avito.tech

Москва 2024

Популярные ошибки в golang и как их избежать

Дмитрий Королев

Дмитрий Королев

Backend-разработчик в Growth

- Software Engineer
- Профессионально занимаюсь разработкой с 2020 года.
- Рад ответить на вопросы tg: @jimiliani.

Массивы и слайсы

Массивы и слайсы

```
type slice struct {
  array unsafe.Pointer
  len   int
  cap   int
}
```

Массивы и слайсы

- при создании нового слайса его len == cap, если вы не указали иное значение в make
  slice1 := []int{1,2,3}
  slice2 := make([]int, 3)
  и slice1 и slice2 будут иметь len == cap == 3

- скорость роста вместимости слайса, начиная с 1.20 изменилась и теперь плавно уменьшается с x2 для слайсов с cap<512 до x1.3 для слайсов с cap>4096

```
type slice struct {
    array unsafe.Pointer
    len   int
    cap   int
}
```

Массивы и слайсы

```go
func changeSliceValues(s []int) {
  s[0] = 1
}

func main() {
  slice := []int{0}
  fmt.Println(slice) // [0]
  changeSliceValues(slice)
  fmt.Println(slice) // [1]
}
```

Массивы и слайсы

```go
func changeSliceValues(s []int) {
  s[0] = 1
  s = append(s, 2)
  s[0] = 3
}

func main() {
  slice := []int{0}
  fmt.Println(slice) // [0]
  changeSliceValues(slice)
  fmt.Println(slice) // [1]
}
```

Массивы и слайсы

```
func main() {
  slice := []int{0, 0, 0}
  newSlice := slice[0:2]
  newSlice = append(newSlice, 1)
  fmt.Println(slice) // [0, 0, 1]
}
```

Массивы и слайсы

```
func main() {
  slice := []int{0, 0, 0}
  newSlice := make([]int, 2)
  copy(newSlice, slice)
  newSlice = append(newSlice, 1)
  fmt.Println(slice) // [0, 0, 0]
  fmt.Println(newSlice) // [0, 0, 1]
}
```

Массивы и слайсы

```
func receiveArticle() string {
  ...
}

func consumeNewsArticles() {
  for {
     article := receiveArticle()
     storeArticlePreview(getArticlePreview([]rune(article)))
  }
}

func getArticlePreview(article []rune) []rune {
  return article[:100]
}
```

Массивы и слайсы

```
func receiveArticle() string {
  ...
}

func consumeNewsArticles() {
  for {
    article := receiveArticle()
    storeArticlePreview(getArticlePreview([]rune(article)))
  }
}

func getArticlePreview(article []rune) []rune {
  return article[:100]
}
```

Строки, руны и байты

8

6

4

7

2

5

1

3

Массивы и слайсы

```
func main() {
  hello := "Hello World"
  helloRunes := []rune(hello)

  fmt.Println(helloRunes[:5]) // [72 101 108 108 111]
  fmt.Println(string(helloRunes[:5])) // Hello
  fmt.Println(hello[:5]) // Hello
}
```

Массивы и слайсы

```
func main() {
  hello := "你好世界"
  helloRunes := []rune(hello)

  fmt.Println(helloRunes[:2])          // [20320 22909]
  fmt.Println(string(helloRunes[:2])) // 你好
  fmt.Println(hello[:2])               // ?
}
```

Массивы и слайсы

```go
func main() {
  hello := "你好世界"
  fmt.Println(hello[:2])  // bytes
  fmt.Println(len(hello)) // bytes

  for i, c := range hello {
     fmt.Println(i, c) // bytes index, rune
  }
}
```

Каналы

8

6

4

7

2

5

1

3

Каналы

Operation	Channel state	Result
Read	`nil`	Block
	Open and Not Empty	Value
	Open and Empty	Block
	Closed	<default value>, false
	Write Only	Compilation Error
Write	`nil`	Block
	Open and Full	Block
	Open and Not Full	Write Value
	Closed	**panic**
	Receive Only	Compilation Error
`close`	`nil`	**panic**
	Open and Not Empty	Closes Channel; reads succeed until channel is drained, then reads produce default value
	Open and Empty	Closes Channel; reads produces default value
	Closed	**panic**
	Receive Only	Compilation Error

Каналы

Operation	Channel state	Result
Read		
	Closed	<default value>, false
Write		
	Closed	panic
close		
	Closed	panic

Каналы

“A sender can `close` a channel to indicate that no more values will be sent.”

