

Distributed Communication 9th practice

Li Jianhao
lijianhao288@hotmail.com

1 Basics

1.1 Exclusive

```
func (ch *Channel) QueueDeclare(name string, durable, autoDelete,
    exclusive, noWait bool, args Table) (Queue, error)
func (ch *Channel) Consume(queue, consumer string, autoAck, exclusive,
    noLocal, noWait bool, args Table) (<-chan Delivery, error)
```

1.2 Fanout exchange example

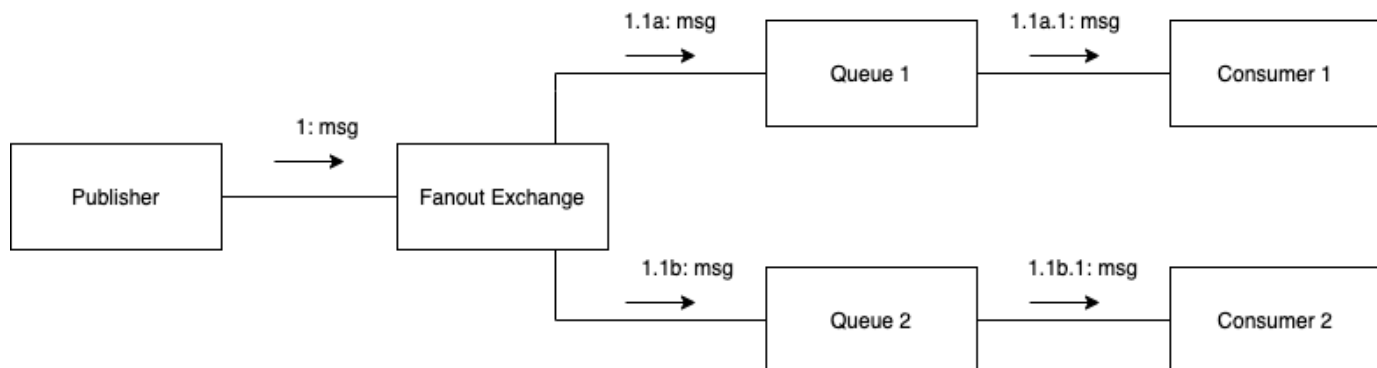


Figure 1: Fanout exchange example

Output:

Consumer1

Waiting for msgs	1
Received: msg	2

Consumer2

Waiting for msgs	1
Received: msg	2

Publisher

Sent: msg	1
<pre>package main</pre>	
<pre>import (</pre>	
<pre> "fmt"</pre>	
<pre> "github.com/streadway/amqp"</pre>	
<pre> "log"</pre>	
<pre>)</pre>	
<pre>func main() {</pre>	
<pre> //Connection</pre>	
<pre> conn, err := amqp.Dial("amqp://guest:guest@localhost:5672/")</pre>	
<pre> printErrorAndExit(err, "Failed to connect to RabbitMQ")</pre>	
<pre> defer conn.Close()</pre>	
<pre> //Channel</pre>	
<pre> ch, err := conn.Channel()</pre>	
<pre> printErrorAndExit(err, "Failed to open a channel")</pre>	
<pre> defer ch.Close()</pre>	
<pre> //Exchange</pre>	
<pre> err = ch.ExchangeDeclare(</pre>	
<pre> "fanoutExchange", // name</pre>	
<pre> "fanout", // type</pre>	
<pre> false, // durable</pre>	
<pre> true, // auto-deleted</pre>	
<pre> false, // internal</pre>	
<pre> false, // no-wait</pre>	
<pre> nil, // arguments</pre>	
<pre>)</pre>	
<pre> printErrorAndExit(err, "Failed to declare an exchange")</pre>	
<pre> //Publish Messages</pre>	
<pre> publishMsg(ch, "fanoutExchange", "anykey1", "msg")</pre>	
<pre>}</pre>	
<pre>func printErrorAndExit(err error, msg string) {</pre>	
<pre> if err != nil {</pre>	
<pre> log.Fatalln(msg, ":", err)</pre>	
<pre> }</pre>	
<pre>}</pre>	
<pre>func publishMsg(c *amqp.Channel, ex string, key string, msg string) {</pre>	
<pre> body := msg</pre>	
<pre> err := (*c).Publish(</pre>	
<pre> ex, // exchange</pre>	
<pre> key, // routing key</pre>	
<pre> false, // mandatory</pre>	
<pre> false, // immediate</pre>	
<pre> amqp.Publishing{</pre>	
<pre> ContentType: "text/plain",</pre>	
<pre> Body: []byte(body),</pre>	
<pre> })</pre>	
<pre> printErrorAndExit(err, "Failed to publish a message")</pre>	
<pre> fmt.Println("Sent:", body)</pre>	
<pre>}</pre>	

