Peter the Great
Saint-Petersburg Polytechnic University

Тема работы

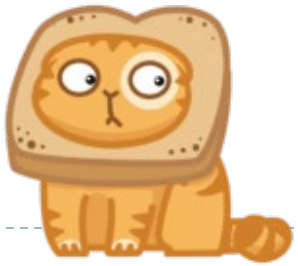
Обработка ошибок в Golang

Исполнители: учащиеся группы 5030102/00201
Руководитель: Иванов Денис Юрьевич

24.10.23

POLYTECH

Исключения, но не совсем (1/3)



- В Go нет конструкций try-catch
  - Есть встроенный тип error
    - Функции могут возвращать несколько значений, поэтому везде, где может быть ошибка, последним аргументом возвращаем error
    - Тип встроен в язык, не нужно ничего подключать
  - Обработка происходит путем проверки на равенство nil
    - Всегда можно работать с ошибками одинаково

Исключения, но не совсем (2/3)

- Если функция возвращает ошибку, это явно указано в сигнатуре
  - А в c++ нам надо было помнить, что необходимо оборачивать в try-except, а что нет
- Минусов почти нет, но...
  - Нужно всегда явно обрабатывать или игнорировать возвращаемую ошибку

Исключения, но не совсем (3/3)

```
5   func div(a, b int) (int, error) {
6       if b == 0 {
7           return 0, fmt.Errorf("делитель равен 0")
8       }
9       return a / b, nil
10  }
11
12  func main() {
13      d, err := div(10, 0)
14      if err != nil {
15          fmt.Println(err)
16      } else {
17          fmt.Printf("d = %d\n", d)
18      }
19  }
20
```

```
PS D:\NewProgrammingLanguages\New-programming-languages\
делитель равен 0
```

Panic(1/4)

- Сигнал о наличие ошибок, из-за которых продолжение работы невозможно.
  - В случае вызова panic среда выполнения Go просматривает стек, пытаясь найти для нее обработчик. Необработанные паники прекращают работу приложения
- Пример
  - Обращение к указателю со значением nil
    - panic: runtime error: invalid memory address or nil pointer dereference
  - Выход индекса за длину массива или среза
    - panic: runtime error: index out of range

Panic(2/4)

- Структура паники
  - Паника состоит из сообщения и трассировки стека. Трассировка стека разбита на отдельные блоки — один для каждой goroutine в вашей программе.

```
panic: runtime error: index out of range [3] with length 3

goroutine 1 [running]:
main.outOfRange()
        /Users/tronyagina/panic/main.go:28 +0x15
main.main()
        /Users/tronyagina/panic/main.go:8 +0xf
exit status 2
```

```
panic: runtime error: invalid memory address or nil pointer dereference
[signal SIGSEGV: segmentation violation code=0x1 addr=0x0 pc=0x1086ed6]

goroutine 1 [running]:
main.nilPointer()
        /Users/tronyagina/panic/main.go:18 +0x16
main.main()
        /Users/tronyagina/panic/main.go:7 +0xf
exit status 2
```

Panic(3/4)

- Возможность использовать внутреннюю функцию panic

```go
func main() {
    foo()
}

func foo() {
    panic("error has occurred")
}
```

```
panic: error has occurred

goroutine 1 [running]:
main.panicFunction(...)
        /Users/tronyagina/panic/main.go:10
main.main()
        /Users/tronyagina/panic/main.go:6 +0x25
exit status 2
```

Panic(4/4)

- Отсроченные функции
  - Вызываются даже в случае при наличие паники

```go
func main() {
    defer func() {
        fmt.Println("hello from the deferred function!")
    }()

    panic("oh no!")
}
```

```
hello from the deferred function!
panic: oh no!

goroutine 1 [running]:
main.main()
        /Users/tronyagina/panic/main.go:13 +0x3e
exit status 2
```

Recover(1/2)

- Единый механизм восстановления ошибок
  - Позволяет перехватить панику по ходу выполнения програмы и не допустить прекращение работы программы.
  - Функция recover опирается на значение ошибки при определении того, была ли сгенерирована паника или нет.

Recover(2/2)

- Пример

```
func main() {
    fmt.Println(divide(1, 0))
    fmt.Println("we survived dividing by zero!")

}

func divide(a, b int) int {
    return a / b
}
```

```
panic: runtime error: integer divide by zero

goroutine 1 [running]:
main.divide(...)
        /Users/tronyagina/panic/main.go:47
main.main()
        /Users/tronyagina/panic/main.go:41 +0xa
exit status 2
```

```
func main() {
    divideByZero()
    fmt.Println("we survived dividing by zero!")

}

func divideByZero() {
    defer func() {
        if err := recover(); err != nil {
            log.Println("panic occurred:", err)
        }
    }()
    fmt.Println(divide(1, 0))
}
```

```
2023/10/22 17:07:35 panic occurred: runtime error: integer divide by zero
we survived dividing by zero!
```

Логирование (1/2)

- Записи об ошибках можно вести в системном логе
  - По умолчанию вывод в консоль
  - Нет уровней логирования в отличие от джавы и других
    - Забудьте про WARNING, INFO, ERROR, DEBUG без использования дополнительных пакетов
    - Обычно их добавляют в виде префикса
- Пакет log
  - Fatal() <=> Print(); os.Exit(1)
  - Output() - вывод в лог
  - Panic() <=> Print(); panic()

Логирование (2/2)

- Примеры

```
15  func getPanic() {
16      log.Panic("U MENYA PANIKA")
17  }
```

```
2023/10/22 16:54:37 U MENYA PANIKA
panic: U MENYA PANIKA

goroutine 1 [running]:
log.Panic({0xc00009df20?, 0xc000070010?, 0x1099d7c?})
        C:/Program Files/Go/src/log/log.go:432 +0x5a
main.getPanic(...)
        D:/NewProgrammingLanguages/New-programming-languages/05_error_hangling/main.go:16
main.main()
        D:/NewProgrammingLanguages/New-programming-languages/05_error_hangling/main.go:25 +0x96
exit status 2
```

```
13      file, err := os.Open(filepath)
14      if err != nil {
15          log.Fatal(err)
16      }
17      defer file.Close()
```