«Отправитель может закрыть канал, чтобы указать, что значения больше не будут отправляться.»

Каналы

```
func writeToChan(ch chan<- int) {
  ch <- 1
  ch <- 2
  ch <- 3
  close(ch)
}

func main() {
  ch := make(chan int)

  go writeToChan(ch)

  for value := range ch {
     fmt.Println(value)
     // some logic
  }
}
```

Каналы

func After

```
func After(d Duration) <-chan Time
```

After waits for the duration to elapse and then sends the current time on the returned channel. It is equivalent to NewTimer(d).C. The underlying Timer is not recovered by the garbage collector until the timer fires. If efficiency is a concern, use NewTimer instead and call Timer.Stop if the timer is no longer needed.

Каналы

```
func consumer(ch <-chan Event) {
  for {
     select {
     case event := <-ch:
        handle(event)
     case <-time.After(time.Minute * 15):
        fmt.Println("warning: no messages received")
     }
  }
}
```

Каналы

1_000_000 * ≈200 = ≈200

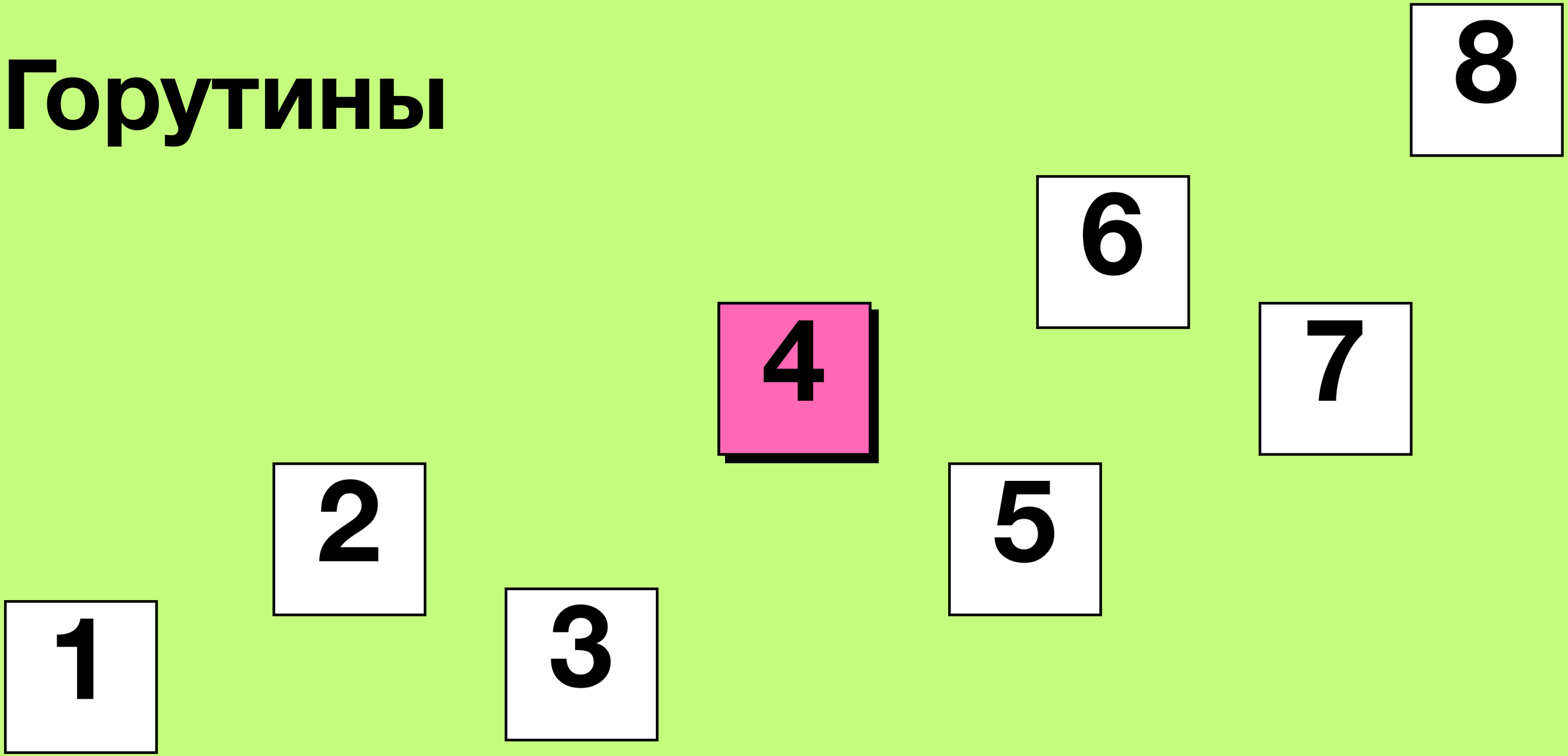
событий байт мб

Каналы

“The underlying Timer is not recovered by the garbage collector until the timer fires. If efficiency is a concern, use NewTimer instead and call Timer.Stop if the timer is no longer needed”

«Таймер не будет собран сборщиком мусора до тех пор, пока таймер не отработает. Если эффективность имеет значение, используйте вместо него NewTimer и вызовите Timer.Stop, если таймер больше не нужен.»

Горутины



Горутины

```
func main() {
  digits := []int64{1, 2, 3, 4, 5}
  var sum int64 = 0
  var wg sync.WaitGroup
  for _, value := range digits {
     go func() {
        wg.Add(1)
        defer wg.Done()
        atomic.AddInt64(&sum, value)
     }()
  }

  wg.Wait()

  fmt.Println(sum)
}
```

Горутины

Go Wiki: LoopvarExperiment

Table of Contents

For Go 1.22, the Go team is considering changing the semantics of for loop variables to prevent unintended sharing in per-iteration closures and goroutines. Go 1.21 contains a preliminary implementation of the change, enabled by setting `GOEXPERIMENT=loopvar` when building your program. We invite anyone who wants to help us understand the effects of the change to try using `GOEXPERIMENT=loopvar` and let us know about any problems or successes encountered.

