WB Tech: level # 2 (Golang)

Как делать задания

Никаких устных решений — только код. Одно решение — один файл с хорошо откомментированным кодом. Каждое решение или невозможность решения надо объяснить.

Разрешается и приветствуется использование любых справочных ресурсов, привлечение сторонних экспертов и т.д. и т.п.

Основной критерий оценки — четкое понимание «как это работает». Некоторые задачи можно решить несколькими способами, в этом случае требуется привести максимально возможное количество вариантов и обосновать наиболее оптимальный из них, если таковой имеется.

Можно задавать вопросы, как по условию задач, так и об их решении.

Задания

Паттерны проектирования

1. Реализовать паттерн «фасад». Объяснить применимость паттерна, его плюсы и минусы, а также реальные примеры использования данного примера на практике.
2. Реализовать паттерн «строитель». Объяснить применимость паттерна, его плюсы и минусы, а также реальные примеры использования данного примера на практике.
3. Реализовать паттерн «посетитель». Объяснить применимость паттерна, его плюсы и минусы, а также реальные примеры использования данного примера на практике.
4. Реализовать паттерн «комманда». Объяснить применимость паттерна, его плюсы и минусы, а также реальные примеры использования данного примера на практике.
5. Реализовать паттерн «цепочка вызовов». Объяснить применимость паттерна, его плюсы и минусы, а также реальные примеры использования данного примера на практике.
6. Реализовать паттерн «фабричный метод». Объяснить применимость паттерна, его плюсы и минусы, а также реальные примеры использования данного примера на практике.
7. Реализовать паттерн «стратегия». Объяснить применимость паттерна, его плюсы и минусы, а также реальные примеры использования данного примера на практике.

8. Реализовать паттерн «состояние». Объяснить применимость паттерна, его плюсы и минусы, а также реальные примеры использования данного примера на практике.

Чтение и понимание кода

1. Что выведет программа? Объяснить вывод программы.

```
package main

import (
    "fmt"
)

func main() {
    a := [5]int{76, 77, 78, 79, 80}
    var b []int = a[1:4]
    fmt.Println(b)
}
```

2. Что выведет программа? Объяснить вывод программы. Объяснить как работают defer’ы и их порядок вызовов.

```
package main

import (
    "fmt"
)


func test() (x int) {
    defer func() {
        x++
    }()
    x = 1
    return
}


func anotherTest() int {
    var x int
    defer func() {
        x++
```

```
    }()
    x = 1
    return x
}


func main() {
    fmt.Println(test())
    fmt.Println(anotherTest())
}
```

3. Что выведет программа? Объяснить вывод программы. Объяснить внутреннее устройство интерфейсов и их отличие от пустых интерфейсов.

```
package main

import (
    "fmt"
    "os"
)

func Foo() error {
    var err *os.PathError = nil
    return err
}

func main() {
    err := Foo()
    fmt.Println(err)
    fmt.Println(err == nil)
}
```

4. Что выведет программа? Объяснить вывод программы.

```
package main

func main() {
    ch := make(chan int)
    go func() {
        for i := 0; i < 10; i++ {
            ch <- i
        }
    }()
```

```
    for n := range ch {
        println(n)
    }
}
```

5. Что выведет программа? Объяснить вывод программы.

```
package main

type customError struct {
     msg string
}

func (e *customError) Error() string {
    return e.msg
}

func test() *customError {
     {
         // do something
     }
     return nil
}

func main() {
    var err error
    err = test()
    if err != nil {
        println("error")
        return
    }
    println("ok")
}
```

6. Что выведет программа? Объяснить вывод программы. Рассказать про внутреннее устройство слайсов и что происходит при передачи их в качестве аргументов функции.

```
package main

import (
  "fmt"
)
```

```
func main() {
  var s = []string{"1", "2", "3"}
  modifySlice(s)
  fmt.Println(s)
}

func modifySlice(i []string) {
  i[0] = "3"
  i = append(i, "4")
  i[1] = "5"
  i = append(i, "6")
}
```

