Лабораторная работа No 1

Пишикина Мария, ИУ9-31Б

10 февраля 2023

1 Код решения

Файл proto.go

```
package proto

import "encoding/json"

// Request -- запрос клиента к серверу.

type Request struct {
        Command string `json:"command"`

        Data *json.RawMessage `json:"data"`
}

// Response -- ответ сервера клиенту.

type Response struct {
        Status string `json:"status"`

        Data *json.RawMessage `json:"data"`
}

// DivisibilityRequest - запрос клиента к серверу для проверки делимости числа на
type DivisibilityRequest struct {
        // Число в виде последовательности десятичных цифр.
```

```
        Number string `json:"number"`
}

// DivisibilityResponse - ответ сервера клиенту после проверки делимости числа на
type DivisibilityResponse struct {
        // Результат проверки: true - число делится на 3, false - число не делится
        Divisible bool `json:"divisible"`
}
```

Файл server.go

```
package main

import (
        "encoding/json"
        "flag"
        "fmt"
        "math/big"
        "net"

        //"proto"

        log "github.com/mgutz/logxi/v1"
)

type Request struct {
        Command string `json:"command"`

        Data *json.RawMessage `json:"data"`
}

// Response -- ответ сервера клиенту.

type Response struct {
        Status string `json:"status"`
```

```
        Data *json.RawMessage `json:"data"`
}

// DivisibilityRequest - запрос клиента к серверу для проверки делимости числа на
type DivisibilityRequest struct {
        // Число в виде последовательности десятичных цифр.
        Number string `json:"number"`
}

// DivisibilityResponse - ответ сервера клиенту после проверки делимости числа на
type DivisibilityResponse struct {
        // Результат проверки: true - число делится на 3, false - число не делится
        Divisible bool `json:"divisible"`
}

// Client - состояние клиента.
type Client struct {
        logger log.Logger     // Объект для печати логов
        conn   *net.TCPConn   // Объект TCP-соединения
        enc    *json.Encoder // Объект для кодирования и отправки сообщений
        sum    *big.Rat      // Текущая сумма полученных от клиента дробей
        count  int64         // Количество полученных от клиента дробей
}

// NewClient - конструктор клиента, принимает в качестве параметра
// объект TCP-соединения.
func NewClient(conn *net.TCPConn) *Client {
        return &Client{
                logger: log.New(fmt.Sprintf("client %s", conn.RemoteAddr().String(
                conn:   conn,
                enc:    json.NewEncoder(conn),
                sum:    big.NewRat(0, 1),
                count:  0,
        }
}

// serve - метод, в котором реализован цикл взаимодействия с клиентом.
// Подразумевается, что метод serve будет вызываться в отдельной go-программе.
```

```
func (client *Client) serve() {
	defer client.conn.Close()
	decoder := json.NewDecoder(client.conn)
	for {
		var req Request
		if err := decoder.Decode(&req); err != nil {
			client.logger.Error("cannot decode message", "reason", err
			break
		} else {
			client.logger.Info("received command", "command", req.Comm
			if client.handleRequest(&req) {
				client.logger.Info("shutting down connection")
				break
			}
		}
	}
}

func isDivisibleBy3(number string) bool {
	n := 0
	for _, digit := range number {
		n += int(digit - '0')
	}
	return n%3 == 0
}

// handleRequest - метод обработки запроса от клиента. Он возвращает true,
// если клиент передал команду "quit" и хочет завершить общение.
func (client *Client) handleRequest(req *Request) bool {
	switch req.Command {
	case "quit":
		client.respond("ok", nil)
		return true

	case "check":
		if req.Data == nil {
			client.logger.Error("check failed", "reason", "data field
			client.respond("failed", "data field is absent")
```

```
                } else {
                        var divisibilityReq DivisibilityRequest
                        if err := json.Unmarshal(*req.Data, &divisibilityReq); err
                                client.logger.Error("check failed", "reason", "mal
                                client.respond("failed", "malformed data field")
                        } else {
                                if isDivisibleBy3(divisibilityReq.Number) {
                                        client.logger.Info("number is divisible by
                                        client.respond("divisibility", &Divisibili
                                } else {
                                        client.logger.Info("number is not divisibl
                                        client.respond("divisibility", &Divisibili
                                }
                        }
                }
        default:
                client.logger.Error("unknown command")
                client.respond("failed", "unknown command")
        }
        return false
}

// respond - вспомогательный метод для передачи ответа с указанным статусом
// и данными. Данные могут быть пустыми (data == nil).
func (client *Client) respond(status string, data interface{}) {
        var raw json.RawMessage
        raw, _ = json.Marshal(data)
        client.enc.Encode(&Response{status, &raw})
}

func main() {
        // Работа с командной строкой, в которой может указываться необязательный
        var addrStr string
        flag.StringVar(&addrStr, "addr", "127.0.0.1:6000", "specify ip address and
        flag.Parse()

        // Разбор адреса, строковое представление которого находится в переменной
        if addr, err := net.ResolveTCPAddr("tcp", addrStr); err != nil {
```

```
                log.Error("address resolution failed", "address", addrStr)
        } else {
                log.Info("resolved TCP address", "address", addr.String())

                // Инициация слушания сети на заданном адресе.
                if listener, err := net.ListenTCP("tcp", addr); err != nil {
                        log.Error("listening failed", "reason", err)
                } else {
                        // Цикл приёма входящих соединений.
                        for {
                                if conn, err := listener.AcceptTCP(); err != nil {
                                        log.Error("cannot accept connection", "rea
                                } else {
                                        log.Info("accepted connection", "address",

                                        // Запуск go-программы для обслуживания кл
                                        go NewClient(conn).serve()
                                }
                        }
                }
        }
}
```

