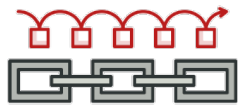





[🏠](#) / [Паттерны проектирования](#) / [Цепочка обязанностей](#) / [Go](#)



Цепочка обязанностей на Go

Цепочка обязанностей — это поведенческий паттерн, позволяющий передавать запрос по цепочке потенциальных обработчиков, пока один из них не обработает запрос.

Избавляет от жёсткой привязки отправителя запроса к его получателю, позволяя выстраивать цепь из различных обработчиков динамически.

 [Подробнее о паттерне Цепочка обязанностей →](#)

Навигация

 [Интро](#)

 [Концептуальный пример](#)

 [department](#)


 [reception](#)

[Главная](#) [Рефакторинг](#) [Паттерны](#) [Премиум контент](#)
[Форум](#) [Связаться](#)



© 2014-2023 Refactoring.Guru.

Все права защищены.

 Иллюстрации нарисовал Дмитрий Жарт

[Условия использования](#)

[Политика конфиденциальности](#)

[Использование контента](#) [About us](#)

Концептуальный пример

Давайте рассмотрим паттерн Цепочка обязанностей на примере приложения больницы. Госпиталь может иметь разные помещения, например:

- Приемное отделение
- Доктор
- Комната медикаментов
- Кассир

Когда пациент прибывает в больницу, первым делом он попадает в Приемное отделение, оттуда – к Доктору, затем в Комнату медикаментов, после этого – к Кассиру, и так далее. Пациент проходит по цепочке помещений, в которой каждое отправляет его по ней дальше сразу после выполнения своей функции.

Этот паттерн можно применять в случаях, когда для выполнения одного запроса есть несколько кандидатов, и когда вы не хотите, чтобы клиент сам выбирал исполнителя. Важно знать, что клиента необходимо оградить от исполнителей, ему необходимо знать лишь о существовании первого звена цепи.

Используя пример больницы, пациент сперва попадает в Приемное отделение. Затем, зависимо от его состояния, Приемное отделение отправляет его к следующему исполнителю в цепи.

department.go: Интерфейс обработчика

```
package main

type Department interface {
    execute(*Patient)
    setNext(Department)
}
```

reception.go: Конкретный обработчик

```

package main

import "fmt"

type Reception struct {
    next Department
}

func (r *Reception) execute(p *Patient) {
    if p.registrationDone {
        fmt.Println("Patient registration already done")
        r.next.execute(p)
        return
    }
    fmt.Println("Reception registering patient")
    p.registrationDone = true
    r.next.execute(p)
}

func (r *Reception) setNext(next Department) {
    r.next = next
}

```

 doctor.go: Конкретный обработчик

```
package main

import "fmt"

type Doctor struct {
    next Department
}

func (d *Doctor) execute(p *Patient) {
    if p.doctorCheckUpDone {
        fmt.Println("Doctor checkup already done")
        d.next.execute(p)
        return
    }
    fmt.Println("Doctor checking patient")
    p.doctorCheckUpDone = true
    d.next.execute(p)
}

func (d *Doctor) setNext(next Department) {
    d.next = next
}
```

 **medical.go: Конкретный обработчик**

```

package main

import "fmt"

type Medical struct {
    next Department
}

func (m *Medical) execute(p *Patient) {
    if p.medicineDone {
        fmt.Println("Medicine already given to patient")
        m.next.execute(p)
        return
    }
    fmt.Println("Medical giving medicine to patient")
    p.medicineDone = true
    m.next.execute(p)
}

func (m *Medical) setNext(next Department) {
    m.next = next
}

```

cashier.go: Конкретный обработчик

```

package main

import "fmt"

type Cashier struct {
    next Department
}

func (c *Cashier) execute(p *Patient) {
    if p.paymentDone {
        fmt.Println("Payment Done")
    }
    fmt.Println("Cashier getting money from patient patient")
}

func (c *Cashier) setNext(next Department) {
    c.next = next
}

```

patient.go

```
package main

type Patient struct {
    name           string
    registrationDone bool
    doctorCheckUpDone bool
    medicineDone   bool
    paymentDone    bool
}
```

main.go: Клиентский код

```
package main

func main() {

    cashier := &Cashier{}

    //Set next for medical department
    medical := &Medical{}
    medical.setNext(cashier)

    //Set next for doctor department
    doctor := &Doctor{}
    doctor.setNext(medical)

    //Set next for reception department
    reception := &Reception{}
    reception.setNext(doctor)

    patient := &Patient{name: "abc"}
    //Patient visiting
    reception.execute(patient)
}
```

output.txt: Результат выполнения

```
Reception registering patient
```

Doctor checking patient
Medical giving medicine to patient
Cashier getting money from patient patient

По материалам: *Go*lang By Example

ВЕРНУТЬСЯ НАЗАД

ЧИТАЕМ ДАЛЬШЕ

← Заместитель на Go

Команда на Go →

Цепочка обязанностей на других языках программирования

