Distributed Communication 11th practice

Li Jianhao
lijianhao288@hotmail.com

1 Basics

1.1 Cancel

https://pkg.go.dev/github.com/streadway/amqp#Channel.Cancel

func (ch *Channel) Cancel(consumer string, noWait bool) error

Delivery:

```
type Delivery struct {
    CorrelationId   string    // application use - correlation identifier
    ReplyTo         string    // application use - address to reply to (ex: RPC)
    // Valid only with Channel.Consume
    ConsumerTag string
    Body []byte
    ...
}
```

1.2 Parallel receive example

Output:
Sequential receive

```
...
goroutine http://web85.com:Good
goroutine http://web86.com:Good
goroutine http://web87.com:Good
goroutine http://web88.com:Good
goroutine http://web89.com:Bad
goroutine http://web90.com:Bad
goroutine http://web91.com:Bad
goroutine http://web92.com:Bad
goroutine http://web93.com:Good
goroutine http://web94.com:Bad
goroutine http://web95.com:Good
goroutine http://web96.com:Bad
goroutine http://web97.com:Good
goroutine http://web98.com:Good
goroutine http://web99.com:Bad
Time:  10.6161775s
```

Parallel Receive

```
...
goroutine 3 http://web86.com:Good
goroutine 0 http://web92.com:Good
goroutine 4 http://web94.com:Bad
goroutine 9 http://web91.com:Bad
goroutine 11 http://web89.com:Bad
goroutine 10 http://web90.com:Bad
goroutine 6 http://web93.com:Good
goroutine 14 http://web95.com:Bad
goroutine 15 http://web88.com:Good
goroutine 13 http://web96.com:Bad
goroutine 7 http://web98.com:Good
goroutine 8 http://web99.com:Good
goroutine 1 http://web97.com:Bad
Time:  745.6764ms
```

```go
package main

import (
    "fmt"
    "log"
    "math/rand"
    "sync"
    "time"

    "github.com/streadway/amqp"
)

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("pExchange", "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, "key", "pExchange", 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")
    for i := 0; i < 100; i++ {
        fakeLink := fmt.Sprintf("http://web%d.com", i)
        err := cho.Publish("pExchange", "key", false, false,
            amqp.Publishing{
                ContentType: "text/plain",
                Body:        []byte(fakeLink),
            })
        failOnError(err, "Failed␣to␣publish")
        fmt.Println("Published␣job:" + fakeLink)
    }
    err = cho.Publish("pExchange", "key", false, false,
        amqp.Publishing{
            ContentType: "text/plain",
            Body:        []byte("END"),
        })
    failOnError(err, "Failed␣to␣publish")
    fmt.Println("Published␣END")
    var wg sync.WaitGroup
    start := time.Now()
    wg.Add(1)
    go func() {
        for d := range msgs {
            s := string(d.Body)
            if s == "END" {
                err = chi.Cancel(d.ConsumerTag, false)
                failOnError(err, "Failed␣to␣cancel␣a␣consumer")
            } else {
                result := linkTest(s)
                fmt.Println("goroutine", result)
            }
            d.Ack(false)
        }
        wg.Done()
    }()
    wg.Wait()
    duration := time.Since(start)
    fmt.Println("Time:␣", duration)
}
func failOnError(err error, msg string) {
    if err != nil {
        log.Fatalf("%s:␣%s", msg, err)
    }
}
func linkTest(link string) string {
    time.Sleep(100 * time.Millisecond)
    if rand.Intn(2) == 1 {
        return link + ":Good"
    } else {
        return link + ":Bad"
    }
}
```

Listing 1: Sequential Receive

```
package main

import (
    "fmt"
    "log"
    "math/rand"
    "runtime"
    "sync"
    "time"

    "github.com/streadway/amqp"
)

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("pExchange", "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, "key", "pExchange", false, nil)
    failOnError(err, "Failed␣to␣bind␣a␣queue")
    msgs, err := chi.Consume(q.Name, "linkConsumer",
        false, false, false, false, nil)
    failOnError(err, "Failed␣to␣register␣a␣consumer")
    for i := 0; i < 100; i++ {
        fakeLink := fmt.Sprintf("http://web%d.com", i)
        err := cho.Publish("pExchange", "key", false, false,
            amqp.Publishing{
                ContentType: "text/plain",
                Body:        []byte(fakeLink),
            })
        failOnError(err, "Failed␣to␣publish")
        fmt.Println("Published␣job:" + fakeLink)
    }
    err = cho.Publish("pExchange", "key", false, false,
        amqp.Publishing{
            ContentType: "text/plain",
            Body:        []byte("END"),
        })
    failOnError(err, "Failed␣to␣publish")
    fmt.Println("Published␣END")
    var wg sync.WaitGroup
    start := time.Now()
    for i := 0; i < runtime.NumCPU(); i++ {
        wg.Add(1)
        go func(index int) {
            for d := range msgs {
                s := string(d.Body)
                if s == "END" {
                    err = chi.Cancel("linkConsumer", false)
                    failOnError(err, "Failed␣to␣cancel␣a␣consumer")
                } else {
                    result := linkTest(s)
                    fmt.Println("goroutine", index, result)
                }
                d.Ack(false)
            }
            wg.Done()
        }(i)
    }
    wg.Wait()
    duration := time.Since(start)
    fmt.Println("Time:␣", duration)
}
```

```
func failOnError(err error, msg string) {
    if err != nil {
        log.Fatalf("%s:␣%s", msg, err)
    }
}
func linkTest(link string) string {
    time.Sleep(100 * time.Millisecond)
    if rand.Intn(2) == 1 {
        return link + ":Good"
    } else {
        return link + ":Bad"
    }
}
```

Listing 2: Parallel Receive

2 Practice

2.1 p1

Create 1 publisher and 2 consumers (solutionLower and solutionUpper). They use a fanout exchange with the name "fanoutExchange".

1. The publisher sends strings ("aaBB", "aABb", "AAbb", "CcDd", "EeFF", "ggHh") to "fanoutExchange".
2. Each consumer binds their private queue to the "fanoutExchange". Each consumer uses 4 goroutines to receive messages from the Golang channel parallelly. The solutionLower converts the received messages to lower case and prints out "Received: $< msg >$ Lower: $< lowerMsg >$". The solutionUpper converts the received messages to upper case and print out "Received: $< msg >$ Upper: $< upperMsg >$".

(Hint(The hint will not appear during the exam): Output)
solutionLower

```
go run solutionLower.go
Waiting for msgs
Received: aABb Lower: aabb
Received: aaBB Lower: aabb
Received: AAbb Lower: aabb
Received: EeFF Lower: eeff
Received: CcDd Lower: ccdd
Received: ggHh Lower: gghh
```

solutionUpper

```
go run solutionUpper.go
Waiting for msgs
Received: aaBB Upper: AABB
Received: aABb Upper: AABB
Received: AAbb Upper: AABB
Received: CcDd Upper: CCDD
Received: EeFF Upper: EEFF
Received: ggHh Upper: GGHH
```

solutionPublisher

```
go run solutionPublisher.go
Sent:  aaBB
Sent:  aABb
Sent:  AAbb
Sent:  CcDd
Sent:  EeFF
Sent:  ggHh
```