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 2
    ReplyTo string // application use - address to reply to (ex: RPC) 3
    // Valid only with Channel.Consume 4
    ConsumerTag string 5
    Body [] byte 6
    ...
}
```

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
                                                                                      11
goroutine http://web95.com:Good
                                                                                      12
goroutine http://web96.com:Bad
goroutine http://web97.com:Good
                                                                                      14
goroutine http://web98.com:Good
                                                                                      15
goroutine http://web99.com:Bad
                                                                                      16
Time: 10.6161775s
                                                                                      17
```

Parallel Receive

```
1
goroutine 3 http://web86.com:Good
                                                                                       2
goroutine 0 http://web92.com:Good
                                                                                       3
goroutine 4 http://web94.com:Bad
                                                                                       4
goroutine 9 http://web91.com:Bad
goroutine 11 http://web89.com:Bad
                                                                                       6
goroutine 10 http://web90.com:Bad
                                                                                       7
goroutine 6 http://web93.com:Good
                                                                                       8
goroutine 14 http://web95.com:Bad
                                                                                       9
goroutine 15 http://web88.com:Good
                                                                                       10
goroutine 13 http://web96.com:Bad
                                                                                       11
goroutine 7 http://web98.com:Good
                                                                                       12
goroutine 8 http://web99.com:Good
goroutine 1 http://web97.com:Bad
                                                                                       14
Time: 745.6764ms
                                                                                       15
```

```
package main
                                                                                                                    2
import (
                                                                                                                    3
    "fmt"
     "log"
                                                                                                                    5
     "math/rand"
                                                                                                                    6
     "sync"
     "time"
                                                                                                                    8
                                                                                                                    9
     "github.com/streadway/amqp"
                                                                                                                    10
)
                                                                                                                    11
                                                                                                                    12
func main() {
                                                                                                                    13
     conn1, err := amqp.Dial("amqp://guest:guest@localhost:5672/")
                                                                                                                    14
     failOnError(err, "Failed_to_connect_to_RabbitMQ")
                                                                                                                    15
     defer conn1.Close()
                                                                                                                    16
     conn2, err := amqp.Dial("amqp://guest:guest@localhost:5672/")
                                                                                                                    17
     failOnError(err, "Failed to connect to RabbitMQ")
                                                                                                                    18
     defer conn2.Close()
                                                                                                                    19
     cho, err := conn1.Channel()
     \texttt{failOnError(err, "Failed}_{\sqcup} \texttt{to}_{\sqcup} \texttt{open}_{\sqcup} \texttt{a}_{\sqcup} \texttt{channel"})
                                                                                                                    21
     defer cho.Close()
                                                                                                                    22
     chi, err := conn2.Channel()
                                                                                                                    23
     \texttt{failOnError}(\texttt{err}, \ \texttt{"Failed}_{\sqcup} \texttt{to}_{\sqcup} \texttt{open}_{\sqcup} \texttt{a}_{\sqcup} \texttt{channel"})
                                                                                                                    24
     defer chi.Close()
                                                                                                                    25
     err = cho.ExchangeDeclare("pExchange", "direct",
     false, true, false, false, nil) failOnError(err, "Failed _{\sqcup} to _{\sqcup} declare _{\sqcup} an _{\sqcup} exchange")
                                                                                                                    27
     q, err := chi.QueueDeclare("", false, true, false, false, nil)
                                                                                                                    29
     \texttt{failOnError(err, "Failed}_{\sqcup} \texttt{to}_{\sqcup} \texttt{declare}_{\sqcup} \texttt{a}_{\sqcup} \texttt{queue"})
                                                                                                                    30
     err = chi.QueueBind(q.Name, "key", "pExchange", false, nil)