Listing 1: Fanout exchange example, Publisher

```

package main
1
2
import (
3
4     "fmt"
5     "github.com/streadway/amqp"
6     "log"
7
8 )
9
10 func main() {
11     //Connection
12     conn, err := amqp.Dial("amqp://guest:guest@localhost:5672/")
13     printErrorAndExit(err, "Failed to connect to RabbitMQ")
14     defer conn.Close()
15
16     //Channel
17     ch, err := conn.Channel()
18     printErrorAndExit(err, "Failed to open a channel")
19     defer ch.Close()
20
21     //Exchange
22     err = ch.ExchangeDeclare(
23         "fanoutExchange", // name
24         "fanout",          // type
25         false,              // durable
26         true,              // auto-deleted
27         false,             // internal
28         false,             // no-wait
29         nil,               // arguments
30     )
31     printErrorAndExit(err, "Failed to declare an exchange")
32
33     //Decalre and bind queue
34     q, err := ch.QueueDeclare(
35         "", // name, empty string let server generate id
36         false, // durable
37         true, // delete when unused
38         true, // exclusive
39         false, // no-wait
40         nil, // arguments
41     )
42     printErrorAndExit(err, "Failed to declare a queue")
43     err = ch.QueueBind(
44         q.Name, // queue name
45         "anykey2", // routing key
46         "fanoutExchange", // exchange
47         false,
48         nil)
49     printErrorAndExit(err, "Failed to bind a queue")
50
51     //Consume
52     msgs, err := ch.Consume(
53         q.Name, // queue
54         "", // consumer, empty string let server generate id
55         false, // auto-ack
56         false, // exclusive
57         false, // no-local
58         false, // no-wait
59         nil, // args
60     )
61     printErrorAndExit(err, "Failed to register a consumer")

```

go func() {	62
for d := range msgs {	63
bodyString := string(d.Body)	64
fmt.Println("Received:", bodyString)	65
d.Ack(false)	66
}	67
}()	68
	69
fmt.Println("Waiting for msgs")	70
forever := make(chan bool)	71
<-forever	72
}	73
	74
func printErrorAndExit(err error, msg string) {	75
if err != nil {	76
log.Fatalln(msg, ":", err)	77
}	78
}	79

Listing 2: Fanout exchange example, Consumer1

package main	1
	2
import (3
"fmt"	4
"github.com/streadway/amqp"	5
"log"	6
)	7
	8
func main() {	9
//Connection	10
conn, err := amqp.Dial("amqp://guest:guest@localhost:5672/")	11
printErrorAndExit(err, "Failed to connect to RabbitMQ")	12
defer conn.Close()	13
	14
//Channel	15
ch, err := conn.Channel()	16
printErrorAndExit(err, "Failed to open a channel")	17
defer ch.Close()	18
	19
//Exchange	20
err = ch.ExchangeDeclare(21
"fanoutExchange", // name	22
"fanout", // type	23
false, // durable	24
true, // auto-deleted	25
false, // internal	26
false, // no-wait	27
nil, // arguments	28
)	29
printErrorAndExit(err, "Failed to declare an exchange")	30
	31
//Decalre and bind queue	32
q, err := ch.QueueDeclare(33
"", // name,,empty string let server generate id	34
false, // durable	35
true, // delete when unused	36
true, // exclusive	37
false, // no-wait	38
nil, // arguments	39
)	40

```

41 printErrorAndExit(err, "Failed to declare a queue")
42 err = ch.QueueBind(
43     q.Name,          // queue name
44     "anykey3",        // routing key
45     "fanoutExchange", // exchange
46     false,
47     nil)
48 printErrorAndExit(err, "Failed to bind a queue")
49
50 //Consume
51 msgs, err := ch.Consume(
52     q.Name, // queue
53     "",     // consumer, empty string let server generate id
54     false,  // auto-ack
55     false,  // exclusive
56     false,  // no-local
57     false,  // no-wait
58     nil,    // args
59 )
60 printErrorAndExit(err, "Failed to register a consumer")
61
62 go func() {
63     for d := range msgs {
64         bodyString := string(d.Body)
65         fmt.Println("Received:", bodyString)
66         d.Ack(false)
67     }
68 }()
69
70 fmt.Println("Waiting for msgs")
71 forever := make(chan bool)
72 <-forever
73
74
75 func printErrorAndExit(err error, msg string) {
76     if err != nil {
77         log.Fatalln(msg, ":", err)
78     }
79 }

