# 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

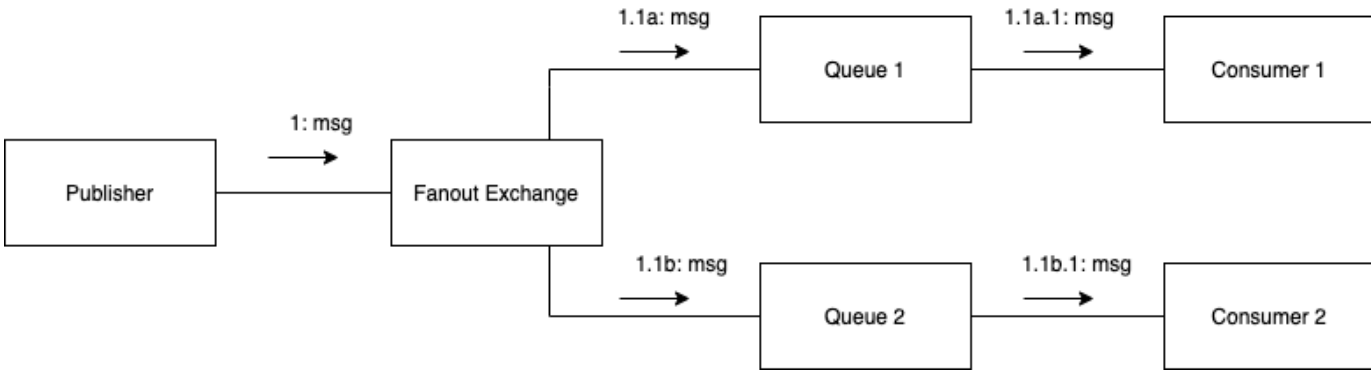


Figure 1: Fanout exchange example

Output:
Consumer1

```
Waiting for msgs                                                    1
Received: msg                                                       2
```

Consumer2

```
Waiting for msgs                                                    1
Received: msg                                                       2
```

## Publisher

```
Sent:  msg
```

```
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(
        "fanoutExchange", // name
        "fanout",         // type
        false,            // durable
        true,             // auto-deleted
        false,            // internal
        false,            // no-wait
        nil,              // arguments
    )
    printErrorAndExit(err, "Failed to declare an exchange")

    //Publish Messages
    publishMsg(ch, "fanoutExchange", "anykey1", "msg")
}

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

    //Exchange
    err = ch.ExchangeDeclare(
        "fanoutExchange", // name
        "fanout",         // 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
        "anykey2",        // routing key
        "fanoutExchange", // 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)
    <-forever
}

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

Listing 2: Fanout exchange example, Consumer1

```
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(
        "fanoutExchange", // name
        "fanout",         // 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
        "anykey3",        // routing key
        "fanoutExchange", // 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)
    <-forever
}