This page answers frequently asked questions about the change.

Пакеты sync и atomic

8

6

4

7

2

5

1

3

Пакеты sync и atomic

```go
type WaitGroup struct {
  noCopy noCopy

  state atomic.Unit64
  sema unit32
}
```

Пакеты sync и atomic

1) .Add(delta int) увеличивает значение семафора на переданное значение
2) .Done() уменьшает значение семафора на единицу
3) .Wait() блокирует выполнение до тех пор, пока значение семафора не станет равно нулю

```
type WaitGroup struct {
  noCopy noCopy

  state atomic.Unit64
  sema unit32
}
```

Пакеты sync и atomic

```
func main() {
  digits := []int64{1, 2, 3, 4, 5}
  var sum int64 = 0
  var wg sync.WaitGroup
  for _, value := range digits {
    go func() {
      wg.Add(1)
      defer wg.Done()
      atomic.AddInt64(&sum, value)
    }()
  }

  wg.Wait()

  fmt.Println(sum)
}
```

Пакеты sync и atomic

```go
type Counter struct {
  m        sync.Mutex
  counters map[string]int
}
func (c Counter) increment(key string) {
  c.m.Lock()
  defer c.m.Unlock()
  c.counters[key]++
}
func (c Counter) IncrementMultiple(key string, n int) {
  for i := 0; i < n; i++ {
    c.increment(key)
  }
}
```

```go
func main() {
  c := Counter{counters:
map[string]int{"key1": 0, "key2": 0}}

  go c.IncrementMultiple("key1", 100000)
  go c.IncrementMultiple("key1", 100000)

  time.Sleep(300 * time.Millisecond)
  fmt.Println(c.counters)
}
```

Пакеты sync и atomic

```go
type Counter struct {
  m        sync.Mutex
  counters map[string]int
}
func (c Counter) increment(key string) {
  c.m.Lock()
  defer c.m.Unlock()
  c.counters[key]++
}
func (c Counter) IncrementMultiple(key string, n int) {
  for i := 0; i < n; i++ {
    c.increment(key)
  }
}
```

```go
func main() {
  c := Counter{counters:
map[string]int{"key1": 0, "key2": 0}}

  go c.IncrementMultiple("key1", 100000)
  go c.IncrementMultiple("key1", 100000)

  time.Sleep(300 * time.Millisecond)
  fmt.Println(c.counters)
}
```

fatal error: concurrent map writes
<goroutines stack>
Process finished with the exit code 2

