyJSON / UnmarshalEasyJSON, для структур из файла
кратно быстрее за счет отсутствия рефлексии

иногда достаточно ldflags

```go
package main

import (
    "fmt"
)

var VersionString = "unset"

func main() {
    fmt.Println("Version:", VersionString)
}
```

```
go run -ldflags '-X main.VersionString=1.0' main.go
```

Вернемся к дженерикам

```
The generic dilemma is this: do you want slow programmers,
slow compilers and bloated binaries, or slow execution times?
(c) Russ Cox
```

Какие варианты:

- copy & paste (см. пакеты strings and bytes)
- интерфейсы

```
type Interface interface {
    Len() int
    Less(i, j int) bool
    Swap(i, j int)
}
```

- type assertions
- рефлексия
- go generate

Generics!

```
go get github.com/cheekybits/genny
```

```
go:generate genny -in=$GOFILE -out=gen-$GOFILE gen "KeyType=string,int ValueType=strin
```

Объявляем заглушки по типам:

```
type KeyType generic.Type
type ValueType generic.Type
```

Пишем обычный код:

```
func SetValueTypeForKeyType(key KeyType, value ValueType) { /* ... */ }
```

Что посмотрели:

- Моки интерфейсов: github.com/josharian/impl
- Stringer: String() для целочисленных типов: golang.org/x/tools/cmd/stringer
- Marshal/Unmarhsal для Enums: github.com/campoy/jsonenums
- Генерация структур из JSON: github.com/ChimeraCoder/gojson/gojson
- easyjson для быстрой работы с JSON
- generics при помощи кодогенерации

Больше примеров для вдохновения:
https://github.com/avelino/awesome-go#generation-and-generics

Protocol buffers

xml:

```
<person>
  <name>Elliot</name>
  <age>24</age>
</person>
```

json:

```
{
  "name": "Elliot",
  "age": 24
}
```

protobuf:

```
[10 6 69 108 108 105 111 116 16 24]
```

Protocol buffers: формат протокола

OTUS

https://developers.google.com/protocol-buffers/docs/proto3

```
syntax = "proto3";

package demo;


message People {
    repeated Person person = 1;
}

message Person {
    string name = 1;
    repeated Address address = 2;
    repeated string mobile = 3;
    repeated string email = 4;
}

message Address {
    string street = 1;
    int32 number = 2;
}
```

Protocol buffers: кодогенерация

```
go get -u github.com/golang/protobuf/protoc-gen-go
```

```
    //go:generate protoc --go_out=. file.proto
```

globbing не поддерживается:

```
    //go:generate protoc --go_out=. file1.proto file2.proto
```

Protocol buffers: кодогенерация

```
message Foo {}
```

```
type Foo struct {
}

// Reset sets the proto's state to default values.
func (m *Foo) Reset()         { *m = Foo{} }

// String returns a string representation of the proto.
func (m *Foo) String() string { return proto.CompactTextString(m) }

// ProtoMessage acts as a tag to make sure no one accidentally implements the
// proto.Message interface.
func (*Foo) ProtoMessage()    {}
```

Protocol buffers: запись и чтение

```
course := &myotus.Course{
    Title:   "Golang",
    Teacher: []*myotus.Teacher{{Name: "Dmitry Smal", Id: 1},
                               {Name: "Alexander Davydov", Id: 2}},
}

out, err := proto.Marshal(course)
if err != nil {
    log.Fatalln("Failed to encode", err)
}
```

```
otusdb := &myotus.Otus{}
if err := proto.Unmarshal(in, otusdb); err != nil {
    log.Fatalln("Failed to parse otus database:", err)
}
```

Protocol buffers: типы данных

https://developers.google.com/protocol-buffers/docs/proto3

Скаляры: default, float, int{32,64}, string, bytes
Поля: одиночные, repeated (порядок сохраняется), reserved (полезно для удаленных полей)

```
message Foo {
  reserved 2, 15, 9 to 11;
  reserved "foo", "bar";
}
```

enums, должны начинаться с 0 как default value

```
enum Corpus {
    UNIVERSAL = 0;
    WEB = 1;
    IMAGES = 2;
```

Maps:

```
map<string, Project> projects = 3;
```

Protocol buffers: задание

Сгенерировать схему/код для работы с событиями календаря:

- название
- начало
- конец
- тип события (enum: встреча, напоминание, другое)

Домашнее задание

- Сделать "заготовку" для микросервиса-календаря.
- Определить структуру определяющую событие, написать методы для добавления/изменения/удаления событий.
- Хранить события в памяти, без персистентности.

Опрос

Заполните пожалуйста опрос

https://otus.ru/polls/4905/

OTUS

Спасибо за внимание!