

Peter the Great
Saint-Petersburg Polytechnic University



Стандартная библиотека ввода/вывода. Управление сложностью кодаа.



Исполнитель: Стивен Спилберг

18.12.1946

Пакет <fmt>

```
fmt.Print("Hello", "World")
```

```
// Вывод: Hello World
```

```
name := "John"
```

```
fmt.Printf("Привет, %s!\n", name)
```

```
// Вывод: Привет, John!
```

```
fmt.Println("Это новая строка")
```

```
// Вывод: Это новая строка
```

Пакет <fmt>

```
var age int
fmt.Print("Введите возраст: ")
fmt.Scan(&age)

var height float64
fmt.Print("Введите ваш рост в метрах: ")
fmt.Scanf("%f", &height)
```

io

`io.Reader` интерфейс:

- Определяет метод `Read(p []byte) (n int, err error)`, `Read` читает до `len(buf)` байтов в `buf` и возвращает количество прочитанных байтов - он возвращает ошибку `io.EOF`, когда заканчивается поток.

```
type Reader interface {  
    Read(buf []byte) (n int, err error)
```

Reader

```
r := strings.NewReader("abcde")

buf := make([]byte, 4)
for {
    n, err := r.Read(buf)
    fmt.Println(n, err, buf[:n])
    if err == io.EOF {
        break
    }
}
```

```
4 <nil> [97 98 99 100]
1 <nil> [101]
0 EOF []
```

FullReader

```
r := strings.NewReader("abcde")

buf := make([]byte, 4)
if _, err := io.ReadFull(r, buf); err != nil {
    log.Fatal(err)
}
fmt.Println(buf)

if _, err := io.ReadFullSl(r, buf); err != nil {
    fmt.Println(err)
}
```

```
[97 98 99 100]
unexpected EOF
```

ReadAll

```
r := strings.NewReader("abcde")

buf, err := ioutil.ReadAll(r)
if err != nil {
    log.Fatal(err)
}
fmt.Println(buf)
```

[97 98 99 100 101]

io

`io.Writer` интерфейс:

- Определяет метод `Write(p []byte) (n int, err error)`, `Write` записывает до `len(p)` байтов из `p` в подлежащий поток данных - он возвращает количество записанных байтов и обнаруженную ошибку, которая привела к преждевременной остановке записи.

```
type Writer interface {  
    Write(p []byte) (n int, err error)
```

```
}
```


Writer and bufio

```
f, err := os.OpenFile("/tmp/123.txt", os.O_WRONLY, 0600)
n, err := f.Write([]byte("writing some data into a file"))
if err != nil {
    panic(err)
}
fmt.Println("wrote %d bytes", n)
```

```
scanner := bufio.NewScanner(file)
// optionally, resize scanner's capacity for lines over 64K
for scanner.Scan() {
    fmt.Println(scanner.Text())
}
```

Управление сложностью кода

- Основные структурные единицы
 - Пакеты
 - Модули
 - Содержат в себе много пакетов
 - Каждый модуль имеет свой `go.mod`

Структура проекта из одного модуля

project

```
├─ package1
│   └─ file1.go
├─ package2
│   ├── file2_1.go
│   └─ file2_2.go
├─ package3
│   ├── file3_1.go
│   └─ file3_2.go
├─ cmd <- main directory
│   └─ main.go
├─ internal
│   └─ internalcode.go
└─ go.mod
```

Особенности пакетов


- Контроль доступа к функциям/полям структур и т.д.
 - Приватные единицы доступны только внутри пакета
 - Названия начинаются с маленькой буквы
 - Публичные единицы доступны в других пакетах
 - Названия начинаются с большой буквы
- Все файлы одного пакета видны друг другу
- Разделение кода на логические, но взаимосвязанные блоки
- Не требуют модификации файла `go.mod`

Пример структуры проекта

myproject

```
|— address <- package
|   └─ Address.go
|— person <- package
|   └─ Person.go
|       └─ User.go
|— myerrors <- package
|   └─ AgeError.go
|       └─ PasswordError.go
|           └─ LoginError.go
|— cmd <- main directory
|   └─ main.go
└─ go.mod
```

Пример (1/6)

address >  Address.go > ...

```
1 package address
2
3 type Address struct {
4     country string
5     city    string
6     street  string
7     building string
8 }
```

```
62 func (a *Address) SetAddress(country string, city string, street string, building string) {
63     // Аналогичные проверки на существование адреса
64     // ...
65     a.country = country
66     a.city = city
67     a.street = street
68     a.building = building
69 }
```

Пример (2/6)

```
person > go Person.go > ...
1  package person
2
3  import (
4      "main/address"
5      "main/myerrors"
6  )
7
8  type Person struct {
9      FirstName string
10     LastName  string
11     age       int
12     Address   address.Address
13 }
```


```
26 // func (p *Person) SetAge(age int) error {
27     if age < 0 {
28         return &myerrors.AgeError{Message: "Возраст не может быть отрицательным"}
29     }
30     p.age = age
31     return nil
32 }
```


