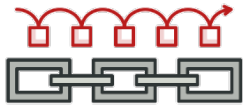

Chain of Responsibility in Go

Chain of Responsibility is behavioral design pattern that allows passing request along the chain of potential handlers until one of them handles request.

The pattern allows multiple objects to handle the request without coupling sender class to the concrete classes of the receivers. The chain can be composed dynamically at runtime with any handler that follows a standard handler interface.

Learn more about Chain of Responsibility →

Navigation

Conceptual Example

Let's look at the Chain of Responsibility pattern with the case of a hospital app. A hospital could have multiple departments such as:

- Reception
- Doctor
- Medical examination room
- Cashier

Whenever any patient arrives, they first get to Reception, then to Doctor, then to Medicine Room, and then to Cashier (and so on). The patient is being sent through a chain of departments, where each department sends the patient further down the chain once their function is completed.

The pattern is applicable when there are multiple candidates to process the same request. It is also useful when you don't want the client to choose the receiver as there are multiple objects can handle the request. Another useful case is when you want to decouple the client from receivers—the client will only need to know the first element in the chain.

As in the example of the hospital, a patient first goes to the reception. Then, based upon a patient's current status, reception sends up to the next handler in the chain.

department.go: Handler interface

```
package main

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

reception.go: Concrete handler

```
package main
```

```go
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: Concrete handler

```go
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: Concrete handler

```
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: Concrete handler

```
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: Client code

```
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: Execution result

```
Reception registering patient
Doctor checking patient
Medical giving medicine to patient
Cashier getting money from patient patient
```

Based on: ***Golang By Example***

RETURN **READ NEXT**

← Proxy in Go | Command in Go →

Chain of Responsibility in Other Languages