                                                                                                                    31
     failOnError(err, "Failed_{\sqcup}to_{\sqcup}bind_{\sqcup}a_{\sqcup}queue")
                                                                                                                    32
     msgs, err := chi.Consume(q.Name, ""
                                                                                                                    33
     false, false, false, false, nil)
failOnError(err, "Failedutouregisteruauconsumer")
                                                                                                                    34
                                                                                                                    35
     for i := 0; i < 100; i++ {
           fakeLink := fmt.Sprintf("http://web%d.com", i)
                                                                                                                    37
           err := cho.Publish("pExchange", "key", false, false,
                                                                                                                    38
                amqp.Publishing{
                                                                                                                    39
                      ContentType: "text/plain",
                                                                                                                    40
                      Body:
                                        []byte(fakeLink),
                                                                                                                    41
                                                                                                                    42
           \verb|failOnError(err, "Failed| to| publish")|
                                                                                                                    43
```

```
fmt.Println("Published_job:" + fakeLink)
                                                                                              44
                                                                                              45
    err = cho.Publish("pExchange", "key", false, false,
                                                                                              46
         amqp.Publishing{
                                                                                              47
             ContentType: "text/plain",
                                                                                              48
                            [] byte("END"),
             Body:
                                                                                              49
        })
                                                                                              50
    failOnError(err, "Failed to publish")
    fmt.Println("Published_END")
                                                                                              52
    var wg sync.WaitGroup
    start := time.Now()
                                                                                              54
    wg.Add(1)
                                                                                              55
    go func() {
        for d := range msgs {
                                                                                              57
             s := string(d.Body)
                                                                                              58
             if s == "END" \{
                 err = chi.Cancel(d.ConsumerTag, false)
                                                                                              60
                 failOnError(err, "Failed_{\sqcup}to_{\sqcup}cancel_{\sqcup}a_{\sqcup}consumer")
                                                                                              61
             } else {
                  result := linkTest(s)
                                                                                              63
                 fmt.Println("goroutine", result)
                                                                                              64
                                                                                              65
             d.Ack(false)
                                                                                              66
                                                                                              67
        wg.Done()
                                                                                              68
    }()
                                                                                              69
                                                                                              70
    wg.Wait()
    duration := time.Since(start)
                                                                                              71
    {\tt fmt.Println("Time:_{\sqcup}",\ duration)}
                                                                                              72
                                                                                              73
func failOnError(err error, msg string) {
                                                                                              74
    if err != nil {
        log.Fatalf("%s:u%s", msg, err)
                                                                                              76
                                                                                              77
func linkTest(link string) string {
                                                                                              79
    time.Sleep(100 * time.Millisecond)
                                                                                              80
    if rand.Intn(2) == 1 {
                                                                                              81
         return link + ":Good"
                                                                                              82
    } else {
                                                                                              83
        return link + ":Bad"
                                                                                              84
                                                                                              85
}
                                                                                              86
```

