Distributed Communication 12th practice

Li Jianhao
lijianhao288@hotmail.com

1 Basics

1.1 Generator Example

New programs:

```
package main
import (
    "fmt"
    "github.com/streadway/amqp"
    "log"
    "os/exec"
)
func main() {
    conn, err := amqp.Dial("amqp://guest:guest@localhost:5672/")
    failOnError(err, "Failed to connect to RabbitMQ")
    defer conn.Close()
    ch, err := conn.Channel()
    failOnError(err, "Failed to open a channel")
    err = ch.ExchangeDeclare("dispatch", "direct", false, true, false, false, nil)
    failOnError(err, "Failed to declare an exchange")
    queueArg, err := ch.QueueDeclare("generatorQueue", false, true, false, false, nil)
    failOnError(err, "Failed to declare a queue")
    err = ch.QueueBind(queueArg.Name, "generator", "dispatch", false, nil)
    failOnError(err, "Failed to bind a queue")
    msg, err := ch.Consume(queueArg.Name, "", false, false, false, false, nil)
    failOnError(err, "Failed to register a consumer")
    go func() {
        for d := range msg {
            cmd := exec.Command("cmd", "/C", "start", "go", "run", "./Reply_Server.go")
            err = cmd.Run()
            failOnError(err, "Failed to generate server")
            fmt.Println("generated one server")
            d.Ack(false)
        }
    }()
    forever := make(chan bool)
    <-forever
}
func failOnError(err error, msg string) {
    if err != nil {
        log.Fatalf("%s: %s", msg, err)
    }
}
```

Listing 1: Generator example, generator

```
package main
import (
    "fmt"
    "github.com/streadway/amqp"
    "log"
)
func main() {
    numWorkers := 10
    conn, err := amqp.Dial("amqp://guest:guest@localhost:5672/")
    failOnError(err, "Failed to connect to RabbitMQ")
    defer conn.Close()
    cho, err := conn.Channel()
    failOnError(err, "Failed to open a channel")
    err = cho.ExchangeDeclare("dispatch", "direct",
        false, true, false, false, nil)
    failOnError(err, "Failed to declare an exchange")
    for i := 0; i < numWorkers; i++ {
        msg := "1"
        err = cho.Publish("dispatch", "generator", false, false,
            amqp.Publishing{
                ContentType: "text/plain",
                Body:        []byte(msg),
            })

        failOnError(err, "Failed to publish a message")
        fmt.Printf("Organizer Published %s \n", string(msg))
    }

}
func failOnError(err error, msg string) {
    if err != nil {
        log.Fatalf("%s: %s", msg, err)
    }
}
```

Listing 2: Generator example, organizer

Programs in previous example:

```
package main

import (
    "fmt"
    "github.com/rs/xid"
    "github.com/streadway/amqp"
    "log"
    "strconv"
    "sync"
)

func main() {
    conn1, err := amqp.Dial("amqp://guest:guest@localhost:5672/")
    failOnError(err, "Failed to connect to RabbitMQ")
    defer conn1.Close()

    conn2, err := amqp.Dial("amqp://guest:guest@localhost:5672/")
    failOnError(err, "Failed to connect to RabbitMQ")
    defer conn2.Close()

    cho, err := conn1.Channel()
```

```
    failOnError(err, "Failed to open a channel")
    defer cho.Close()
    chi, err := conn2.Channel()
    failOnError(err, "Failed to open a channel")
    defer chi.Close()

    err = cho.ExchangeDeclare("replyExchange", "direct", false, true, false, false, nil)
    failOnError(err, "Failed to declare an exchange")

    q, err := chi.QueueDeclare("", false, true, false, false, nil)
    failOnError(err, "Failed to declare a queue")

    err = chi.QueueBind(q.Name, q.Name, "replyExchange", false, nil)
    failOnError(err, "Failed to bind a queue")

    var jobCorr = make(map[string]string)
    var mu sync.Mutex

    msgs, err := chi.Consume(q.Name, "", false, false, false, false, nil)
    failOnError(err, "Failed to register a consumer")

    go func() {
        for d := range msgs {
            mu.Lock()
            v, ok := jobCorr[d.CorrelationId]
            if ok {
                fmt.Println("Job", v, "Get response:"+string(d.Body))
                delete(jobCorr, d.CorrelationId)
            } else {
                fmt.Println("Get a not related msg")
            }
            mu.Unlock()
            d.Ack(false)
        }
    }()

    ints := []string{}
    for i := 0; i < 10; i++ {
        s := strconv.Itoa(i)
        ints = append(ints, s)
    }

    for _, i := range ints {
        var corrId = randomString()
        err := cho.Publish("replyExchange", "key", false, false,
            amqp.Publishing{
                ContentType:   "text/plain",
                CorrelationId: corrId,
                ReplyTo:       q.Name,
                Body:          []byte(i),
            })

        failOnError(err, "Failed to publish")
        fmt.Println("Published " + i)
        mu.Lock()
        jobCorr[corrId] = i
        mu.Unlock()
    }

    forever := make(chan bool)
    <-forever
```

```
}
func failOnError(err error, msg string) {
    if err != nil {
        log.Fatalf("%s: %s", msg, err)
    }
}

func randomString() string {
    guid := xid.New()
    return guid.String()
}
```

