Текстовый вариант видеоурока из предыдущего шага

И мы продолжаем, и на повестке у нас тема: «Паник». Паник в языке Go – это встроенная функция, которая создает ошибку runtime-а. Обычно паники используются, когда что-то происходит очень критичное и серьезное. Как правило, после паник сервис наш крошится. Кто-то может найти какое-то сходство с исключениями, но помните, что паника – это не исключения, и не нужно ни в коем случае их использовать как исключения. Вообще в Go принято придерживаться принципа: Don’t panic! Что означает: по возможности мы не кидаем паники. И стараемся обходиться только ошибками.

Теперь мы с вами знаем, что паника – это функция. Вызвать её можно следующим образом – это функция panic, в которую мы передаем сообщение, описывающее нашу проблему. В данном случае вызов этой функции приведет к пробросу паники выше по stack-у. И, если она больше нигде не обработается, в результате у нас произойдёт аварийное завершение нашей программы.

Как с этим можно работать? Допустим, у нас есть функция foo, в которой вызывается паника. Вот мы вызываем функцию panic и передаем наше сообщение. Функцией main мы сначала выводим «Start», чтобы сигнализировать сначала нашей программы, потом вызываем функцию foo и в итоге выводим на экран «End». Давайте запустим и посмотрим, что выведется на экран.

Смотрите, здесь мы напечатали Start, и после в функции foo произошла паника. Мы вывели информацию о том, что это паника и наше сообщение. Дальше в панике выводится (HP3 02:00), чтобы сообщить нам: где именно паника произошла. Как мы видим, после вызова паники наше приложение вышло со status-ом 2. То есть до этой части мы не дошли – у нас приложение остановилось с аварийным завершением.

Но чтобы избежать аварийного завершения нашего приложения, нам необходимо каким-то образом перехватить панику и восстановиться после неё. Для этого в Go есть функция recover() – она как раз перехватит нашу панику и в результате вернет ошибку. Тем самым, мы предотвратим аварийное завершение приложения. Если на пути stack-а где-то паника встретит функцию recover(), то мы восстановимся из паники, и аварийного завершения не произойдёт.

Давайте восстановимся после нашей паники. Попробуем после функции foo(), которая паникует, вызвать recover(), и выведем ошибку, которая вернет recover. Запустим наше приложение, и мы видим, что ничего не происходит. Паника как была, так и осталась. На самом деле так случается, потому что на момент паники выполнение функции приостанавливается. То есть в данной точке мы остановили выполнение и начали пробрасывать панику выше по stack-у, поэтому к этой части мы даже не доходим. Как же тогда нам восстанавливаться после паники?

Функцию recover() обычно использую в конструкции defer. Следующим образом это можно сделать. Так как defer гарантировано выполнится перед тем, как функция main() завершится, то мы можем восстановить панику именно в этой конструкции. Давайте напишем, что «Восстановление» тут происходит. И посмотрим на наш результат.

Как мы видим, здесь уже как таковой паники не происходит, – мы просто выводим ошибку. Ошибка, хочу заметить, с тем же текстом, с которым мы создали панику. Но как вы видите, ни (НРЗ 04:18), ни других признаков паники здесь нет. Мы, на самом деле, всё равно остановились на этой точке и не дошли до этой строчки, но теперь наше приложение не завершилось аварийно, а вышло, потому что функция main() завершила свою работу как полагается.

Посмотрим ещё раз, как происходят паники. Допустим, у нас есть функция bar(), которая паникует. Мы объявляем функцию foo(), которая в defer восстанавливается из паники методом recover и в цикле функция foo() вызывает bar() 10 раз. Но и, конечно же, функция main(), которая вызывает функцию foo().

Как будем формироваться наш stack вызовов? Сначала на stack-е появится функция main(), потому что это входная точка нашего приложения. Дальше из main-а произойдет вызов функции foo(), и в первом же цикле функции foo() вызовется функция bar(). Как раз здесь в функции bar() произойдет паника, которая остановит дальнейший вызов функций и вернет панику обратно в foo(), потому что мы прокидываем её по stack-у. Но в функции foo() паника обработается в конструкции defer, поэтому здесь мы остановим панику и прокинем её дальше в main(). Таким образом, здесь никакого аварийного завершения не будет, и мы завершим выполнение нашего приложения в обычном штатном режиме.

А что произойдет, если мы в функции foo(), bar() будем вызывать в горутине? Таким образом, это происходит. Тот же цикл у нас остается, но здесь мы уже функцию bar() вызываем в отдельной горутине.

Давайте посмотрим, как себя ведет stack. Всё также у нас на stack кладется вызов функции main(), потом у нас вызывается функция foo() и здесь, на самом деле, запускаются горутины bar(). И у горутин есть такая особенность, что когда горутина запускается, у них создается свой собственный stack, то есть они уже про stack main() ничего не знают. Поэтому когда мы бросим панику в горутине с функцией bar(), то наше приложение аварийно завершится, потому что проброс по stack-у будет происходить, но до функции foo() никак не дойдет, потому что у функции bar(), так как она в отдельной горутине, свой stack вызовов. И на это очень часто попадаются, думаю, что defer, который находится в foo() – обработает и

восстановит нашу панику, но, на самом деле, этого не происходит, и наше приложение падает.

Итак, при обработке наших ошибок давайте помнить о том, что в языке Go ошибки – это значения, то есть любой тип данных, который реализует interface error может выступать в качестве ошибки. В Go с версии 1.13 можно создавать вложенные ошибки – это позволяет нам делать цепочку из ошибок и проходиться по ней в поисках нужного нам типа.

Паники в языке – это не исключения, давайте не будем использовать паники в качестве исключение. Они нужны только для серьезных ситуаций, которые приводят к завершению программы.

Помним, что восстановление из паник в горутинах происходят с небольшим нюансом, так как у горутины свой stack, то recover нужно делать на новом stack-е. Если мы будем делать recover на stack-е функции, где вызывается горутина, то никакого восстановления происходит не будет.

И не забывайте, конечно же, про принцип: Don’t PANIC! По возможности, если можно заменить панику ошибкой, то лучше это сделать в виде ошибки.