```

Listing 3: Fanout exchange example, Consumer2

1.3 Direct exchange example

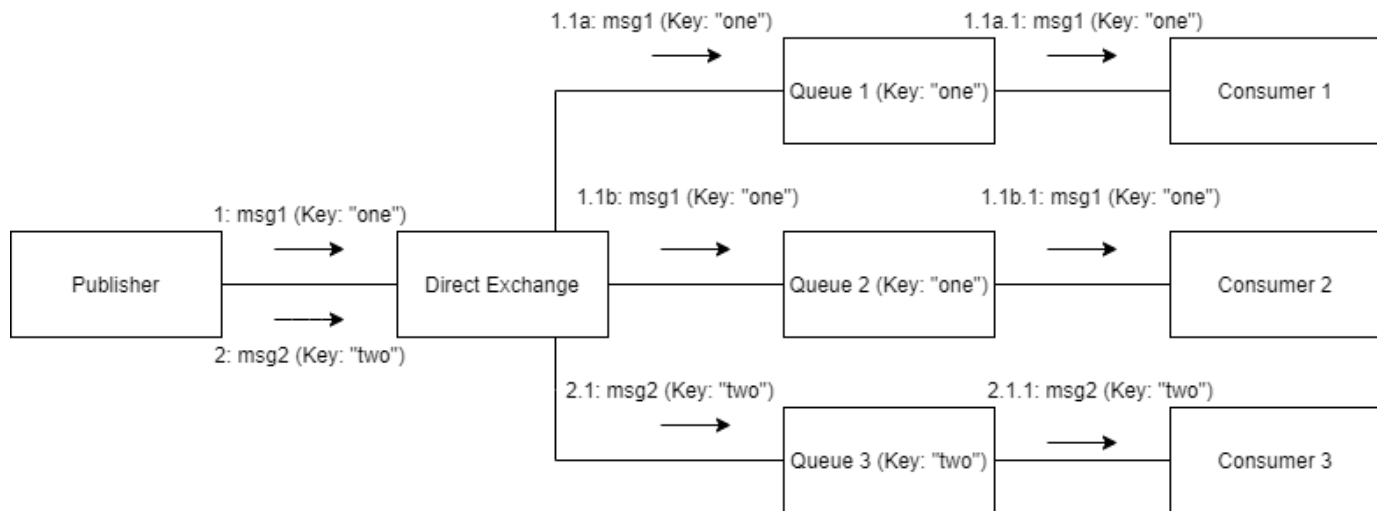


Figure 2: Direct exchange example

Output:

Consumer1

Waiting for msgs	1
Received: msg1	2

Consumer2

Waiting for msgs	1
Received: msg1	2

Consumer3

Waiting for msgs	1
Received: msg2	2

Publisher

Sent: msg1	1
Sent: msg2	2

package main	1
	2
import (3
"fmt"	4
"github.com/streadway/amqp"	5
"log"	6
)	7
	8

```

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

    //Channel
    ch, err := conn.Channel()
    printErrorAndExit(err, "Failed to open a channel")
    defer ch.Close()

    //Exchange
    err = ch.ExchangeDeclare(
        "directExchange", // name
        "direct",          // type
        false,              // durable
        true,               // auto-deleted
        false,              // internal
        false,              // no-wait
        nil,                // arguments
    )
    printErrorAndExit(err, "Failed to declare an exchange")

    //Publish Messages
    publishMsg(ch, "directExchange", "one", "msg1")
    publishMsg(ch, "directExchange", "two", "msg2")
}

func printErrorAndExit(err error, msg string) {
    if err != nil {
        log.Fatalln(msg, ":", err)
    }
}

func publishMsg(c *amqp.Channel, ex string, key string, msg string) {
    body := msg
    err := (c).Publish(
        ex, // exchange
        key, // routing key
        false, // mandatory
        false, // immediate
        amqp.Publishing{
            ContentType: "text/plain",
            Body: []byte(body),
        })
    printErrorAndExit(err, "Failed to publish a message")
    fmt.Println("Sent:", body)
}