Файл client.go

```
package main

import (
        "encoding/json"
        "flag"
        "fmt"
        "net"

        //"proto"

        "github.com/skorobogatov/input"
```

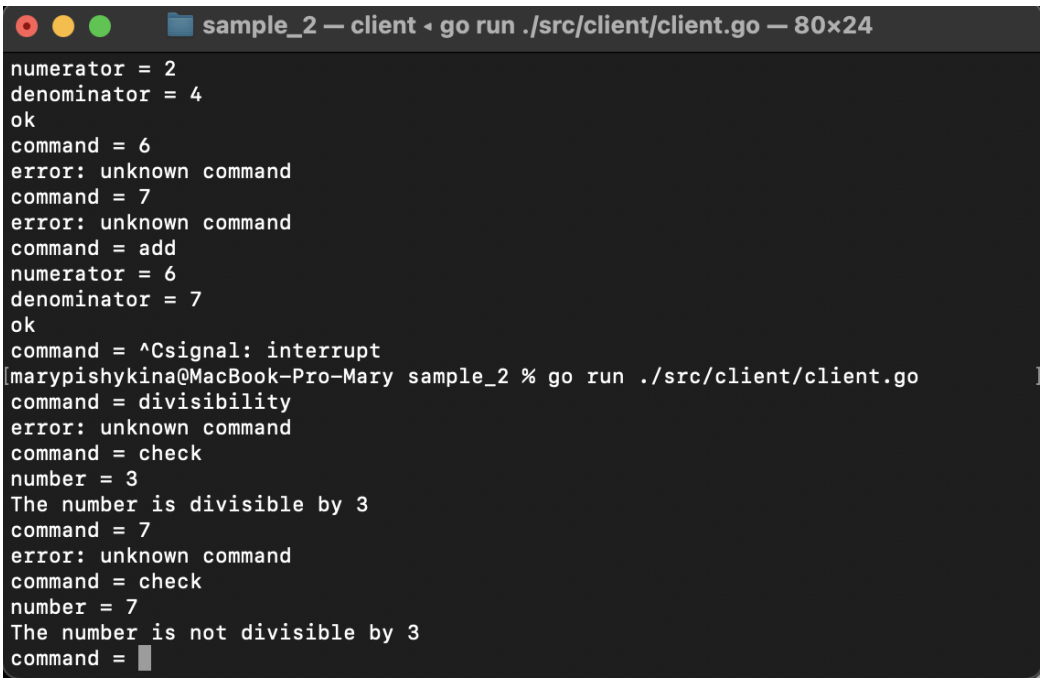
```
)

type Request struct {
        Command string `json:"command"`

        Data *json.RawMessage `json:"data"`
}

// Response -- ответ сервера клиенту.

type Response struct {
        Status string `json:"status"`

        Data *json.RawMessage `json:"data"`
}

// DivisibilityRequest - запрос клиента к серверу для проверки делимости числа на
type DivisibilityRequest struct {
        // Число в виде последовательности десятичных цифр.
        Number string `json:"number"`
}

// DivisibilityResponse - ответ сервера клиенту после проверки делимости числа на
type DivisibilityResponse struct {
        // Результат проверки: true - число делится на 3, false - число не делится
        Divisible bool `json:"divisible"`
}

// interact - функция, содержащая цикл взаимодействия с сервером.
func interact(conn *net.TCPConn) {
        defer conn.Close()
        encoder, decoder := json.NewEncoder(conn), json.NewDecoder(conn)
        for {
                // Чтение команды из стандартного потока ввода
                fmt.Printf("command = ")
                command := input.Gets()

                // Отправка запроса.
```

```
switch command {
case "quit":
        send_request(encoder, "quit", nil)
        return
case "avg":
        send_request(encoder, "avg", nil)
case "check":
        var number string
        fmt.Printf("number = ")
        number = input.Gets()
        send_request(encoder, "check", &DivisibilityRequest{
                Number: number,
        })
default:
        fmt.Printf("error: unknown command\n")
        continue
}

// Получение ответа.
var resp Response
if err := decoder.Decode(&resp); err != nil {
        fmt.Printf("error: %v\n", err)
        break
}

// Вывод ответа в стандартный поток вывода.
switch resp.Status {
case "ok":
        fmt.Printf("ok\n")
case "failed":
        if resp.Data == nil {
                fmt.Printf("error: data field is absent in respons
        } else {
                var errorMsg string
                if err := json.Unmarshal(*resp.Data, &errorMsg); e
