Go_Day09-1

Развилка из недоступного проекта

Обновление с 5
Администратор создал 5 месяцев назад

Имя	Последний коммит	Последнее обновление
образцы кода	Инициировать фиксацию	1 год назад
образцы данных	Инициировать фиксацию	1 год назад
наборы данных	Инициировать фиксацию	1 год назад
материалы	Обновление с 5	5 месяцев назад
разное	Инициировать фиксацию	1 год назад
источник	Инициировать фиксацию	1 год назад
.gitignore	Обновление с 3	1 год назад
ИЗМЕНЕНИЙ	Инициировать фиксацию	1 год назад
ЛИЦЕНЗИЯ	Инициировать фиксацию	1 год назад
README.md	Обновление с 4	7 месяцев назад

README.md

День 09. Учебный лагерь

Повседневные дела

Содержание

Глава I

Основные правила

- Ваши программы не должны неожиданно завершать работу (выдавая ошибку при правильном вводе). Если это произойдет, ваш проект будет считаться неработоспособным и получит 0 баллов при оценке.
- Мы рекомендуем вам создать тестовые программы для вашего проекта, даже если эту работу не нужно будет отправлять и она не будет оцениваться. Это даст вам возможность легко проверить свою работу и работу ваших коллег. Вы найдете эти тесты особенно полезными во время вашей защиты. Действительно, во время защиты вы можете использовать свои тесты и/или тесты коллеги, которого вы оцениваете.
- Отправьте свою работу в назначенный репозиторий git. Оцениваться будет только работа в репозитории git.

- Если ваш код использует внешние зависимости, он должен использовать модули Go для управления ими.

Глава II

Правила дня

- Вы должны только включить чернила `\*.go` , `\*_test.go` файлы и (в случае внешних зависимостей) `go.mod` + `go.sum`
- Ваш код для этой задачи должен быть собран с помощью всего лишь `go build`
- Все ваши тесты должны быть запущены путем вызова стандартного `go test ./...`

Глава III

вступление

Иногда мы слышим, что есть люди, которые «могут делать несколько дел параллельно». Хотя теоретически возможно делать совершенно разные вещи (скажем, разными руками), эта фраза обычно относится к людям, выполняющим несколько задач одновременно, но НЕ параллельно. Что я имею в виду?

"Parallel" in computer science usually means that progress is made on more than one task at the same time. But with people it is a bit different - the real trick is to keep the context and switch over from one task to another quickly. From the side it may even look like "parallelism", but it's not - it is *concurrency*, which is a bit more wide concept. And yes, it means concurrency can be inplemented using parallelism, but it can also work without it (like most people do).

When programming things to run in parallel, we are generally thinking of explicitly creating several separate threads and giving each of them a target function. But it is not how it works in case of Golang - it operates in terms of *concurrency*, which means we don't really need to think if and how actual parallelization is happening under the hood.

That gives us a lot of power, but with great power comes great responsibility...

Chapter IV

Exercise 00: Sleepsort

So, let's write a toy algorithm as an example. It is pretty much useless for production, but it helps to grasp the concept.

You need to write a function called `sleepSort` that accepts an unsorted slice of integers and returns an integer channel, where these numbers will be written one by one in sorted order. To test it, in main goroutine you should read and print output values from a returned channel and gracefully stop the application in the end.

The idea of Sleepsort (what makes it a "toy") is that we're spawning a number of goroutines equal to the size of an input slice. Then, each goroutine sleeps for amount of seconds equal to the received number. After that it wakes up and this number to the output channel. It's easy to notice that this way numbers will be returned in a sorted order.

Chapter V

Exercise 01: Spider-Sense

You probably remember how Peter Parker realised he now has superpowers when he woke up in the morning. Well, let's write our own spider (or crawler) for parsing the web. You need to implement a function `crawlWeb` which will accept an input channel (for sending in URLs) and return another channel for crawling results (pointers to web page body as a string). Also, at any moment in time there shouldn't be more than 8 goroutines querying pages in parallel.

But we want to be quick and flexible, so another requirement is to be able to stop the crawling process at any time by pressing Ctrl+C (and your code should perform a graceful shutdown). For that you may add more input arguments to `crawlWeb` function, which should be either `context.Context` for cancellation or `done` channel. If not interrupted, the program should gracefully stop after all given URLs are processed.

Chapter VI

Exercise 02: Dr. Octopus

Okay, so now we have to slain the villain! The main problem with Dr.Octopus is that he has a lot of tech tentacles, and it's hard to keep track of them all. Let's tie them together!

Для этого упражнения вам нужно написать функцию, вызываемую `multiplex` с *переменным* числом аргументов (принимает переменное количество аргументов). Он должен принимать каналы (`chan interface{}`) в качестве аргументов и возвращать один канал того же типа. Идея состоит в том, чтобы перенаправлять любые входящие сообщения из этих входных каналов только в один выходной канал, эффективно реализуя шаблон «расширения».

В качестве доказательства работы вы должны написать тест на выборочных данных, который будет явно показывать, что все значения, отправленные случайным образом на любые входные каналы, поступают далее на тот же выходной канал.

И... вы только что победили злодея!