Пакеты sync и atomic

main.go

```go
package main

import (
    "fmt"
    "sync/atomic"
)

func main() {
    var num int64 = 1
    go func() {
        for {
            atomic.AddInt64(&num, delta: 1)
        }

    }()

    for {
        if atomic.LoadInt64(&num)%2 == 0 {
            fmt.Println(atomic.LoadInt64(&num))
            return
        }
    }
}
```

Run: go build test1

```
GOROOT=/Users/dskorolev/go/go1.21.4 #gosetup
GOPATH=/Users/dskorolev/go #gosetup
/Users/dskorolev/go/go1.21.4/bin/go build -o /private/var/folders/qh/w5_sb2356xn_q524l9hh5ll80000gn/T/GoLand/___go_build_test1 test1 #gosetup
/private/var/folders/qh/w5_sb2356xn_q524l9hh5ll80000gn/T/GoLand/___go_build_test1
287

Process finished with the exit code 0
```

defer

8

6

4

7

2

5

1

3

defer

```go
type ProfileType string
const (
  SimpleProfile      ProfileType = "simple"
  InvestmentProfile ProfileType = "investment"
  BusinessProfile    ProfileType = "business"
)
type Profile struct {
  Type ProfileType
}
```

```go
func (p *Profile) GetBalance() (balance
int) {
  switch p.Type {
  case BusinessProfile:
     return
p.getBusinessProfileBalance()
  case InvestmentProfile:
     return
p.getInvestmentProfileBalance()
  case SimpleProfile:
     return p.getSimpleProfileBalance()
  default:
     panic("unknown profile type")
  }
}
```

defer

```go
type ProfileType string
const (
  SimpleProfile      ProfileType = "simple"
  InvestmentProfile ProfileType = "investment"
  BusinessProfile    ProfileType = "business"
)
type Profile struct {
  Type ProfileType
}
```

```go
func (p *Profile) GetBalance() (balance int) {
  defer fmt.Println("profile balance:", balance)
  switch p.Type {
  case BusinessProfile:
     return p.getBusinessProfileBalance()
  case InvestmentProfile:
     return p.getInvestmentProfileBalance()
  case SimpleProfile:
     return p.getSimpleProfileBalance()
  default:
     panic("unknown profile type")
  }
}
```

defer

“The arguments to the deferred function (which include the receiver if the function is a method) are evaluated when the *defer* executes, not when the *call* executes”.

«Значение аргументов для функции в defer (в том числе и ресивер метода) вычисляются в момент исполнения defer, а не в момент исполнения функции»

defer

```go
type ProfileType string
const (
  SimpleProfile      ProfileType = "simple"
  InvestmentProfile ProfileType = "investment"
  BusinessProfile    ProfileType = "business"
)
type Profile struct {
  Type ProfileType
}
```

```go
func (p *Profile) GetBalance() (balance int) {
  defer func() {
    fmt.Println("profile balance:", balance)
  }()
  switch p.Type {
  case BusinessProfile:
     return p.getBusinessProfileBalance()
  case InvestmentProfile:
     return p.getInvestmentProfileBalance()
  case SimpleProfile:
     return p.getSimpleProfileBalance()
  default:
     panic("unknown profile type")
  }
}
```

Интерфейсы

8

6

4

7

2

5

1

3

Интерфейсы

```go
type Requester interface {
  MakeRequest() int
}
type ConcreteRequester struct {
  someField int
}
func (r *ConcreteRequester) MakeRequest() int {
  return r.someField
}
```

```go
func makeRequester(someVal int) Requester {
  var requester *ConcreteRequester
  if someVal > 0 {
     requester = &ConcreteRequester{someField: someVal}
  }
  return requester
}

func main() {
  requester := makeRequester(0)
  fmt.Println("got requester: ", requester)
  if requester == nil {
     fmt.Println("requester is nil")
  } else {
     fmt.Println("requester is not nil")
  }
}
```

