OTUS

Golang Developer. Professional

otus.ru

Проверить, идет ли запись

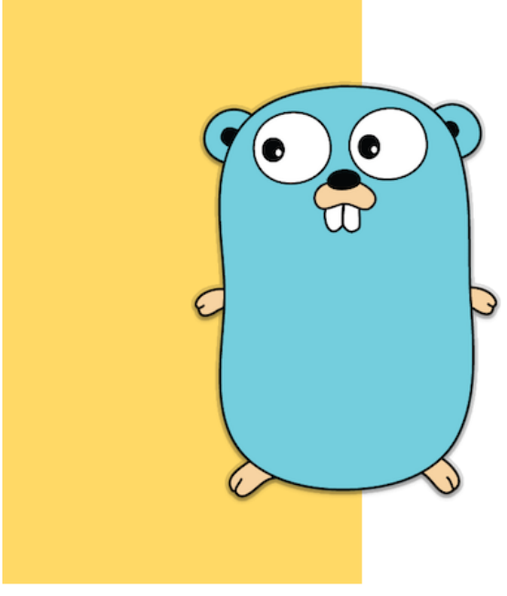
Меня хорошо видно && слышно?

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

Тема вебинара

Кодогенерация и AST

Алексей Романовский



Правила вебинара

Активно
участвуем

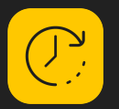
Off-topic обсуждаем
в учебной группе

Задаем вопрос
в чат или голосом

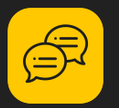
Вопросы вижу в чате,
могу ответить не сразу

Условные обозначения

Индивидуально



Время, необходимое
на активность



Пишем в чат

Говорим голосом

Документ

Ответьте себе или
задайте вопрос

План занятия

- Поговорим о кодогенерации
- Посмотрим, где она может нам помочь
- Посмотрим что такое AST

Кодогенерация

- Что это такое?

Кодогенерация

- Зачем она нужна?
- Какие задачи помогает решить?

Зачем нужна кодогенерация

- Генерировать код по метаописанию (swagger, protobuf, ...)
- Генерировать заглушки для интерфейсов
- Встраивать данные в код

Пример в стандартной библиотеке:
https://golang.org/src/unicode/tables.go

Go 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!
```

https://github.com/golang/go/blob/master/src/cmd/go/internal/generate/generate.go

Цикл разработки пакета с go generate

```
% edit …
% go generate
% go test

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

Принципы go generate

Go generate: A Proposal

- Запускать на машине разработчика пакета, а не пользователя.
  - утилиты для генерации нужны только разработчику
  - генерация не происходит автоматически при go get
- Добавлять disclaimer.

```
/*
* CODE GENERATED AUTOMATICALLY WITH tool name
* THIS FILE SHOULD NOT BE EDITED BY HAND
*/
```

- Работать только с .go-файлами, как часть тулкита go.

Go generate

псевдоним:

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

exec:

```
go generate -run enums
```

выводить команды:

```
go generate -x
```

список команд к выполнению:

```
go generate -n
```

Awesome go: generators

https://github.com/avelino/awesome-go?tab=readme-ov-file#generators

Go embed (since go 1.16)

https://golang.org/pkg/embed/

```
//go:embed static/gopher.png
var gopherPngBytes []byte
```

Заменяет bindata

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
}
```

Генерация Marshal/Unmarshal: easyjson

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

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

Генерирует MarshalEasyJSON / UnmarshalEasyJSON, для структур из файла
Быстрее за счет отсутствия рефлексии

P.S. https://github.com/json-iterator/go

Реализация интерфейсов: impl

```
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")
}
```

Моки интерфейсов: gomock

