GO-02 06: Основы языка. defer - обработка выхода из функции

Описание:

В Go реализован механизм отложенного вызова функций. Отложенный вызов часто используется на практике для высвобождения занятых ресурсов — файловых дескрипторов, соединений с базой данной и т.д. Кроме этого, отложенный вызов удобно применять, если в вашей функции присутствует несколько операторов return, а по завершении функции нужно выполнять какие-то однотипные действия. Без применения отложенного вызова пришлось бы перед каждым оператором return прописывать одни и те же действия, что в случае расширения функционала доставляло бы неудобства, потому что надо не забыть добавить эти же инструкции перед каждым новым return. Отложенный вызов осуществляется при помощи оператора defer, который планирует выполнение функции перед завершением вызывающей функции.

```
func main() {
  path := "f.txt"
  getFile(path)
}

func getFile(path string) {
  f, _ := os.Open(path)
  defer f.Close()

  s := bufio.NewScanner(f)
  for s.Scan() {
    fmt.Println(s.Text())
  }
}
```

Выше показан пример работы с файлом. При помощи функции пакета os.Open() мы получаем доступ к файлу. В примере применяется пустой оператор, чтобы пропустить возвращаемую ошибку, так как об обработке ошибок поговорим в следующем разделе. Далее следует отложенный вызов f.Close(), который закрывает файловый дескриптор. Хорошая практика — описывать инструкции по освобождению ресурсов сразу, как только они были заняты.

Можно указывать несколько отложенных вызовов — порядок их выполнения будет обратным порядку объявления.

```
func main() {
  log()
}

func log() {
  defer fmt.Println(1)
  defer fmt.Println(2)
  defer fmt.Println(3)
}
```

Вывод программы в консоль будет следующим:

```
3
2
1
```

Интересно обстоят дела с параметрами, передаваемыми в defer. Они вычисляются в момент объявления отложенного вызова, а не в момент его выполнения. Рассмотрим это на примере.

```
func main() {
  i := 1
  defer prt(i)

  i++
  fmt.Printf("Main func execution: %d\n", i)
}

func prt(i int) {
  fmt.Printf("Deferred func call: %d\n", i)
}
```

В главной функции main() мы объявили переменную i и присвоили ей значение 1. Затем объявили отложенный вызов функции prt() и передали ей в качестве параметра переменную i. Выполнили инкремент переменной и вывели ее значение. Как вы уже знаете, defer планирует выполнение отложенного вызова в конце работы функции, где он объявлен, поэтому отложенный вызов будет выполнен после инкремента переменной. А теперь запустим код и посмотрим на результат его работы.

```
Main func execution: 2
Deferred func call: 1
```

В функцию prt() было передано значение переменной i до инкремента. То есть значение передаваемого в функцию параметра было вычислено на момент объявления отложенного вызова.
Что если требуется использовать значение переменной на момент выполнения отложенного вызова? Можно применить замыкание.

```
func main() {
  i := 1
  defer func() {
     prt(i)
  }()

  i++
  fmt.Printf("Main func execution: %d\n", i)
}

func prt(i int) {
  fmt.Printf("Deferred func call: %d\n", i)
}
```

В исправленном примере при выполнении вызова функции prt() значение переменной i будет взято из области видимости функции.

```
Main func execution: 2
Deferred func call: 2
```

Полезные ссылки:

- Как работает функция Defer в Golang
- Defer statement in Golang (part I)
- 5 Gotchas of Defer in Go — Part I
- 5 More Gotchas of Defer in Go — Part II
- https://golangbot.com/read-files/

Задание:

1. Ознакомьтесь с пакетом os и bufio.
2. Создайте в проекте module02 новую ветку 06_task.
3. Создайте новую директорию с файлом main.go.
4. Скачайте файл in.txt и положите его в директорию data рядом с файлом main.go.

5. В файле main.go напишите код, который считывает данные из файла in.txt и построчно записывает их в файл out.txt, нумеруя каждую строку. Если файла out.txt нет, то он должен создаваться.
6. С помощью отложенных вызовов закройте файловые дескрипторы. При закрытии файла out.txt программа должна вывести в консоль, сколько строк и байт было записано в файл.
7. Реализуйте функцию logTime(), которая не принимает на вход параметров, определяет и выводит на экран время выполнения вашей программы. Вывод времени должен происходить перед завершением работы функции main().
8. Зафиксируйте изменения в ветке и отправьте их в удаленный репозиторий проекта.
9. В качестве ответа пришлите ссылку на merge request в ветку master вашего проекта ветки 06_task.