Listing 3: Reply example, Client

```
package main

import (
    "fmt"
    "github.com/streadway/amqp"
    "log"
    "strconv"
)

func main() {
    conn1, err := amqp.Dial("amqp://guest:guest@localhost:5672/")
    failOnError(err, "Failed to connect to RabbitMQ")
    defer conn1.Close()

    conn2, err := amqp.Dial("amqp://guest:guest@localhost:5672/")
    failOnError(err, "Failed to connect to RabbitMQ")
    defer conn2.Close()

    cho, err := conn1.Channel()
    failOnError(err, "Failed to open a channel")
    defer cho.Close()
    chi, err := conn2.Channel()
    failOnError(err, "Failed to open a channel")
    defer chi.Close()

    err = cho.ExchangeDeclare("replyExchange", "direct", false, true, false, false, nil)
    failOnError(err, "Failed to declare an exchange")

    q, err := chi.QueueDeclare("jobQueue", false, true, false, false, nil)
    failOnError(err, "Failed to declare a queue")

    err = chi.QueueBind(q.Name, "key", "replyExchange", false, nil)
    failOnError(err, "Failed to bind a queue")

    msgs, err := chi.Consume(q.Name, "", false, false, false, false, nil)
    failOnError(err, "Failed to register a consumer")

    forever := make(chan bool)

    go func() {
        for d := range msgs {
            i, err := strconv.Atoi(string(d.Body))
            failOnError(err, "Failed to convert")
            result := strconv.Itoa(i * 2)
            fmt.Println("Reply result:", result)
            err = cho.Publish(
                "replyExchange", d.ReplyTo, false, false,
```

```
                amqp.Publishing{
                    ContentType:   "text/plain",
                    CorrelationId: d.CorrelationId,
                    Body:          []byte(result),
                })
            failOnError(err, "Failed to publish a message")
            d.Ack(false)
        }
    }()
    fmt.Println("Waiting for jobs")
    <-forever
}

func failOnError(err error, msg string) {
    if err != nil {
        log.Fatalf("%s: %s", msg, err)
    }
}
```

Listing 4: Reply example, Server

Output:
Run three generators and one organizer.

Three generators:

```
go run generator.go
generated one server
generated one server
generated one server
generated one server
```

```
go run generator.go
generated one server
generated one server
generated one server
```

```
go run generator.go
generated one server
generated one server
generated one server
```

Organizer:

```
go run organizer.go
Organizer Published 1
Organizer Published 1
Organizer Published 1
Organizer Published 1
Organizer Published 1
Organizer Published 1
Organizer Published 1
Organizer Published 1
Organizer Published 1
Organizer Published 1
```

Run the reply client.

Reply client:

```
go run Reply_Client.go
Published job:0
Published job:1
Published job:2
Published job:3
Published job:4
Published job:5
Published job:6
Published job:7
Published job:8
Published job:9
Job: 0 Got response:0
Job: 1 Got response:2
Job: 2 Got response:4
Job: 3 Got response:6
Job: 5 Got response:10
Job: 8 Got response:16
Job: 7 Got response:14
Job: 4 Got response:8
Job: 9 Got response:18
Job: 6 Got response:12
```

Reply servers:

```
Waiting for jobs
Received job: 0 Published response: 0
```

```
Waiting for jobs
Received job: 2 Published response: 4
```

```
Waiting for jobs
Received job: 4 Published response: 8
```

```
Waiting for jobs
Received job: 5 Published response: 10
```

```
Waiting for jobs
Received job: 1 Published response: 2
```

```
Waiting for jobs
Received job: 3 Published response: 6
```

```
Waiting for jobs
Received job: 6 Published response: 12
```

```
Waiting for jobs
Received job: 7 Published response: 14
```

```
Waiting for jobs
Received job: 9 Published response: 18
```

```
Waiting for jobs
Received job: 8 Published response: 16
```

2 Practice

2.1 p1

Create program “Solution_generator” and “Solution_organizer”.

Run one “Reply_Server” manually. Run one “Solution_generator” and one “Solution_organizer” to generate 5 “Reply_Client”. All the “Reply_Client” will send requests and get responses.