Интерфейсы

```go
type Requester interface {
  MakeRequest() int
}
type ConcreteRequester struct {
  someField int
}
func (r *ConcreteRequester) MakeRequest() int {
  return r.someField
}
```

```go
func makeRequester(someVal int) Requester {
  var requester *ConcreteRequester
  if someVal > 0 {
     requester = &ConcreteRequester{someField: someVal}
  }
  return requester
}

func main() {
  requester := makeRequester(0)
  fmt.Println("got requester: ", requester)
  if requester == nil {
     fmt.Println("requester is nil")
  } else {
     fmt.Println("requester is not nil")
  }
}
```

```
got requester:  <nil>
requester is not nil
```

Интерфейсы

```
type eface struct {
  _type *_type
  data  unsafe.Pointer
}
type iface struct {
  tab  *itab
  data unsafe.Pointer
}
type itab struct {
  _type *_type
  …
}
```

Интерфейсы

```go
type Requester interface {
  MakeRequest() int
}
type ConcreteRequester struct {
  someField int
}
func (r *ConcreteRequester) MakeRequest()
 int {
  return r.someField
}
```

```go
func makeRequester(someVal int) Requester {
  var requester *ConcreteRequester
  if someVal > 0 {
     requester = &ConcreteRequester{someField: someVal}
  }
  return requester
}

func main() {
  requester := makeRequester(0)
  fmt.Println("got requester: ", requester)
  fmt.Printf("requester=(%T,%v)\n", requester, requester)
  if requester == nil {
     fmt.Println("requester is nil")
  } else {
     fmt.Println("requester is not nil")
  }
}
```

```
got requester:  <nil>
requester=(*main.ConcreteRequester,<nil>)
requester is not nil
```

