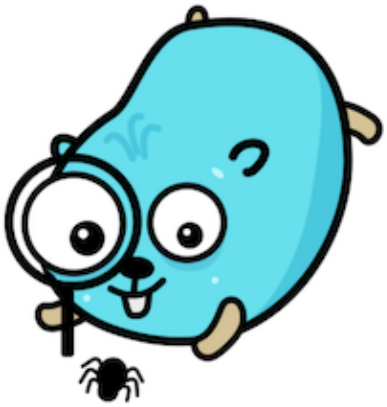
Линтеры и ошибки (часть 2)

Продолжим тему линтеров, связанных с ошибками в Go.



forbidigo

`Forbids identifiers.`

Линтер, **выключенный** по умолчанию в **golangci-lint** и запрещающий использовать строки кода, попадающие в заданные "запрещённые" паттерны.

Из коробки отлавливает `fmt.Print.*`, чтобы ваш отладочный код случайно не попал в релизную версию приложения.

Конфигурация линтера выглядит следующим образом:

```
linters-settings:
  forbidigo:
    forbid:
      - fmt.Errorf # consider errors.Errorf in github.com/pkg/errors
      - fmt.Print.* # too much log noise
      - ginkgo\\.F.* # these are used just for local development

    exclude_godoc_examples: false
```

Как он связан с ошибками?

Нас заинтересовало, что в предлагаемых авторами паттернах (см. выше) есть пример запрета использования `fmt.Errorf` в пользу **github.com/pkg/errors**.

Мы подробно рассматриваем данный модуль в соответствующем уроке, а что использовать – выбирать вам.

ruleguard

*analysis-based Go linter that runs dynamically loaded rules. You write the rules, **ruleguard** checks whether they are satisfied.*

Линтер, помогающий писать собственные правила. **Выключен** по умолчанию в **golangci-lint**.

Полезен, когда:

- правило слишком тривиально для целого отдельного линтера;
- правило специфично только для отдельно взятого проекта;
- нет готового линтера, реализующего правило, и вам влом создавать его :)

В контексте ошибок он полезен в случаях, когда обычных регулярок для **forbidigo** не хватает.

Например, мы хотим отслеживать отсутствие врапинга через **github.com/pkg/errors** (исходник примера).

Заводим правило:

```
package gorules

import "github.com/quasilyte/go-ruleguard/dsl"

// wrapWithPkgErrors - правило, которое предлагает врапить ошибки через github.com/pkg/errors.
```

```
func wrapWithPkgErrors(m dsl.Matcher) {
    m.Import("github.com/pkg/errors")

    m.Match(`if err := $x; err != nil { return err }`).
        Report(`err is not wrapped`).
        Suggest(`if err := $x; err != nil { return errors.Wrap($x,
"FIXME: wrap the error") }`)

    // Прочие варианты.
    // ...
}
```

Применяем **ruleguard** к нашим сорцам:

```
package example

import "github.com/pkg/errors"

func foo() error {
    if err := bar(); err != nil {
        return err
    }
    return nil
}

func bar() error {
    return errors.Wrap(errors.New("bad request"), "do request")
}
```

```
$ ruleguard -rules gorules/rules.go -fix example.go
example.go:8:2: wrapWithPkgErrors: err is not wrapped (rules.go:9)
```

Получаем автофикс:

```
func foo() error {
    if err := bar(); err != nil {
        return errors.Wrap(bar(), "FIXME: wrap the error")
    }
    return nil
```

```
}
```

В общем, интересный линтер, с помощью которого можно сильно упростить себе процесс ревью.

Подумайте на досуге, какие правила можно было бы создать для вашего проекта.

depguard

Go linter that checks package imports are in a list of acceptable packages.

Выключен по умолчанию в **golangci-lint**.

В контексте работы с ошибками полезен, когда вы хотите запретить использование в проекте внешних модулей:

```
linters-settings:
  depguard:
    list-type: blacklist
    include-go-root: false
    packages:
      - github.com/pkg/errors
      - github.com/cockroachdb/errors
    packages-with-error-message:
      - github.com/pkg/errors: "only std errors & fmt.Errorf are allowed"
      - github.com/cockroachdb/errors: "only std errors & fmt.Errorf are allowed"
```

errname

Checks that sentinel errors are prefixed with the `Err` and error types are suffixed with the `Error`.

Выключен по умолчанию в **golangci-lint**.

Чисто стилевой линтер, проверяющий, что вы следуете лучшим практикам, о которых мы говорили в шаге **"Нейминг переменных и типов ошибок"**.

```
// Плохо.
type DecodeErr struct{}
func (d *DecodeErr) Error() string { /*...*/ }

// Хорошо.
type DecodeError struct{} // или decodeError

func (d *DecodeError) Error() string { /*...*/ }

// Плохо.
var InvalidURLErr = errors.New("invalid url")

// Хорошо.

var ErrInvalidURL = errors.New("invalid url") // или errInvalidURL
```

Увеличивает долю единообразия в вашем коде, а также упрощает поиск "ошибочных" типов и sentinel ошибок.

nilnil

Checks that there is no simultaneous return of `nil` error and an invalid value.

Выключен по умолчанию в **golangci-lint**.

Линтер, позволяющий избегать конструкций, о которых мы говорили ранее в шаге **"Гарантия результата при отсутствии ошибки"**:

```
// Плохо.

func (rl *http2clientConnReadLoop) handleResponse(/* ... */)
(*Response, error) {
    if statusCode >= 100 && statusCode <= 199 {
        return nil, nil // <- Опасно и неочевидно.
    }
}

// Хорошо.
```

```
var errNeedSkipFrame = errors.New("need skip frame")

func (rl *http2clientConnReadLoop) handleResponse(/* ... */)
(*Response, error) {
    if statusCode >= 100 && statusCode <= 199 {
        return nil, errNeedSkipFrame
    }

}
```

В базовой конфигурации линтер ругается на `return nil, nil` для следующих типов, использование непроинициализированных значений которых может привести к панике:

- указатели, функции и интерфейсы (`panic: invalid memory address or nil pointer dereference`);
- мапы (`panic: assignment to entry in nil map`);
- каналы (`fatal error: all goroutines are asleep - deadlock!`).