Listing 1: Sequential Receive

```
package main
                                                                                           1
                                                                                           2
import (
                                                                                           3
    "fmt."
                                                                                           4
    "log"
    "math/rand"
    "runtime"
                                                                                           7
    "sync"
    "time"
                                                                                           10
    "github.com/streadway/amqp"
                                                                                           11
                                                                                           12
                                                                                            13
func main() {
                                                                                           14
    conn1, err := amqp.Dial("amqp://guest:guest@localhost:5672/")
                                                                                           15
```

```
failOnError(err, "FailedutouconnectutouRabbitMQ")
                                                                                                                    16
     defer conn1.Close()
                                                                                                                    17
     conn2, err := amqp.Dial("amqp://guest:guest@localhost:5672/")
                                                                                                                    18
     failOnError(err, "Failed_{\sqcup}to_{\sqcup}connect_{\sqcup}to_{\sqcup}RabbitMQ")
                                                                                                                    19
     defer conn2.Close()
                                                                                                                    20
     cho, err := conn1.Channel()
                                                                                                                    21
     \texttt{failOnError}(\texttt{err}, \ \texttt{"Failed}_{\sqcup} \texttt{to}_{\sqcup} \texttt{open}_{\sqcup} \texttt{a}_{\sqcup} \texttt{channel} \texttt{"})
                                                                                                                    22
     defer cho.Close()
                                                                                                                    23
     chi, err := conn2.Channel()
                                                                                                                    24
     \texttt{failOnError}(\texttt{err}, \ \texttt{"Failed}_{\sqcup} \texttt{to}_{\sqcup} \texttt{open}_{\sqcup} \texttt{a}_{\sqcup} \texttt{channel} \texttt{"})
                                                                                                                    25
     defer chi.Close()
                                                                                                                    26
     err = cho.ExchangeDeclare("pExchange", "direct",
                                                                                                                    27
     false, true, false, false, nil)
failOnError(err, "Failedutoudeclareuanuexchange")
q, err := chi.QueueDeclare("", false, true, false, false, nil)
                                                                                                                    28
                                                                                                                    29
                                                                                                                    30
      failOnError(err, "Failed_uto_udeclare_a_queue")
     err = chi.QueueBind(q.Name, "key", "pExchange", false, nil)
                                                                                                                    32
     \texttt{failOnError(err, "Failed}_{\sqcup} \texttt{to}_{\sqcup} \texttt{bind}_{\sqcup} \texttt{a}_{\sqcup} \texttt{queue"})
                                                                                                                    33
     msgs, err := chi.Consume(q.Name, "linkConsumer",
     false, false, false, false, nil) failOnError(err, "Failed_{\sqcup}to_{\sqcup}register_{\sqcup}a_{\sqcup}consumer")
                                                                                                                    35
                                                                                                                    36
     for i := 0; i < 100; i++ {
                                                                                                                    37
           fakeLink := fmt.Sprintf("http://web%d.com", i)
                                                                                                                    38
           err := cho.Publish("pExchange", "key", false, false,
                                                                                                                    39
                amqp.Publishing{
                                                                                                                    40
                      ContentType: "text/plain",
                                                                                                                    41
                                        []byte(fakeLink),
                                                                                                                    42
                })
                                                                                                                    43
           \verb|failOnError(err, "Failed| to| publish")|
                                                                                                                    44
           fmt.Println("Published_job:" + fakeLink)
                                                                                                                    45
                                                                                                                    46
     err = cho.Publish("pExchange", "key", false, false,
                                                                                                                    47
           amqp.Publishing{
                                                                                                                    48
                 ContentType: "text/plain",
                                                                                                                    49
                 Body:
                                  []byte("END"),
                                                                                                                    50
           })
                                                                                                                    51
     \texttt{failOnError(err, "Failed}_{\sqcup} \texttt{to}_{\sqcup} \texttt{publish"})
                                                                                                                    52
     fmt.Println("Published_END")
                                                                                                                    53
     var wg sync.WaitGroup
                                                                                                                    54
     start := time.Now()
                                                                                                                    55
     for i := 0; i < runtime.NumCPU(); i++ {</pre>
                                                                                                                    56
           wg.Add(1)
                                                                                                                    57
           go func(index int) {
                                                                                                                    58
                for d := range msgs \{
                                                                                                                    59
                      s := string(d.Body)
                                                                                                                    60
                      if s == "END" {
                                                                                                                    61
                            err = chi.Cancel("linkConsumer", false)
                                                                                                                    62
                            failOnError(err, "Failed_to_cancel_a_consumer")
                      } else {
                                                                                                                    64
                            result := linkTest(s)
                                                                                                                    65
                            fmt.Println("goroutine", index, result)
                                                                                                                    67
                      d.Ack(false)
                                                                                                                    68
                                                                                                                    69
                wg.Done()
                                                                                                                    70
           }(i)
                                                                                                                    71
     }
                                                                                                                    72
     wg.Wait()
                                                                                                                    73
     duration := time.Since(start)
                                                                                                                    74
     \verb|fmt.Println("Time: $\sqcup$", duration)|
                                                                                                                    75
}
                                                                                                                    76
```

```
func failOnError(err error, msg string) {
    if err != nil {
         log.Fatalf("%s: "%s", msg, err)
                                                                                                  80
func linkTest(link string) string {
                                                                                                  82
    time.Sleep(100 * time.Millisecond)
if rand.Intn(2) == 1 {
                                                                                                  83
         return link + ":Good"
    } else {
                                                                                                  86
         return link + ":Bad"
                                                                                                  87
                                                                                                  88
}
                                                                                                  89
```

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
Acceived: AAbb Lower: aabb
Received: EeFF Lower: eeff
Received: CcDd Lower: ccdd
Received: ggHh Lower: gghh
```

solution Upper

```
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
      1

      Sent: aaBB
      2

      Sent: aABb
      3

      Sent: AAbb
      4

      Sent: CcDd
      5

      Sent: EeFF
      6

      Sent: ggHh
      7
```