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

Listing 3: Fanout exchange example, Consumer2

## 1.3 Direct exchange example

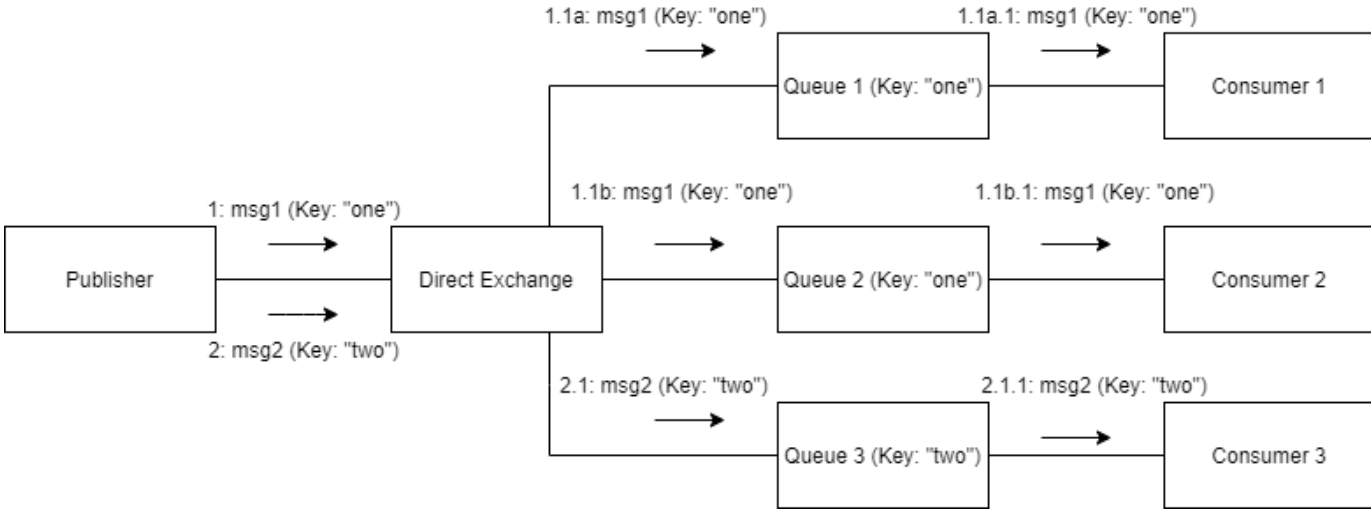


Figure 2: Direct exchange example

Output:
Consumer1

```
Waiting for msgs
Received: msg1
```

Consumer2

```
Waiting for msgs
Received: msg1
```

Consumer3

```
Waiting for msgs
Received: msg2
```

Publisher

```
Sent:  msg1
Sent:  msg2
```

```
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")

    //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

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
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 5: Direct exchange example, Consumer1

```
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
```

```
    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

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
        "two",            // 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)
    <-forever
}

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

Listing 7: Direct exchange example, Consumer3

## 1.4 Shared queue example

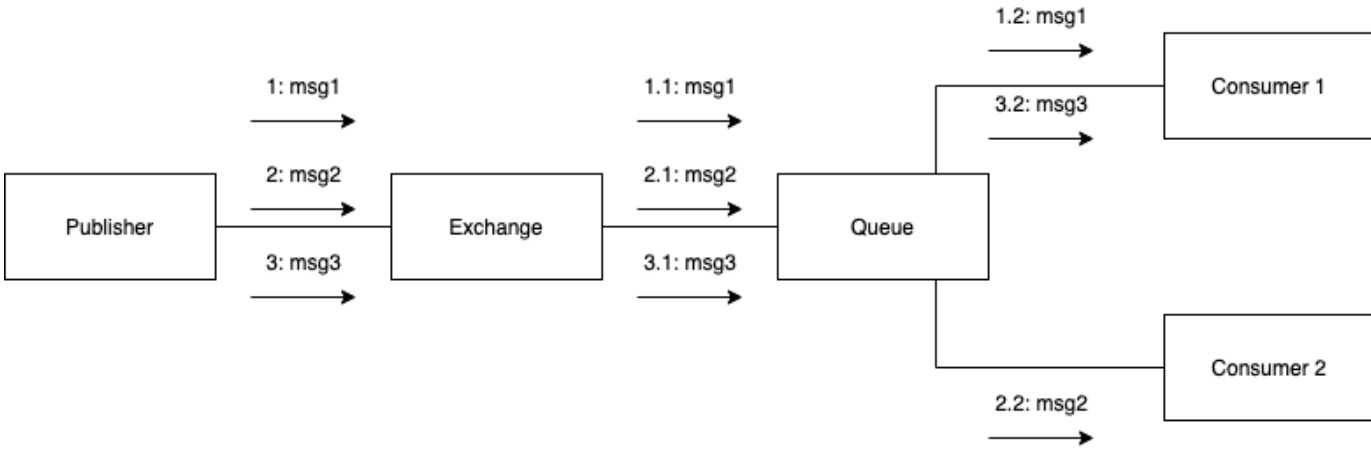


Figure 3: Shared queue example

Output:
Consumer1

```
Waiting for msgs
Received: msg1
Received: msg3
```

Consumer2

```
Waiting for msgs
Received: msg2
```

Publisher

```
Sent:  msg1
Sent:  msg2
Sent:  msg3
```

```
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(
        "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
```

```
    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")

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

func printErrorAndExit(err error, msg string) {
    if err != nil {
```

```
        log.Fatalln(msg, ":", err)
    }
}
```

Listing 9: Shared queue example, Consumer1

```
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(
        "sharedQExchange", // 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(
        "SharedQueue", // name,,empty string let server generate id
        false,         // durable
        true,          // delete when unused
        false,         // exclusive
        false,         // no-wait
        nil,           // arguments
    )
    printErrorAndExit(err, "Failed to declare a queue")
    err = ch.QueueBind(
        q.Name,            // queue name
        "one",             // routing key
        "sharedQExchange", // 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)
    <-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
Received private message: private msg1
Received broadcast message: broadcast1
Received task message: task1
Received broadcast message: broadcast2
Received task message: task3
Received task message: task5
```

Consumer2

```
Waiting for msgs
Received private message: private msg2
Received broadcast message: broadcast1
Received broadcast message: broadcast2
Received task message: task2
Received task message: task4
```

Publisher

```
Sent:  private msg1
Sent:  private msg2
Sent:  broadcast1
Sent:  broadcast2
Sent:  task1
Sent:  task2
Sent:  task3
Sent:  task4
Sent:  task5
```