Пример (3/6)


```
person > go User.go > ...
1  package person
2
3  import (
4      "main/address"
5      "main/myerrors"
6  )
7
8  type User struct {
9      Person
10     Email    string
11     password string
12 }
```

```
24 func (u *User) ChangePassword(oldPassword string, newPassword string) error {
25     if oldPassword != u.password {
26         return &myerrors.IncorrectPasswordError{Message: "Неверный старый пароль"}
27     }
28     if len(newPassword) < 8 {
29         return &myerrors.InvalidPasswordError{Message: "Пароль должен удовлетворять условиям безопасности"}
30     }
31     u.password = newPassword
32     return nil
33 }
```


Пример (4/6)

myerrors >  AgeError.go > ...

```
1 package myerrors
2
3 type AgeError struct {
4     Message string
5 }
6
7 func (e *AgeError) Error() string {
8     return e.Message
9 }
```

myerrors >  LoginError.go > ...

```
1 package myerrors
2
3 type InvalidLoginDataError struct {
4     Message string
5 }
6
7 func (e *InvalidLoginDataError) Error() string {
8     return e.Message
9 }
```

myerrors >  PasswordError.go > ...

```
1 package myerrors
2
3 type IncorrectPasswordError struct {
4     Message string
5 }
6
7 func (e *IncorrectPasswordError) Error() string {
8     return e.Message
9 }
10
11 type InvalidPasswordError struct {
12     Message string
13 }
14
15 func (e *InvalidPasswordError) Error() string {
16     return e.Message
17 }
```

Пример (5/6)

cmd > go main.go > ...

```
1  package main
2
3  import (
4      "fmt"
5      "main/address"
6      "main/person"
7  )
8
9  func main() {
10     address1 := address.NewAddress("Russia", "Saint-P", "Vavilovikh", "10c2")
11     person1, err := person.NewPerson("Misha", "Shisha", 7, address1)
12     if err != nil {
13         fmt.Println(err.Error())
14     } else {
15         fmt.Println(*person1)
16         user1, err := person.NewUser("Misha", "Shisha", 7, address1, "misha@gmail.com", "123456789")
17         if err != nil {
18             fmt.Println(err.Error())
19         } else {
20             fmt.Println(*user1)
21             fmt.Println(user1.Login("misha@gmail.com", "123456789"))
22             user1.ChangePassword("123456789", "zxcasdqwe")
23         }
24     }
25 }
26
```

Пример (6/6)

```
user1.Address. |
```

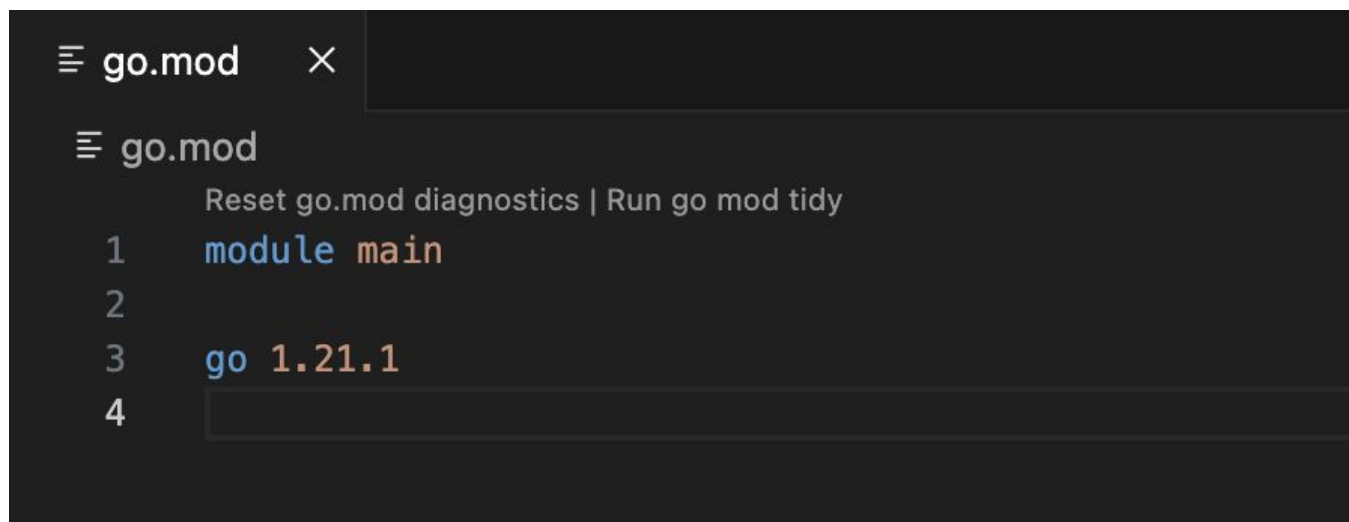
- GetAddress func() (string, string, string, string)
- GetBuilding
- GetCity
- GetCountry
- GetStreet
- SetAddress
- SetBuilding
- SetCity
- SetCountry
- SetStreet
- print!
- var!