```

Listing 4: Direct exchange example, Publisher

```

package main
1
2
import (
3
4     "fmt"
5     "github.com/streadway/amqp"
6     "log"
7
8 )
9
func main() {
10     //Connection

```

```

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

//Channel
ch, err := conn.Channel()
printErrorAndExit(err, "Failed to open a channel")
defer ch.Close()

//Exchange
err = ch.ExchangeDeclare(
    "directExchange", // name
    "direct",          // type
    false,             // durable
    true,              // auto-deleted
    false,             // internal
    false,             // no-wait
    nil,               // arguments
)
printErrorAndExit(err, "Failed to declare an exchange")

//Decalre and bind queue
q, err := ch.QueueDeclare(
    "", // name, empty string let server generate id
    false, // durable
    true, // delete when unused
    true, // exclusive
    false, // no-wait
    nil, // arguments
)
printErrorAndExit(err, "Failed to declare a queue")
err = ch.QueueBind(
    q.Name, // queue name
    "one", // routing key
    "directExchange", // exchange
    false,
    nil)
printErrorAndExit(err, "Failed to bind a queue")

//Consume
msgs, err := ch.Consume(
    q.Name, // queue
    "", // consumer, empty string let server generate id
    false, // auto-ack
    false, // exclusive
    false, // no-local
    false, // no-wait
    nil, // args
)
printErrorAndExit(err, "Failed to register a consumer")

go func() {
    for d := range msgs {
        bodyString := string(d.Body)
        fmt.Println("Received:", bodyString)
        d.Ack(false)
    }
}()

fmt.Println("Waiting for msgs")
forever := make(chan bool)

```


<pre> <-forever } func printErrorAndExit(err error, msg string) { if err != nil { log.Fatalln(msg, ":", err) } } </pre>	<pre> 72 73 74 75 76 77 78 79 </pre>
---	--------------------------------------

Listing 5: Direct exchange example, Consumer1

<pre> package main import ("fmt" "github.com/streadway/amqp" "log") func main() { //Connection conn, err := amqp.Dial("amqp://guest:guest@localhost:5672/") printErrorAndExit(err, "Failed to connect to RabbitMQ") defer conn.Close() //Channel ch, err := conn.Channel() printErrorAndExit(err, "Failed to open a channel") defer ch.Close() //Exchange err = ch.ExchangeDeclare("directExchange", // name "direct", // type false, // durable true, // auto-deleted false, // internal false, // no-wait nil, // arguments) printErrorAndExit(err, "Failed to declare an exchange") //Decalre and bind queue q, err := ch.QueueDeclare("", // name,,empty string let server generate id false, // durable true, // delete when unused true, // exclusive false, // no-wait nil, // arguments) printErrorAndExit(err, "Failed to declare a queue") err = ch.QueueBind(q.Name, // queue name "one", // routing key "directExchange", // exchange false, nil) printErrorAndExit(err, "Failed to bind a queue") //Consume </pre>	<pre> 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 </pre>
---	---

```

msgs, err := ch.Consume(
    q.Name, // queue
    "",     // consumer, empty string let server generate id
    false,  // auto-ack
    false,  // exclusive
    false,  // no-local
    false,  // no-wait
    nil,    // args
)
printErrorAndExit(err, "Failed to register a consumer")

go func() {
    for d := range msgs {
        bodyString := string(d.Body)
        fmt.Println("Received:", bodyString)
        d.Ack(false)
    }
}()

fmt.Println("Waiting for msgs")
forever := make(chan bool)
<-forever
}

func printErrorAndExit(err error, msg string) {
    if err != nil {
        log.Fatalln(msg, ":", err)
    }
}

