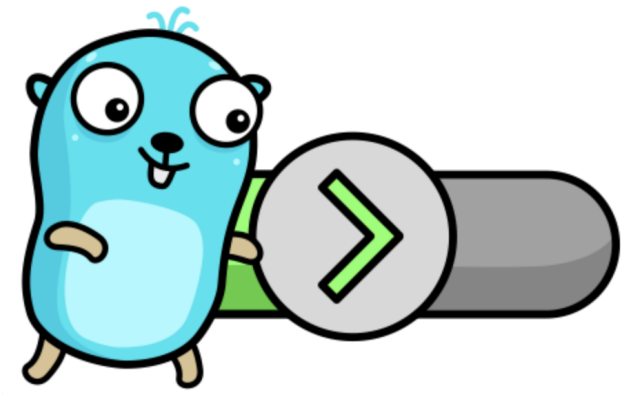
Оборачивание io.EOF и иже с ним

В этом уроке мы поговорим об очередном исключении из правил, а именно о том, что не нужно оборачивать ошибки из пакета `io` при реализации соответствующих интерфейсов пакета.



История вопроса

Когда врапинг ошибок после релиза Go 1.13 стал повсеместным, код, напрямую сравнивающий ошибки между собой, перестал работать.

```
// Не будет работать, если кто-то сделает вместо `return err`,
например,
// `return fmt.Errorf("cannot do op: %w", err)`
if err == externalpkg.ErrExpected {
    // ...
}

// Так уже лучше.
if errors.Is(err, externalpkg.ErrExpected) {
    // ...

}
```

Аналогичная ситуация при использовании **type switch** по типу ошибки.

Соответственно сообществом был написан ряд линтеров (подробнее о них поговорим позже), отлавливающих ситуации выше.

Например, [polyfloyd/go-errorlint]:

```
$ go-errorlint example.go

example.go:9:5: comparing with == will fail on wrapped errors. Use
errors.Is to check for a specific error
```

Но, как водится, оказалось не всё так просто. Бывают исключения из этого правила, например, есть [мнение], что некоторые ошибки вроде `io.EOF` оборачивать **нельзя**.

Почему нельзя врапить io.EOF?

Сторонники данного тезиса аппелируют к тому, что в [документации] метода `Read` интерфейса `io.Reader` говорится о том, что метод должен возвращать ошибку `io.EOF` саму по себе, а не обёрнутую:

Read must return EOF itself, not an error wrapping EOF, because callers will test for EOF using ==.

Да, что там говорить, сам Роб Пайк одобряет.

Столь сильные аргументы развязывают нам руки, для того чтобы можно было ими писать конструкции вида:

```
if err == io.EOF {
    // Do something.

}
```

Что там у линтеров?

Линтерам пришлось прогнуться и поддержать подобные исключения, чтобы не давать ложноположительные срабатывания.

Можно заметить, что кроме `io.EOF` в этот список входят следующие ошибки:

- `io.ErrClosedPipe`;
- `io.ErrShortBuffer`;
- `io.ErrUnexpectedEOF`;
- `sql.ErrNoRows`.

Т.е. те ошибки, которые согласно документации к соответствующим функциям, возвращаются из них в "чистом" виде.

Мнение

На самом деле вся эта картина выглядит не очень хорошо. Аргументы, приведённые сторонниками тезиса о том, что ошибки вроде `io.EOF` оборачивать нельзя, сами по себе сильны, но есть проблемы.

Первая проблема в том, что соглашение о возврате `io.EOF` из метода `Read` появилось задолго до того, как сформировалась концепция обработки ошибок в Go 1.13, и сейчас все вынуждены поддерживать обратную совместимость, держа в уме данный нюанс. И вроде бы неправильно напрямую сравнивать ошибки, но если очень хочется, то можно. Потому что в официальной документации так написано.

Проблема вторая как раз и заключается в том, что это соглашение на уровне документации (по сути комментария к методу). Оно никем не может контролироваться и не даёт никаких гарантий (хотя все говорят, что даёт).

Например, разработчик какой-либо библиотеки, которую вы используете, может не знать или забыть про это соглашение, и возвращать вам не голый `io.EOF`, а обернутый в несколько слоёв (как вы в детстве, когда выходили кататься на горку). Написанная вами конструкция `if err == io.EOF` работать не будет. Что делать? Варианта, похоже, три: править библиотеку, не использовать её или использовать `errors.Is(err, io.EOF)`.

Да, что там говорить, сам Роб Пайк [одобряет](#).

Апелляция к авторитету – не лучший аргумент: не всегда одного авторитетного мнения достаточно, чтобы докопаться до истины. И опять же – одобрение кого-либо на запрет оборачивания не даёт никаких гарантий, что никто так делать не будет.

Так что делать?

Мы склоняемся к тому, что

- если вы реализуете один из интерфейсов `io`, то лучше следовать документации к методам и возвращать ошибки в чистом виде, как просят разработчики языка;

- если же вы проверяете ошибку от вызова одного из `io`-методов, то проще всего даже `io.EOF` проверять не через прямое сравнение `==`, а через `errors.Is(err, io.EOF)`, тем самым поддержав любую из двух реализаций и не думать об этом.

А что думаете вы? Пишите в комментариях.