Особенности вендоринга

8

6

4

7

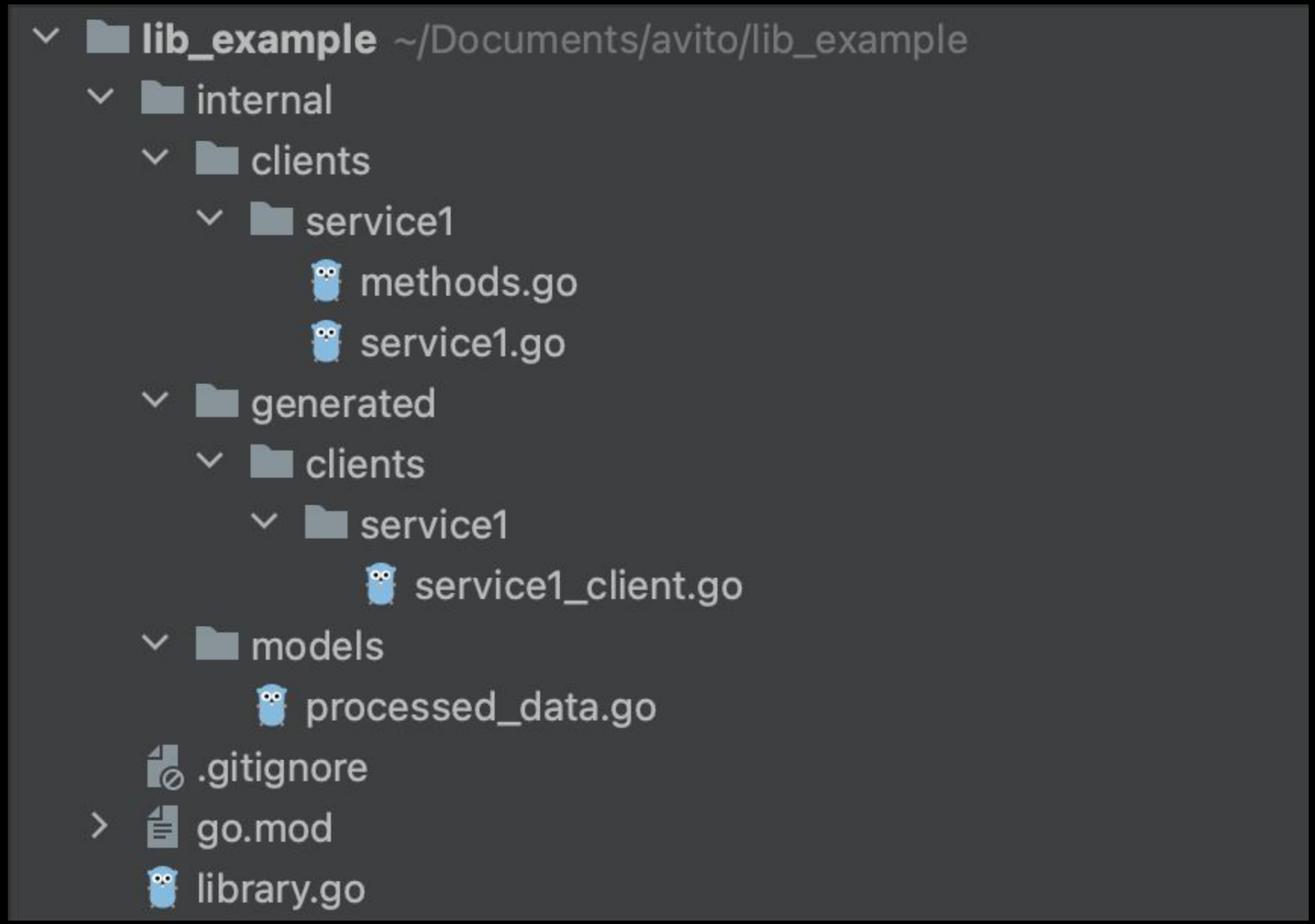
2

5

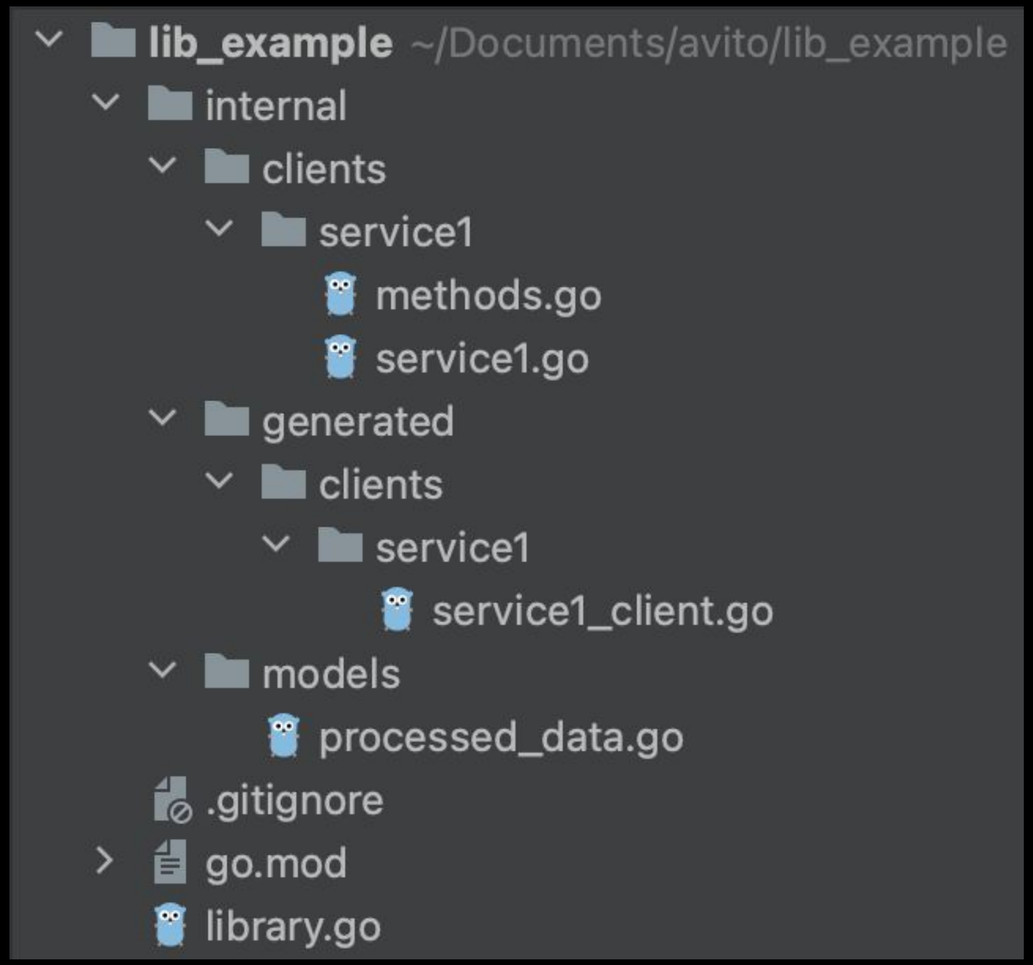
1

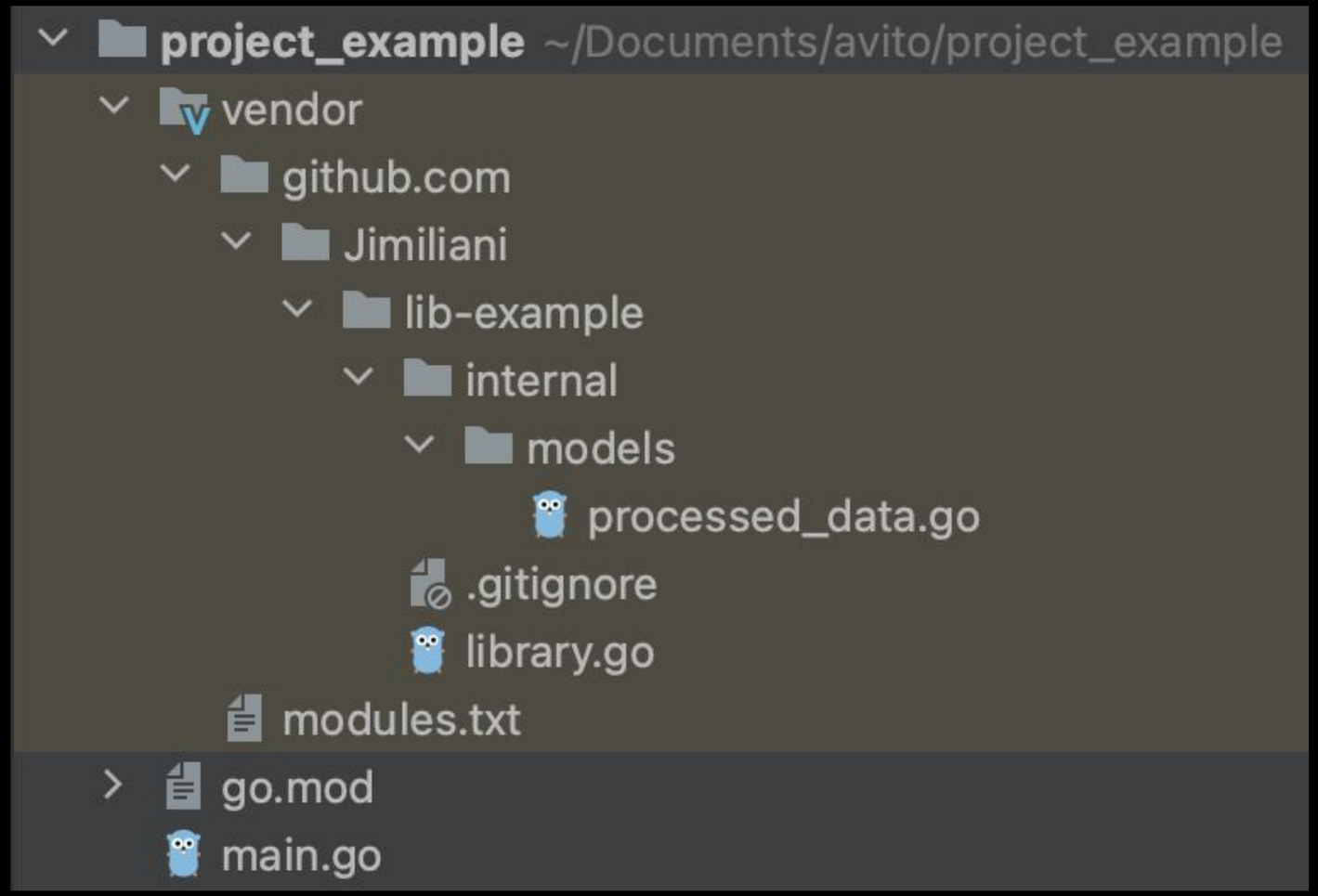
3

Особенности вендоринга



Особенности вендоринга





Особенности вендоринга

“The go mod vendor command constructs a directory named vendor in the main module’s root directory containing copies of all packages needed to build and test packages in the main module”.

avito.tech Москва 2024

Популярные ошибки в golang и как их избежать

Дмитрий Королев