```

Listing 6: Direct exchange example, Consumer2

```

package main
1
2
import (
3
4     "fmt"
5     "github.com/streadway/amqp"
6     "log"
7
8
9
10    func main() {
11        //Connection
12        conn, err := amqp.Dial("amqp://guest:guest@localhost:5672/")
13        printErrorAndExit(err, "Failed to connect to RabbitMQ")
14        defer conn.Close()
15
16        //Channel
17        ch, err := conn.Channel()
18        printErrorAndExit(err, "Failed to open a channel")
19        defer ch.Close()
20
21        //Exchange
22        err = ch.ExchangeDeclare(
23            "directExchange", // name
24            "direct",        // type
25            false,            // durable
26            true,            // auto-deleted
27            false,          // internal
28            false,          // no-wait
29            nil,            // arguments
30        )
31    }
32

```

```

printErrorAndExit(err, "Failed to declare an exchange")
30
//Decalre and bind queue
31
q, err := ch.QueueDeclare(
32
    "", // name,,empty string let server generate id
33
    false, // durable
34
    true, // delete when unused
35
    true, // exclusive
36
    false, // no-wait
37
    nil, // arguments
38
)
39
printErrorAndExit(err, "Failed to declare a queue")
40
err = ch.QueueBind(
41
    q.Name, // queue name
42
    "two", // routing key
43
    "directExchange", // exchange
44
    false,
45
    nil)
46
printErrorAndExit(err, "Failed to bind a queue")
47
//Consume
48
msgs, err := ch.Consume(
49
    q.Name, // queue
50
    "", // consumer,empty string let server generate id
51
    false, // auto-ack
52
    false, // exclusive
53
    false, // no-local
54
    false, // no-wait
55
    nil, // args
56
)
57
printErrorAndExit(err, "Failed to register a consumer")
58
go func() {
59
    for d := range msgs {
60
        bodyString := string(d.Body)
61
        fmt.Println("Received:", bodyString)
62
        d.Ack(false)
63
    }
64
}()
65
fmt.Println("Waiting for msgs")
66
forever := make(chan bool)
67
<-forever
68
}
69
func printErrorAndExit(err error, msg string) {
70
    if err != nil {
71
        log.Fatalln(msg, ":", err)
72
    }
73
}
74
75
76
77
78
79

```

Listing 7: Direct exchange example, Consumer3

1.4 Shared queue example

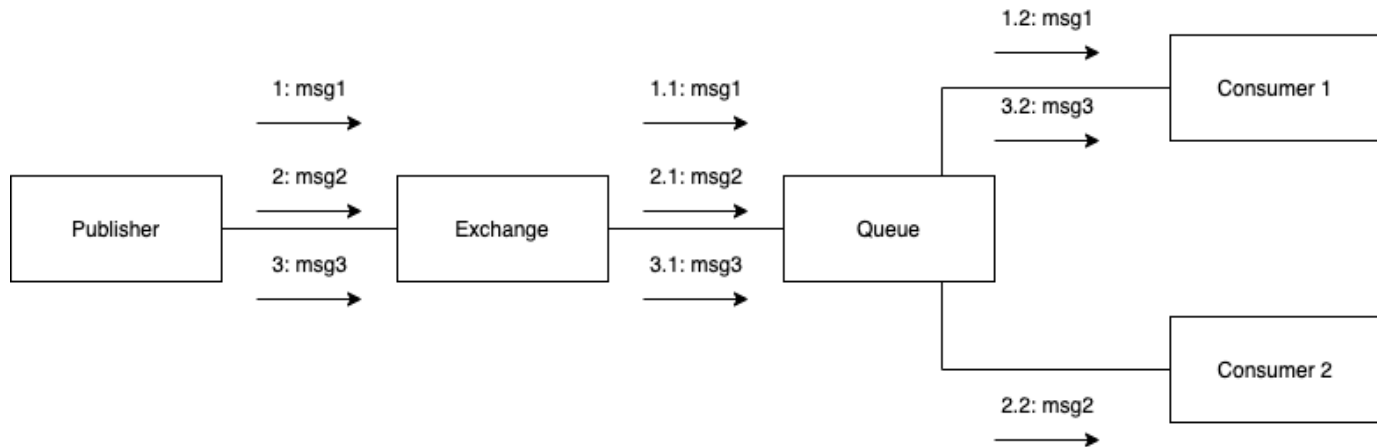


Figure 3: Shared queue example

Output:

Consumer1

Waiting for msgs	1
Received: msg1	2
Received: msg3	3

Consumer2

Waiting for msgs	1
Received: msg2	2

Publisher

Sent: msg1	1
Sent: msg2	2
Sent: msg3	3

```
package main                                1
                                              2
import (                                     3
    "fmt"                                    4
    "github.com/streadway/amqp"             5
    "log"                                    6
)                                              7
                                              8
func main() {                               9
    //Connection                             10
    conn, err := amqp.Dial("amqp://guest:guest@localhost:5672/") 11
    printErrorAndExit(err, "Failed to connect to RabbitMQ") 12
    defer conn.Close()                      13
                                              14
```

```

//Channel
ch, err := conn.Channel()
printErrorAndExit(err, "Failed to open a channel")
defer ch.Close()

//Exchange
err = ch.ExchangeDeclare(
    "sharedQExchange", // name
    "direct",           // type
    false,              // durable
    true,               // auto-deleted
    false,              // internal
    false,              // no-wait
    nil,                // arguments
)
printErrorAndExit(err, "Failed to declare an exchange")

//Publish Messages
publishMsg(ch, "sharedQExchange", "one", "msg1")
publishMsg(ch, "sharedQExchange", "one", "msg2")
publishMsg(ch, "sharedQExchange", "one", "msg3")
}

func printErrorAndExit(err error, msg string) {
    if err != nil {
        log.Fatalln(msg, ":", err)
    }
}

func publishMsg(c *amqp.Channel, ex string, key string, msg string) {
    body := msg
    err := (c).Publish(
        ex, // exchange
        key, // routing key
        false, // mandatory
        false, // immediate
        amqp.Publishing{
            ContentType: "text/plain",
            Body: []byte(body),
        })
    printErrorAndExit(err, "Failed to publish a message")
    fmt.Println("Sent:", body)
}

```

Listing 8: Shared queue example, Publisher

```

package main

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

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

    //Channel

```

```

16 ch, err := conn.Channel()
17 printErrorAndExit(err, "Failed to open a channel")
18 defer ch.Close()
19
20 //Exchange
21 err = ch.ExchangeDeclare(
22     "sharedQExchange", // name
23     "direct",           // type
24     false,              // durable
25     true,               // auto-deleted
26     false,              // internal
27     false,              // no-wait
28     nil,                // arguments
29 )
30 printErrorAndExit(err, "Failed to declare an exchange")
31
32 //Declare and bind queue
33 q, err := ch.QueueDeclare(
34     "SharedQueue", // name, empty string let server generate id
35     false,         // durable
36     true,          // delete when unused
37     false,        // exclusive
38     false,        // no-wait
39     nil,          // arguments
40 )
41 printErrorAndExit(err, "Failed to declare a queue")
42 err = ch.QueueBind(
43     q.Name,           // queue name
44     "one",            // routing key
45     "sharedQExchange", // exchange
46     false,
47     nil)
48 printErrorAndExit(err, "Failed to bind a queue")
49
50 //Consume
51 msgs, err := ch.Consume(
52     q.Name, // queue
53     "",    // consumer, empty string let server generate id
54     false, // auto-ack
55     false, // exclusive
56     false, // no-local
57     false, // no-wait
58     nil,   // args
59 )
60 printErrorAndExit(err, "Failed to register a consumer")
61
62 go func() {
63     for d := range msgs {
64         bodyString := string(d.Body)
65         fmt.Println("Received:", bodyString)
66         d.Ack(false)
67     }
68 }()
69
70 fmt.Println("Waiting for msgs")
71 forever := make(chan bool)
72 <-forever
73
74 }
75
76 func printErrorAndExit(err error, msg string) {
77     if err != nil {