                        fmt.Printf("error: malformed data field ir
                } else {
                        fmt.Printf("failed: %s\n", errorMsg)
```

```
                                }
                        }
                case "divisibility":
                        if resp.Data == nil {
                                fmt.Printf("error: data field is absent in respons
                        } else {
                                var divisibilityResp DivisibilityResponse
                                if err := json.Unmarshal(*resp.Data, &divisibilit
                                        fmt.Printf("error: malformed data field in
                                } else {
                                        if divisibilityResp.Divisible {
                                                fmt.Printf("The number is divisibl
                                        } else {
                                                fmt.Printf("The number is not divi
                                        }
                                }
                        }
                default:
                        fmt.Printf("error: server reports unknown status %q\n", re
                }
        }
}

// send_request - вспомогательная функция для передачи запроса с указанной команд
// и данными. Данные могут быть пустыми (data == nil).
func send_request(encoder *json.Encoder, command string, data interface{}) {
        var raw json.RawMessage
        raw, _ = json.Marshal(data)
        encoder.Encode(&Request{command, &raw})
}

func main() {
        // Работа с командной строкой, в которой может указываться необязательный
        var addrStr string
        flag.StringVar(&addrStr, "addr", "127.0.0.1:6000", "specify ip address and
        flag.Parse()

        // Разбор адреса, установка соединения с сервером и
```

```
        // запуск цикла взаимодействия с сервером.
        if addr, err := net.ResolveTCPAddr("tcp", addrStr); err != nil {
                fmt.Printf("error: %v\n", err)
        } else if conn, err := net.DialTCP("tcp", nil, addr); err != nil {
                fmt.Printf("error: %v\n", err)
        } else {
                interact(conn)
        }
}
```

2 Пример работы программы



1