```
GO111MODULE=on go get github.com/golang/mock/mockgen@latest
```

```
//go:generate mockgen -source=$GOFILE
//-destination ./mocks/mock_getter.go -package mocks Getter
type Getter interface {
    Get(url string) (resp *http.Response, err error)
}
```

AST

AST - это Abstract Syntax Tree, то есть дерево, которое в абстрактном виде представляет структуру программы. AST создаётся парсером по мере синтаксического разбора программы. В современных компиляторах AST и список диагностик (ошибок, предупреждений) - это два результата вызова модуля синтаксического разбора.

AST: свойства

- AST не является бинарным деревом: например, у унарного оператора будет один дочерний узел
- AST является гетерогенным деревом, состоящим из узлов разного типа
- В этом AST похож на DOM-представление документа HTML/XML
- В каждом поддереве дочерними узлами становятся лексически вложенные сущности: например, для узла объявления функции дочерними узлами являются инструкции, составляющие тело функции, а также объявления параметров функции (если они выделены в отдельные узлы AST волей автора компилятора)

AST: go

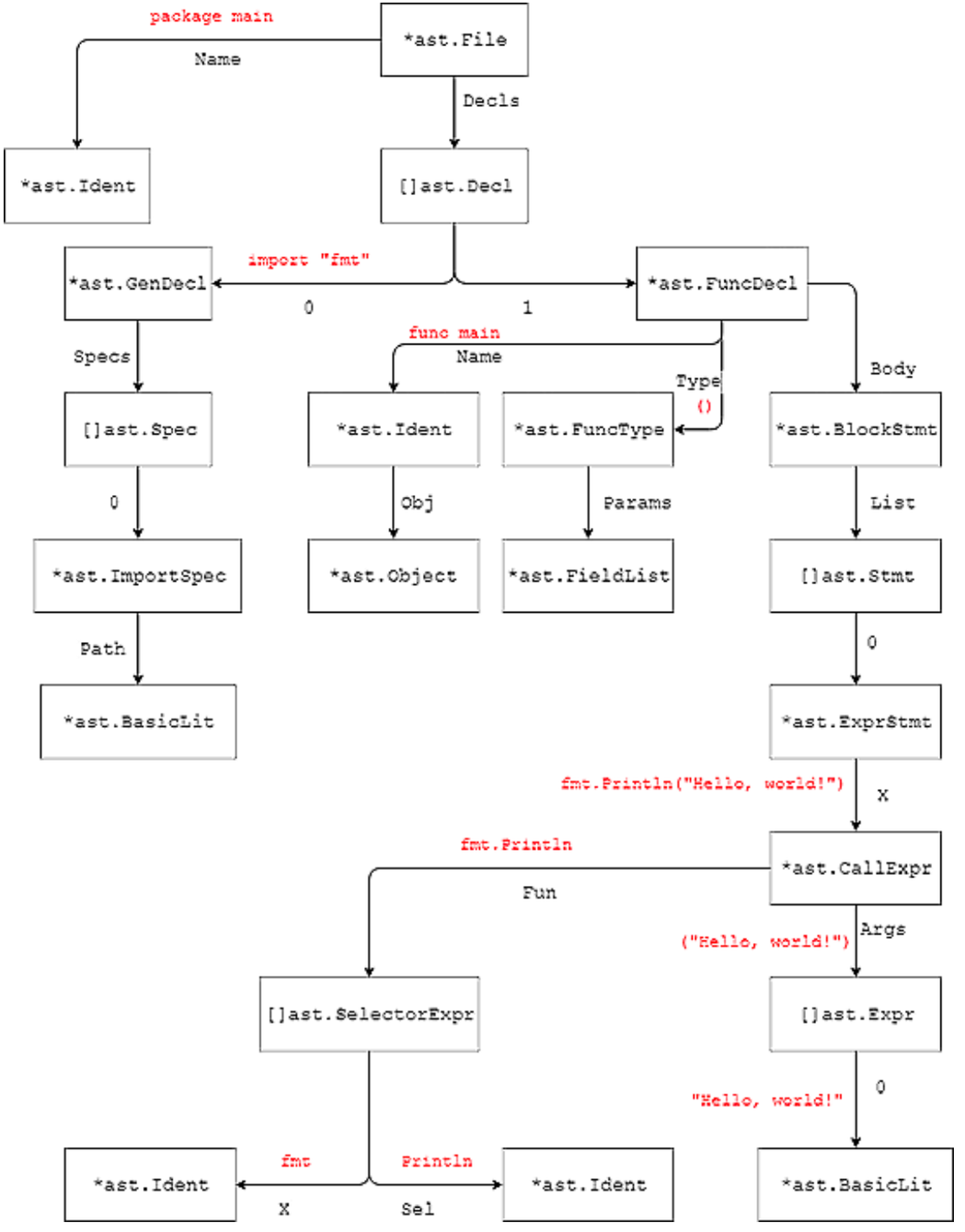
- go/ast — декларирует типы дерева разбора;
- go/parser — разбирает исходный код в эти типы;
- go/printer — выливает AST в файл исходного кода;
- go/token — обеспечивает привязку дерева разбора к файлу исходного кода.

Пример разбора

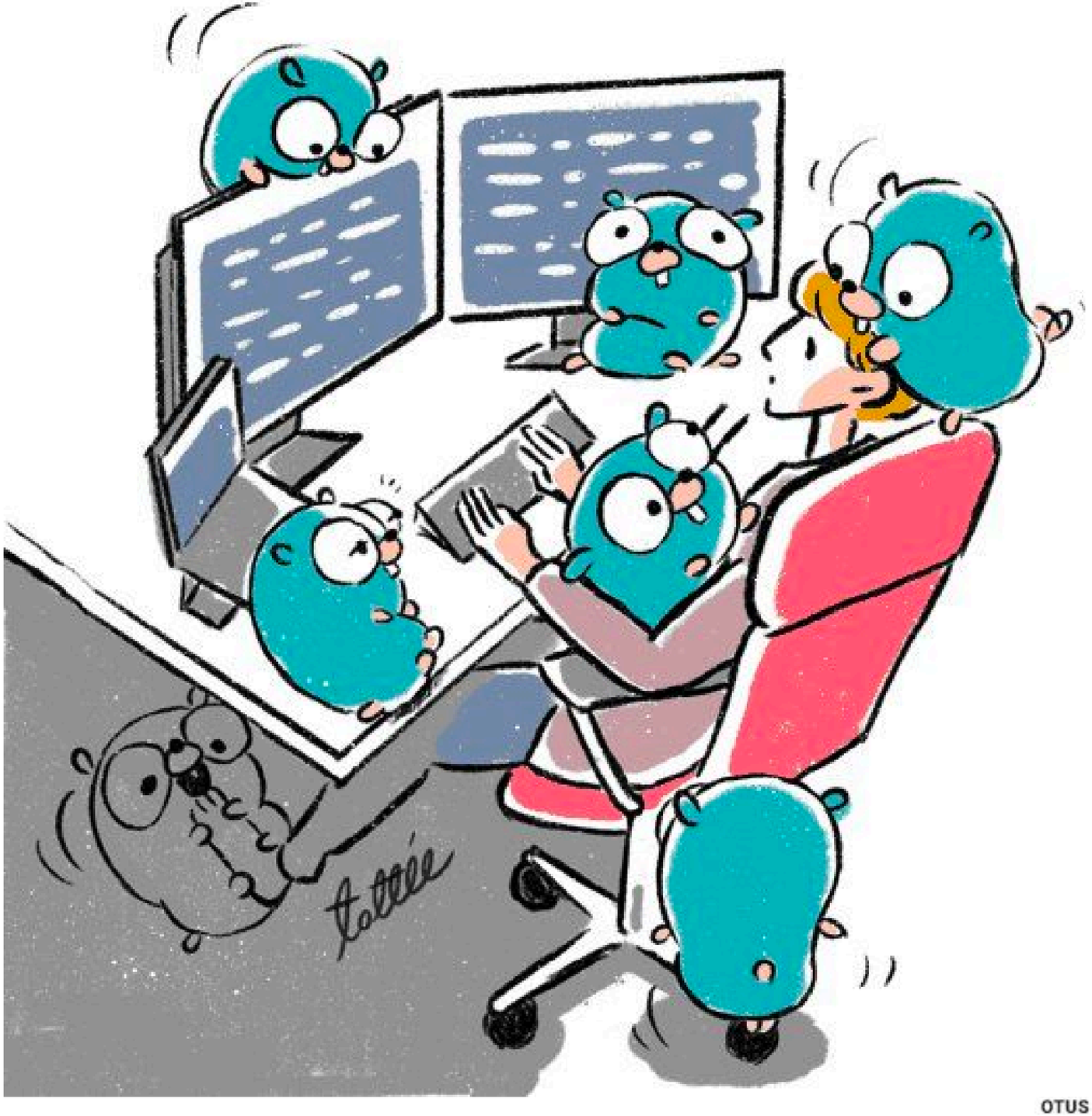
```
package main

import "fmt"

func main() {
    fmt.Println("Hello, World!")
}
```

- https://astexplorer.net/

Пример разбора



Практика



Что посмотрели:

- встраивание данных в код
- Stringer: String() для целочисленных типов: golang.org/x/tools/cmd/stringer
- easyjson для быстрой работы с JSON
- моки интерфейсов: github.com/josharian/impl
- AST и пример использования

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

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Заполните, пожалуйста, опрос о занятии по ссылке в чате

Спасибо за внимание!

Приходите на следующие вебинары

Алексей Романовский