Примеры программ на Golang

28 октября 2016

Владимир Ананьев
АО "Программный Регион"

Домашнее задание: track-server

- Не так сложно, как могло показаться
- Используем документацию golang.org/pkg/net/http/ (https://golang.org/pkg/net/http/)
- Сходу может быть сложновато, поэтому гуглим **golang http server** :)
- Также поступаем с golang.org/pkg/encoding/base64/ (https://golang.org/pkg/encoding/base64/)
- Также поступаем с **golang redirect**

```
http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
    str := r.URL.EscapedPath()[1:] // Все элементы массива, начиная с индекса 1
    url, err := base64.StdEncoding.DecodeString(str)
    if err != nil {
        .. // Обработка ошибки
    }

    http.Redirect(w, r, "http://"+string(url), http.StatusFound) // 302, трехсотые коды ответа
})
http.ListenAndServe(":8085", nil))
```

Возвращаемые значения

- gobyexample.com/ (https://gobyexample.com/)

```
package main

import "fmt"

func vals() (int, int) {
    return 3, 7
}

func main() {
    a, b := vals()
    fmt.Println(a) // 3
    fmt.Println(b) // 7

    _, c := vals()
    fmt.Println(c) // 7
}
```

Функции с переменным числом параметров

```
package main

import "fmt"

func sum(nums ...int) {
    fmt.Print(nums, " ")
    total := 0
    for _, num := range nums {
        total += num
    }
    fmt.Println(total)
}

func main() {
    sum(1, 2)    // 3
    sum(1, 2, 3) // 6

    nums := []int{1, 2, 3, 4}
    sum(nums...) // 10
}
```

Замыкания (анонимные функции)

```
package main

import "fmt"

func intSeq() func() int {
    i := 0
    return func() int {
        i += 1
        return i
    }
}

func main() {
    nextInt := intSeq()

    fmt.Println(nextInt()) // 1
    fmt.Println(nextInt()) // 2
    fmt.Println(nextInt()) // 3

    newInts := intSeq()
    fmt.Println(newInts()) // 1
}
```

Указатели

```
func zeroval(ival int) {
    ival = 0
}

func zeroptr(iptr *int) {
    *iptr = 0
}

func main() {
    i := 1
    fmt.Println("initial:", i) // 1

    zeroval(i)
    fmt.Println("zeroval:", i) // 1

    zeroptr(&i)
    fmt.Println("zeroptr:", i) // 0

    fmt.Println("pointer:", &i) // 0x1040e0f8
}
```

Goroutines

```
func f(from string) {
    for i := 0; i < 3; i++ {
        fmt.Println(from, ":", i)
    }
}

func main() {
    f("direct")
    go f("goroutine")

    go func(msg string) {
        fmt.Println(msg)
    }("going")

    var input string
    fmt.Scanln(&input) // Ожидание ввода
    fmt.Println("done")
}
```

Каналы

```
package main

import "fmt"

func main() {
    messages := make(chan string)

    go func() { messages <- "ping" }()

    msg := <-messages
    fmt.Println(msg)
}
```

Буферизованные каналы

```
package main

import "fmt"

func main() {
    messages := make(chan string, 2)

    messages <- "buffered"
    messages <- "channel"

    fmt.Println(<-messages)
    fmt.Println(<-messages)
}
```

Синхронизация с помощью каналов

```
package main

import "fmt"
import "time"

func worker(done chan bool) {
    fmt.Print("working...")
    time.Sleep(time.Second)
    fmt.Println("done")

    done <- true
}

func main() {
    done := make(chan bool, 1)
    go worker(done)

    <-done
}
```

Направление каналов

```
package main

import "fmt"

func ping(pings chan<- string, msg string) {
    pings <- msg
}

func pong(pings <-chan string, pongs chan<- string) {
    msg := <-pings
    pongs <- msg
}

func main() {
    pings := make(chan string, 1)
    pongs := make(chan string, 1)
    ping(pings, "passed message")
    pong(pings, pongs)
    fmt.Println(<-pongs)
}
```

Select по каналам

```
func main() {
    c1 := make(chan string)
    c2 := make(chan string)

    go func() {
        time.Sleep(time.Second * 1)
        c1 <- "one"
    }()
    go func() {
        time.Sleep(time.Second * 2)
        c2 <- "two"
    }()

    for i := 0; i < 2; i++ {
        select { // default - ?
        case msg1 := <-c1:
            fmt.Println("received", msg1)
        case msg2 := <-c2:
            fmt.Println("received", msg2)
        }
    }
}
```

Range по каналам

```
package main

import "fmt"

func main() {
    queue := make(chan string, 2)
    queue <- "one"
    queue <- "two"
    close(queue)

    for elem := range queue { // Широко используется в клиентских библиотеках
        fmt.Println(elem)
    }
}
```

Panic

```
package main

import "os"

func main() {

    panic("a problem")

    _, err := os.Create("/tmp/file")
    if err != nil {
        panic(err)
    }
}
```

- Панику можно отловить (recover github.com/golang/go/wiki/PanicAndRecover (https://github.com/golang/go/wiki/PanicAndRecover))

Defer

```
func main() {
    f := createFile("/tmp/defer.txt")
    defer closeFile(f)
    writeFile(f)
}

func createFile(p string) *os.File {
    fmt.Println("creating")
    f, err := os.Create(p)
    if err != nil {
        panic(err)
    }
    return f
}

func writeFile(f *os.File) {
    fmt.Println("writing")
    fmt.Fprintln(f, "data")
}

func closeFile(f *os.File) {
    fmt.Println("closing")
    f.Close()
}
```

Thank you

28 октября 2016

Владимир Ананьев
АО "Программный Регион"
vladimir.ananyev@regium.com (mailto:vladimir.ananyev@regium.com)