## Текстовый вариант видеоурока из предыдущего шага

Следующая важная тема наших уроков – это обработка ошибок в языке Go. Здесь мы узнаем, что ошибки в Go – это, на самом деле, интерфейс error и научимся им пользоваться на практике. Разберемся, почему же всё-таки в Go не стали реализовывать exception-ы, а сделали это ошибками. Познакомимся с паниками и с методами восстановления после них.

В Go нет исключений. Почему же так произошло? Исторически сложилось всё это таким образом, что во всех языках программирования, в которых есть исключение, разработчики начали использовать их где попало. То есть исключения подразумевали под собой что-то исключительное: в нашей программе что-то серьезное произошло, и мы бросаем exception. Но на деле exception-ы начали использовать где попало и в любой удобной и непонятной ситуации, поэтому в Go решили подойти более осознанно к этой проблеме и разделили ситуацию на 2 части. То есть у нас есть ошибки, которые находятся под управлением разработчика. И есть паники, которые бросаются системой в каких-то исключительных ситуациях, когда у нас что-то серьезное сломалось, и приложение должно остановить свою работу.

И мы сказали, что ошибки в Go – это interface. То есть получается, что любой тип данных, которые реализует методы этого interface-а – в Go может выступать в качестве ошибки. Как вы видите для того, чтобы стать ошибкой, типу необходимо реализовать всего 1 метод – это метод Error, возвращающий строку. Как раз-таки эта строка и будем сообщением нашей ошибки.

Чтобы создать ошибку в языке Go, есть несколько методов. Для этого мы можем воспользоваться встроенным в язык пакетом errors. В этом пакете есть метод New, в который мы можем передать наши сообщения об ошибке и в итоге получить сформированную ошибку. Помимо этого метода можно воспользоваться пакетом

fmt и методом Errorf – в этом случае также поддерживается строка с сообщением об ошибке, но в строке могут быть дополнительные команды, например, как «%T», что означает, что мы в это место вставим тип переменной, которая у нас идет после этой строки.

Помимо такого формата, функция Errorf ещё поддерживается создание вложенных ошибок. Что это значит? В данном примере у нас создается ошибка с текстом «wrapped err», и вместо команды «%w» мы вкладываем другую ошибку. То есть здесь, получается, создается цепочка ошибок одна вложенная в другую.

При этом у нас поддерживается несколько методов для работы с цепочками ошибок. Как вы видите, в первой строке нашего приведенного кода в переменную prevErr мы получим результат функции Unwrap, вызванной из пакета errors. Что эта функция делает? Как мы видели на предыдущем слайде, мы можем создавать вложенные ошибки. В данном случае функция Unwrap принимает нашу переменную с ошибкой currentErr, и если она вложенная, то мы распаковываем эту ошибку и в переменную prevErr получаем ошибку, которая была вложена внутрь currentErr.

Вложенность ошибок даёт нам возможность ещё делать некоторые проверки. Как вы видите в следующей строке в проверке if мы путем вызова метода пакета errors, метод Is проверяем, является ли ошибка err ошибкой, которая лежит в переменной target. То есть в этом случае мы пробегаемся по всей цепочке ошибок в переменной err и сравниваем эти ошибки с ошибкой из переменной target. Если хотя бы 1 ошибка совпадает, то считается, что проверка прошла.

А в следующем примере мы используем функцию As. В чём её отличие? В том, что в случае с Is мы проверяем конкретное значение ошибки, а в случае As мы проверяем, является ли ошибка данного типа. То есть мы можем объявить

какую-нибудь структуру ImportMissingError, и она может реализовать интерфейс ошибок. То есть у этой структуры может быть метод error, следовательно, её можно использовать как ошибку. И в данном примере мы можем проверить цепочку ошибок, которая лежит в переменной err на предмет соответствия типу ImportMissingError.

Стоит отметить, что цепочку ошибок можно создавать только с версией Go 1.13, потому что эта возможность появилась именно с этой версии. Посмотрим, как можно работать с ошибками на примерах в коде.

Здесь у меня объявлен метод getClient, который, как вы видите, не принимает никаких параметров, но возвращает 2 результата. 1-й – это структура client и 2-м результатом – тип ошибки error. Давайте посмотрим и удостоверимся, что это действительно interface.

Как мы видим, в коде библиотеки Go error объявлен как Interface, и у него есть метод Error, который должен реализовать какой-либо тип данных, чтобы соответствовать типу ошибки в языке. Поэтому мы здесь просто указываем, что метод может вернуть какой-то результат в виде нужной нам структуры, а также может вернуть ошибку. Такой подход принят в Go и используется очень часто. То есть метод может вернуть настоящий результат работы либо если там что-то пошло не так, и ошибку в том числе. Поэтому здесь мы создаем пустую структуру Client и следом после этого создаем ошибку. То есть возникла какая-то ошибка – мы создаем её с помощью пакета errors и вызова функции New, передаём внутрь нашу…, наше сообщение об ошибке. И в итоге возвращаем пустую структуру и ошибку. В методе main как раз мы вызываем функцию getClient и принимаем как результат, данные в соответствующие переменные.