7. Что выведет программа? Объяснить вывод программы.

```
package main

import (
    "fmt"
    "math/rand"
    "time"
)

func asChan(vs ...int) <-chan int {
   c := make(chan int)

   go func() {
       for _, v := range vs {
           c <- v
           time.Sleep(time.Duration(rand.Intn(1000)) *
time.Millisecond)
       }

       close(c)
  }()
  return c
}

func merge(a, b <-chan int) <-chan int {
   c := make(chan int)
   go func() {
       for {
           select {
```

```
                case v := <-a:
                    c <- v
                case v := <-b:
                    c <- v
            }
        }
    }()
 return c
}

func main() {

    a := asChan(1, 3, 5, 7)
    b := asChan(2, 4 ,6, 8)
    c := merge(a, b )
    for v := range c {
        fmt.Println(v)
    }
}
```

Задачи на разработку

1. Базовая задача

   Создать программу печатающую точное время с использованием NTP библиотеки.Инициализировать как go module. Использовать библиотеку github.com/beevik/ntp. Написать программу печатающую текущее время / точное время с использованием этой библиотеки.

   1. Программа должна быть оформлена с использованием как go module.
   2. Программа должна корректно обрабатывать ошибки библиотеки: распечатывать их в STDERR и возвращать ненулевой код выхода в OS.
   3. Программа должна проходить проверки go vet и golint.

2. Задача на распаковку.

   Создать Go функцию, осуществляющую примитивную распаковку строки, содержащую повторяющиеся символы / руны, например:

   - "a4bc2d5e" => "aaaabccddddde"
   - "abcd" => "abcd"

- "45" => "" (некорректная строка)
- "" => ""

Дополнительное задание: поддержка escape - последовательностей

- `qwe\4\5` => `qwe45` (*)
- `qwe\45` => `qwe44444` (*)
- `qwe\\5` => `qwe\\\\\` (*)

В случае если была передана некорректная строка функция должна возвращать ошибку. Написать unit-тесты.

Функция должна проходить все тесты. Код должен проходить проверки go vet и golint.

3. Утилита sort.

Отсортировать строки (man sort)
Основное

Поддержать ключи

-k — указание колонки для сортировки
-n — сортировать по числовому значению
-r — сортировать в обратном порядке
-u — не выводить повторяющиеся строки

Дополнительное

Поддержать ключи

-M — сортировать по названию месяца
-b — игнорировать хвостовые пробелы
-c — проверять отсортированы ли данные
-h — сортировать по числовому значению с учётом суффиксов

Программа должна проходить все тесты. Код должен проходить проверки go vet и golint.

4. Поиск анаграмм по словарю.

Напишите функцию поиска всех множеств анаграмм по словарю.
Например:
'пятак', 'пятка' и 'тяпка' - принадлежат одному множеству,
'листок', 'слиток' и 'столик' - другому.
Входные данные для функции: ссылка на массив - каждый элемент которого - слово на русском языке в кодировке utf8.
Выходные данные: Ссылка на мапу множеств анаграмм.

Ключ - первое встретившееся в словаре слово из множества
Значение - ссылка на массив, каждый элемент которого, слово из множества. Массив должен быть отсортирован по возрастанию.
Множества из одного элемента не должны попасть в результат.
Все слова должны быть приведены к нижнему регистру.
В результате каждое слово должно встречаться только один раз.

Программа должна проходить все тесты. Код должен проходить проверки go vet и golint.

5. Утилита grep.

Реализовать утилиту фильтрации (man grep)

Поддержать флаги:
-A - "after" печатать +N строк после совпадения
-B - "before" печатать +N строк до совпадения
-C - "context" (A+B) печатать ±N строк вокруг совпадения
-c - "count" (количество строк)
-i - "ignore-case" (игнорировать регистр)
-v - "invert" (вместо совпадения, исключать)
-F - "fixed", точное совпадение со строкой, не паттерн
-n - "line num", печатать номер строки

Программа должна проходить все тесты. Код должен проходить проверки go vet и golint.

6. Утилита cut.

