Distributed Communication 10th practice

Li Jianhao
lijianhao288@hotmail.com

1 Basics

1.1 Globally unique id generator

https://github.com/rs/xid

1. go get github.com/rs/xid
2. import "github.com/rs/xid"
3. ```
   guid := xid.New()
   uniqueId := guid.String()
   ```

1.2 Reply example

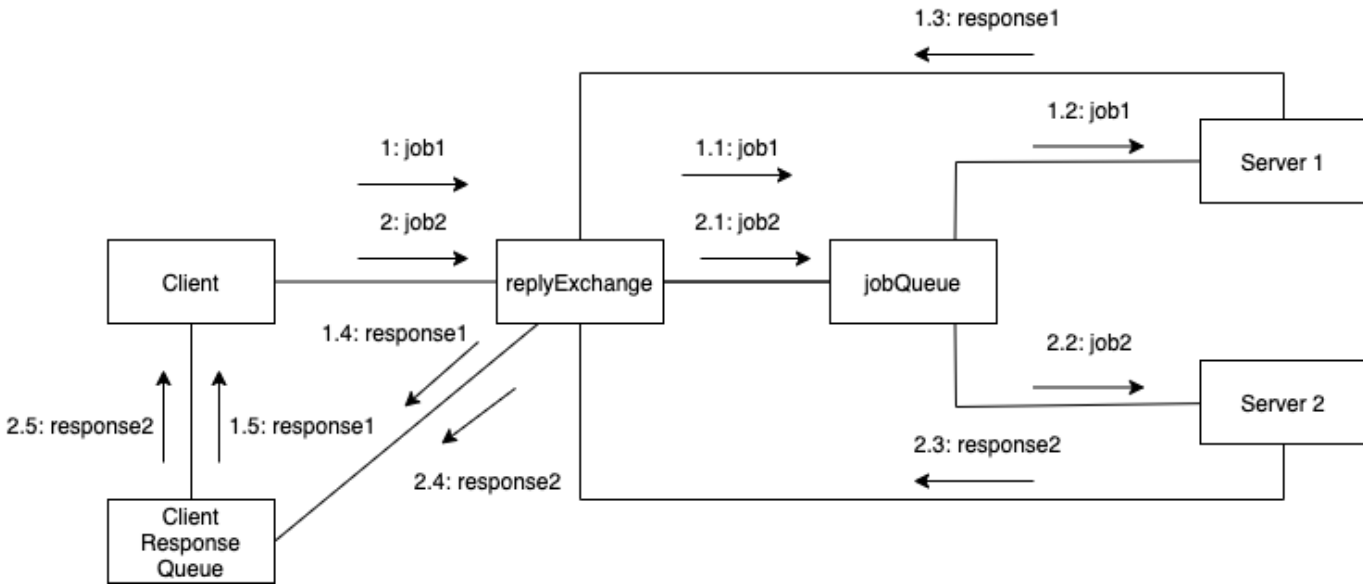

Figure 1: Reply example
   ```

Output:
First Server

```
go run Reply_Server.go
Waiting for jobs
Received job: 0 Published response: 0
Received job: 2 Published response: 4
Received job: 4 Published response: 8
Received job: 6 Published response: 12
Received job: 8 Published response: 16
```

Second Server

```
go run Reply_Server.go
Waiting for jobs
Received job: 1 Published response: 2
Received job: 3 Published response: 6
Received job: 5 Published response: 10
Received job: 7 Published response: 14
Received job: 9 Published response: 18
```

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: 1 Got response:2
Job: 3 Got response:6
Job: 5 Got response:10
Job: 7 Got response:14
Job: 9 Got response:18
Job: 0 Got response:0
Job: 2 Got response:4
Job: 4 Got response:8
Job: 6 Got response:12
Job: 8 Got response:16
```

```
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 1: 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 2: Reply example, Server

2 Practice

2.1 p1

Create 1 server program file (but you can run it multiple times) and 2 clients. The only difference of the clients is that they send different jobs. The client1 send "$http://web<0-9>.com$". The client2 send "$http://web<10-19>.com$" Create a direct exchange with the name "solutionExchange".

1. Create a function **linkTest**. It sleeps 500 milliseconds and randomly returns "Good" or "Bad".

2. The clients publish links to "solutionExchange" and print out "Published job: $< msg >$". The links will be checked by the servers. Each client will receive their own responses. For each received response message, if the response is related to the job sent by this client, this client will print out "Job: $< job >$ Got

response: $< response >$". If not, print out "Got a not related msg"
(Hint(The hint will not appear during the exam): Each client consumes the response messages from a private queue. The queues are bound to the "solutionExchange" with their generated queue name.)

3. The servers receive the jobs in a balanced round-robin way. For each received message, the server uses the function **linkTest** to get a result, send back the result to the related client, and print out "Received job: $< job >$ Published response: $< response >$". (Hint(The hint will not appear during the exam): Take care about the **ReplyTo** and the **CorrelationId**. Each server consumes the job messages from a shared queue. The name of the shared queue is known by two servers)

(Hint(The hint will not appear during the exam): Output if you run two servers.)
First server

```
<After client1 run>
go run Solution_Server.go
Waiting for jobs
<After client1 run>
Received job: http://web0.com Published response: Good
Received job: http://web2.com Published response: Good
Received job: http://web4.com Published response: Good
Received job: http://web6.com Published response: Good
Received job: http://web8.com Published response: Good
<After client2 run>
Received job: http://web10.com Published response: Bad
Received job: http://web12.com Published response: Good
Received job: http://web14.com Published response: Bad
Received job: http://web16.com Published response: Bad
Received job: http://web18.com Published response: Bad
```

Second server

```
go run Solution_Server.go
Waiting for jobs
<After client1 run>
Received job: http://web1.com Published response: Good
Received job: http://web3.com Published response: Good
Received job: http://web5.com Published response: Good
Received job: http://web7.com Published response: Good
Received job: http://web9.com Published response: Good
<After client2 run>
Received job: http://web11.com Published response: Bad
```

```
Received job: http://web13.com Published response: Good
Received job: http://web15.com Published response: Bad
Received job: http://web17.com Published response: Bad
Received job: http://web19.com Published response: Bad
```

Client1

```
go run Solution_Client1.go
Published job:http://web0.com
Published job:http://web1.com
Published job:http://web2.com
Published job:http://web3.com
Published job:http://web4.com
Published job:http://web5.com
Published job:http://web6.com
Published job:http://web7.com
Published job:http://web8.com
Published job:http://web9.com
Job: http://web1.com Got response:Good
Job: http://web0.com Got response:Good
Job: http://web2.com Got response:Good
Job: http://web3.com Got response:Good
Job: http://web4.com Got response:Good
Job: http://web5.com Got response:Good
Job: http://web7.com Got response:Good
Job: http://web6.com Got response:Good
Job: http://web8.com Got response:Good
Job: http://web9.com Got response:Good
```

Client2

```
go run Solution_Client2.go
Published job:http://web10.com
Published job:http://web11.com
Published job:http://web12.com
Published job:http://web13.com
Published job:http://web14.com
Published job:http://web15.com
Published job:http://web16.com
Published job:http://web17.com
Published job:http://web18.com
Published job:http://web19.com
Job: http://web10.com Got response:Bad
Job: http://web11.com Got response:Bad
Job: http://web12.com Got response:Good
Job: http://web13.com Got response:Good
Job: http://web14.com Got response:Bad
Job: http://web15.com Got response:Bad
Job: http://web17.com Got response:Bad
Job: http://web16.com Got response:Bad
Job: http://web18.com Got response:Bad
Job: http://web19.com Got response:Bad
```