Дальше нам обязательно нужно проверить, вернул ли метод ошибку? – Это даёт нам понимание, можно ли доверять данным, которые вернул наш метод. Если ошибки никакой не произошло, значит, мы считаем, что в методе всё прошло успешно и данным, которые возвращает наш метод, можно доверять и дальше с ними работать. Но если случилась какая-то ошибка, то предполагается, что структура Client какая-то, может быть, кривая, потому что в методе getClient случилась ошибка. Поэтому мы сначала проверяем на ошибку. Если возникла ошибка, то мы её выводим и выходим из метода. Если же ошибки никакой не произошло – мы выводим нашу структуру Client. Давайте запустим и посмотрим на результат.

Как мы видим, здесь мы просто вывели наше сообщение об ошибке. Если же никакой ошибки не произошло, то мы, например, в методе getClient можем вместо err вернуть nil. Так как error – это Interface, его значение может быть nil, потому что как мы помним, Interface-ы мы проверяем по ссылке. В этом случае у нас эта проверка уже не сработает, и мы в результате должны увидеть вывод нашей структуры. Давайте запустим и посмотрим на результат.

Давайте попробуем создать свою собственную ошибку. В этом же файле у меня реализована структура ServerError, у которой есть всего одно поле: err, которое соответствует interface-у ошибок. Но сама структура также соответствует interface-у, потому что, как вы видите, здесь у меня есть реализация метода Error, которая необходима, чтобы быть ошибкой. То есть у меня (НРЗ 09:08) тоже подсказывает, что данный метод удовлетворяет interface error. Поэтому структуру ServerError можно использовать в качестве ошибки.

Давайте мы здесь напишем метод, который создает нашу структуру: NewServerError, который как параметр будет принимать сообщения и возвращать interface Error. И здесь мы просто сделаем следующее: вернем созданную

структуру ServerError, внутри которой засунем поле таким образом. Единственное, здесь нужно передать…, возвращать это по ссылке. Таким образом, вызвав функцию NewServerError, мы возвращаем в результате новую структуру ServerError, но в самом методе мы описываем, что это должен быть interface Error, то есть это ошибка.

Теперь давайте здесь вместо того, что мы написали, сделаем следующим образом: NewServerError и здесь напишем сообщение. И, соответственно, нам надо вернуть нашу ошибку. Давайте запустим и посмотрим, что получилось в итоге.

Результат мы получили абсолютно тот же, что и был раньше. Для чего же тогда мы заморачивались и делали свою структуру? Чтобы лучше это прочувствовать, давайте попробуем сделать вложенные ошибки. Как мы помним, вложенность ошибок можно делать с помощью функции Errorf из пакета fmt. Таким образом, и здесь мы можем дать команду «%w». Дальше нам нужно написать базовую ошибку, которая будет вложена в новую. Давайте базовую ошибку сделаем объявленную заранее, которая будет обозначать, что произошло внутренняя ошибка. Объявим переменную internalErr, которая будет в значении содержать нашу структуру NewServerError. И придумаем какое-нибудь сообщение об ошибке.

Таким образом, в нашем методе getClient мы можем обернуть этот метод. Давайте посмотрим, что получим в результате. Смотрите, здесь у нас уже дополнилось наше сообщение: «ошибка получения клиента», и дальше мы дополнили: «внутренняя ошибка». Но при этом если бы мы не оборачивали… Смотрите, в Errorf если мы используем команду «%w» – это получается, что мы вкладываем ошибку, то есть у нас в результате будет вложенная ошибка. Но если мы используем «%v», что означает: «вставить сюда значение», то мы просто создадим новую ошибку, у которой будет сообщение склеено из этой переменной

и данного сообщения. То есть смотрите, здесь результат будет абсолютно такой же, только если у нас есть вложенность – мы можем сделать следующим образом. Здесь мы сможем проверить вот так. То есть мы проверяем, является ли наша ошибка внутренней ошибкой. Если является – давайте выведем: «внутренняя ошибка».

Итак, запустим, посмотрим на результат. Да, и смотрите, мы прошли проверку на InternalError, вывели: «произошла внутренняя ошибка», а потом вывели сам текст ошибки. Если бы мы здесь не сделали вложение, то такой проверки бы не получилось.

По сути, метод Is просто в цикле распаковывает нашу ошибку, проходя по всей цепочке, и проверяет на соответствие тому, что мы передали. Происходит это следующим образом: смотрите, у нас также доступен метод Unwrap, про который мы говорили, как раз, который распаковывает нашу ошибку. Что значит «распаковывает»? – Мы просто по цепочке двигаемся на 1 ошибку дальше. То есть у нас есть верхняя ошибка и вложенная внутри ошибка. Метод Unwrap даёт следующую ошибку, которая вложена. Давайте выведем NewErr и посмотрим на результат.

Смотрите, NewErr вывел нам: «внутренняя ошибка», так как вложенная ошибка у нас является этой: internalErr. Если мы не будем оборачивать ничего, то получим в качестве результата Unwrap значение nil. Получили мы nil из-за того, что оборачиваний никаких нет, а значит, мы дошли до конца. То есть, в данном случае у нас всего…, цепочка состоит всего из 1-й ошибки.