Модули

- ▣ Модули - более высокий уровень организации кода, они объединяют несколько пакетов в единый юнит совместного управления зависимостями. Каждый пакет, определенный в модуле может быть использован в коде этого модуля. Модуль также может быть зависим от других модулей.

Файл go.mod

Файл go.mod - это корень управления зависимостями в GoLang. Все модули, которые необходимы или будут использоваться в проекте, хранятся в файле go.mod.



```
go.mod x
go.mod
Reset go.mod diagnostics | Run go mod tidy
1 module main
2
3 go 1.21.1
4
```

Использование подмодулей(1/3)

myproject

```
├─ module1
│   └─ module1.go
├─ module2
│   └─ module2.go
├─ main.go
└─ go.mod
```


Использование подмодулей(2/3)

Необходимо создать **go.mod** файлы для каждого подмодуля и импортировать их в наш модуль.

```
module1 > ≡ go.mod
      Reset go.mod diagnostics | Run go mod tidy
1  module module1/module1
2
3  go 1.21.3
```

```
module2 > ≡ go.mod
      Reset go.mod diagnostics | Run go mod tidy
1  module main/module2
2
3  go 1.21.1
```

```
module1 > go module1.go > ...
1  package module1
2
3  import "fmt"
4
5  func SayHello() {
6      fmt.Printf("Hello from first module!")
7  }
```

```
module2 > go module2.go > ...
1  package module2
2
3  import "fmt"
4
5  func SayHi() {
6      fmt.Printf("Hi from second module!")
7  }
```

Использование подмодулей(3/3)

```
GO main.go > ...
```

```
1  package main
2
3  import (
4      "fmt"
5      "./module1/module1"
6      "./module2/module2"
7  )
8
9  func main() {
10     module2.SayHi()
11     module1.SayHello()
12     fmt.Printf("Modules, nice to meet you!")
13 }
```

Загрузка внешних модулей

- Для загрузки внешнего модуля необходимо воспользоваться командой - `go get`

```
tronyagina@192 test % go get rsc.io/quote
go: downloading rsc.io/quote v1.5.2
go: downloading rsc.io/sampler v1.3.0
go: downloading golang.org/x/text v0.0.0-20170915032832-14c0d48ead0c
go: added golang.org/x/text v0.0.0-20170915032832-14c0d48ead0c
go: added rsc.io/quote v1.5.2
go: added rsc.io/sampler v1.3.0
```

≡ go.mod

Reset go.mod diagnostics | Run go mod tidy | Create vendor directory

```
1 module main
```

```
2
```

```
3 go 1.21.1
```

```
4
```

Check for upgrades | Upgrade transitive dependencies | Upgrade direct dependencies

```
5 require (
```

```
6     golang.org/x/text v0.0.0-20170915032832-14c0d48ead0c // indirect
```

```
7     rsc.io/quote v1.5.2 // indirect
```

```
8     rsc.io/sampler v1.3.0 // indirect
```

```
9 )
```

Загрузка внешних модулей

После добавления зависимостей. В файл **go.mod** добавилась директива **require()**, которая содержит определения подключаемых зависимостей. Помимо этого появился файл **go.sum**, который содержит контрольную сумму для подключаемых пакетов.

≡ go.sum

```
1  golang.org/x/text v0.0.0-20170915032832-14c0d48ead0c h1:qg0Y6WgZ0aTkIIMiVjBQcw93ERBE4m30iBm00nKL0i8=
2  golang.org/x/text v0.0.0-20170915032832-14c0d48ead0c/go.mod h1:NqM8EU0U14njkJ3fqMW+pc6Ldnwhi/IjpwHt7yyuw0Q=
3  rsc.io/quote v1.5.2 h1:w5fcysjrx7yqtD/a0+QwRjYZ0KnaM9Uh2b40tElTs3Y=
4  rsc.io/quote v1.5.2/go.mod h1:LzX7hefJvL54yjeFDEDHNONDjII0t9xZLPXsUe+TKr0=
5  rsc.io/sampler v1.3.0 h1:7uVkiFmeBqHfdjD+gZwtXXI+R0DJ2Wc407MPEh/QiW4=
6  rsc.io/sampler v1.3.0/go.mod h1:T1hPZKmBbMNahiBKfy5HrXp6adAjACjK9JXDnKaTXpA=
7
```