Принимает STDIN, разбивает по разделителю (TAB) на колонки, выводит запрошенные

Поддержать флаги:
-f - "fields" - выбрать поля (колонки)
-d - "delimiter" - использовать другой разделитель
-s - "separated" - только строки с разделителем

Программа должна проходить все тесты. Код должен проходить проверки go vet и golint.

7. Or channel.

Реализовать функцию, которая будет объединять один или более *done* каналов в *single* канал если один из его составляющих каналов закроется.

Одним из вариантов было бы очевидно написать выражение при помощи *select*, которое бы реализовывало эту связь, однако иногда

неизвестно общее число *done* каналов, с которыми вы работаете в рантайме. В этом случае удобнее использовать вызов единственной функции, которая, приняв на вход один или более *or* каналов, реализовывала весь функционал.

Определение функции:

```
var or func(channels ...<- chan interface{}) <- chan interface{}
```

Пример использования функции:

```
sig := func(after time.Duration) <- chan interface{} {
	c := make(chan interface{})
	go func() {
		defer close(c)
		time.Sleep(after)
	}()
	return c
}

start := time.Now()
<-or (
	sig(2*time.Hour),
	sig(5*time.Minute),
	sig(1*time.Second),
	sig(1*time.Hour),
	sig(1*time.Minute),
)

fmt.Printf("fone after %v", time.Since(start))
```

8. Взаимодействие с ОС.

Необходимо реализовать собственный шелл
встроенные команды: cd/pwd/echo/kill/ps
поддержать fork/exec команды
конвеер на пайпах
Реализовать утилиту netcat (nc) клиент
принимать данные из stdin и отправлять в соединение (tcp/udp)
Программа должна проходить все тесты. Код должен проходить проверки go vet и golint.

9. Утилита wget.

Реализовать утилиту wget с возможностью скачивать сайты целиком

Программа должна проходить все тесты. Код должен проходить проверки go vet и golint.

10. Утилита telnet.

Реализовать примитивный telnet клиент:
Примеры вызовов:
go-telnet --timeout=10s host port go-telnet mysite.ru 8080
go-telnet --timeout=3s 1.1.1.1 123

Программа должна подключаться к указанному хосту (ip или доменное имя) и порту по протоколу TCP После подключения STDIN программы должен записываться в сокет, а данные полученные и сокета должны выводиться в STDOUT

Опционально в программу можно передать таймаут на подключение к серверу (через аргумент --timeout, по умолчанию 10s).

При нажатии Ctrl+D программа должна закрывать сокет и завершаться. Если сокет закрывается со стороны сервера, программа должна также завершаться. При подключении к несуществующему сервер, программа должна завершаться через timeout.

11. HTTP server

Реализовать HTTP сервер для работы с календарем. В рамках задания необходимо работать строго со стандартной HTTP библиотекой.
В рамках задания необходимо:

1. Реализовать вспомогательные функции для сериализации объектов доменной области в JSON.
2. Реализовать вспомогательные функции для парсинга и валидации параметров методов /create_event и /update_event.
3. Реализовать HTTP обработчики для каждого из методов API, используя вспомогательные функции и объекты доменной области.
4. Реализовать middleware для логирования запросов

Методы API: POST /create_event POST /update_event POST /delete_event GET /events_for_day GET /events_for_week GET /events_for_month

Параметры передаются в виде www-url-form-encoded (т.е. обычные user_id=3&date=2019-09-09). В GET методах параметры передаются через queryString, в POST через тело запроса.

В результате каждого запроса должен возвращаться JSON документ содержащий либо {"result": "..."} в случае успешного выполнения метода, либо {"error": "..."} в случае ошибки бизнес-логики.

В рамках задачи необходимо:

1. Реализовать все методы.
2. Бизнес логика НЕ должна зависеть от кода HTTP сервера.
3. В случае ошибки бизнес-логики сервер должен возвращать HTTP 503. В случае ошибки входных данных (невалидный int например) сервер должен возвращать HTTP 400. В случае остальных ошибок сервер должен возвращать HTTP 500. Web-сервер должен запускаться на порту указанном в конфиге и выводить в лог каждый обработанный запрос.
4. Код должен проходить проверки go vet и golint.