```

<code>log.Fatalln(msg, ":", err)</code>	77
<code>}</code>	78
<code>}</code>	79

Listing 9: Shared queue example, Consumer1

<code>package main</code>	1
<code>import (</code>	2
<code> "fmt"</code>	3
<code> "github.com/streadway/amqp"</code>	4
<code> "log"</code>	5
<code>)</code>	6
<code></code>	7
<code></code>	8
<code>func main() {</code>	9
<code> //Connection</code>	10
<code> conn, err := amqp.Dial("amqp://guest:guest@localhost:5672/")</code>	11
<code> printErrorAndExit(err, "Failed to connect to RabbitMQ")</code>	12
<code> defer conn.Close()</code>	13
<code></code>	14
<code> //Channel</code>	15
<code> ch, err := conn.Channel()</code>	16
<code> printErrorAndExit(err, "Failed to open a channel")</code>	17
<code> defer ch.Close()</code>	18
<code></code>	19
<code> //Exchange</code>	20
<code> err = ch.ExchangeDeclare(</code>	21
<code> "sharedQExchange", // name</code>	22
<code> "direct", // type</code>	23
<code> false, // durable</code>	24
<code> true, // auto-deleted</code>	25
<code> false, // internal</code>	26
<code> false, // no-wait</code>	27
<code> nil, // arguments</code>	28
<code>)</code>	29
<code> printErrorAndExit(err, "Failed to declare an exchange")</code>	30
<code></code>	31
<code> //Decalre and bind queue</code>	32
<code> q, err := ch.QueueDeclare(</code>	33
<code> "SharedQueue", // name,,empty string let server generate id</code>	34
<code> false, // durable</code>	35
<code> true, // delete when unused</code>	36
<code> false, // exclusive</code>	37
<code> false, // no-wait</code>	38
<code> nil, // arguments</code>	39
<code>)</code>	40
<code> printErrorAndExit(err, "Failed to declare a queue")</code>	41
<code> err = ch.QueueBind(</code>	42
<code> q.Name, // queue name</code>	43
<code> "one", // routing key</code>	44
<code> "sharedQExchange", // exchange</code>	45
<code> false,</code>	46
<code> nil)</code>	47
<code> printErrorAndExit(err, "Failed to bind a queue")</code>	48
<code></code>	49
<code> //Consume</code>	50
<code> msgs, err := ch.Consume(</code>	51
<code> q.Name, // queue</code>	52
<code> "", // consumer,empty string let server generate id</code>	53
<code> false, // auto-ack</code>	54
<code> false, // exclusive</code>	55

```

        false, // no-local
        false, // no-wait
        nil,    // args
    )
    printErrorAndExit(err, "Failed to register a consumer")

    go func() {
        for d := range msgs {
            bodyString := string(d.Body)
            fmt.Println("Received:", bodyString)
            d.Ack(false)
        }
    }()

    fmt.Println("Waiting for msgs")
    forever := make(chan bool)
    <-forever
}

func printErrorAndExit(err error, msg string) {
    if err != nil {
        log.Fatalln(msg, ":", err)
    }
}

```

Listing 10: Shared queue example, Consumer2

2 Practice

2.1 p1

Create 1 publisher and 2 consumers. They use a direct exchange with the name "EX".

1. The publisher sends two private messages to "EX" (one for consumer1, another for consumer2). Each consumer will receive their own private message. For each received message, the consumer print out "Received private message: " and the message. (Hint(The hint will not appear during the exam): Each consumer consumes the private messages from a private queue. The queues are bound to the "EX" with different keys)
2. The publisher sends two broadcast messages to "EX". All the consumers will receive all the broadcast messages. For each received message, the consumer print out "Received broadcast message: " and the message.

(Hint(The hint will not appear during the exam): Each consumer consumes the private messages from a private queue. The queues are bound to the "EX" with the same key)

3. The publisher sends 5 tasks messages to "EX". The consumers receive the tasks in a balanced round-robin way. For each received message, the consumer print out "Received task message:" and the message.

(Hint(The hint will not appear during the exam): Each consumer consumes the task messages from a shared queue. The name of the shared queue is known by two consumers)

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

Consumer1

Waiting for msgs	1
Received private message: private msg1	2
Received broadcast message: broadcast1	3
Received task message: task1	4
Received broadcast message: broadcast2	5
Received task message: task3	6
Received task message: task5	7

Consumer2

Waiting for msgs	1
Received private message: private msg2	2
Received broadcast message: broadcast1	3
Received broadcast message: broadcast2	4
Received task message: task2	5
Received task message: task4	6

Publisher

Sent: private msg1	1
Sent: private msg2	2
Sent: broadcast1	3
Sent: broadcast2	4
Sent: task1	5
Sent: task2	6
Sent: task3	7
Sent: task4	8
Sent: task5	9