RabbitMq 调研

目录

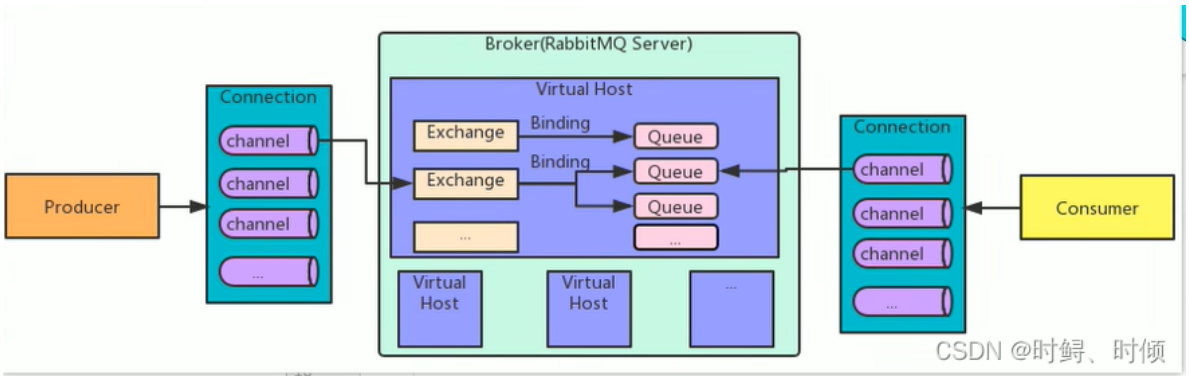
序

参考资料：

- golang操作RabbitMq的使用_golang mq-CSDN博客
- Golang RabbitMQ实现的延时队列_go rabbitmq 延迟队列_UPUP小亮的博客-CSDN博客
- https://github.com/jeffcail/go-rabbitmq-tutorial/tree/main

1、架构

producer -> **exchange -> queue** -> consumer



2、mac安装

```
#安装
brew update
brew install rabbitmq
#查看
brew info rabbitmq
#启动
brew services start rabbitmq
#停止
brew services stop rabbitmq
----------------------------------------
#管理界面
```

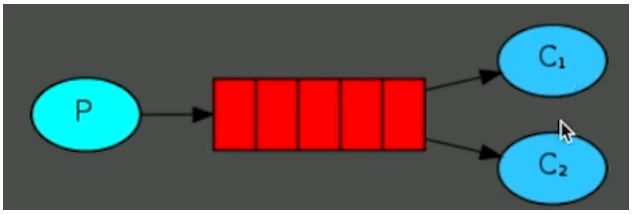
```
http://127.0.0.1:15672/#/
用户|密码:  guest | guest
```

3、YBB技术架构解耦方案

场景1：异步处理消息

1个生产 + 多消费模式

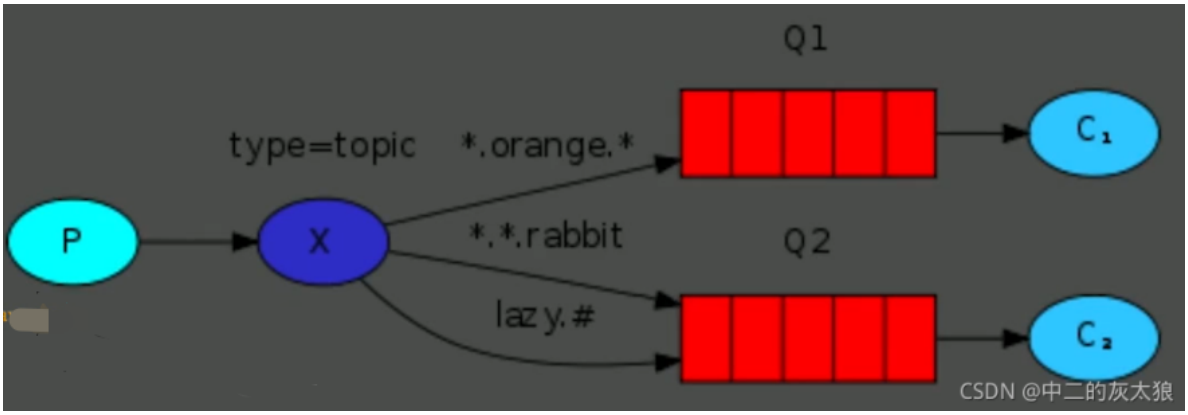
工作队列

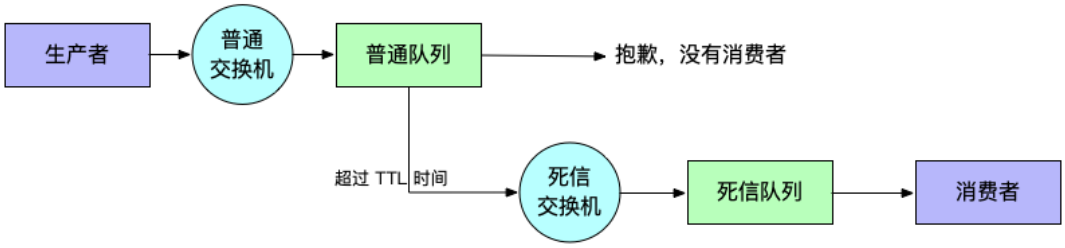


场景2：延迟任务场景

如xx分钟未支付关闭订单，卡卷过期等

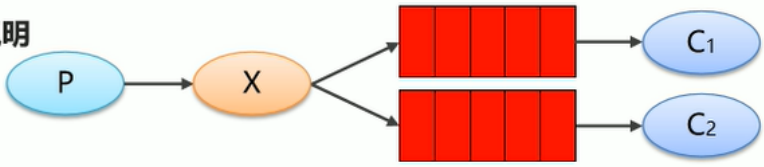
死信队列 + 延迟队列





4.2 Pub/Sub 订阅模式

1. 模式说明



在订阅模型中，多了一个 Exchange 角色，而且过程略有变化：

- P：生产者，也就是要发送消息的程序，但是不再发送到队列中，而是发给X（交换机）
- C：消费者，消息的接收者，会一直等待消息到来
- Queue：消息队列，接收消息、缓存消息
- Exchange：交换机（X）。一方面，接收生产者发送的消息。另一方面，知道如何处理消息，例如递交给某个特别队列、递交给所有队列、或是将消息丢弃。到底如何操作，取决于Exchange的类型。Exchange有常见以下3种类型：
  - Fanout：广播，将消息交给所有绑定到交换机的队列
  - Direct：定向，把消息交给符合指定routing key 的队列
  - Topic：通配符，把消息交给符合routing pattern（路由模式） 的队列

Exchange（交换机）只负责转发消息，不具备存储消息的能力，因此如果没有任何队列与 Exchange 绑定，或者没有符合路由规则的队列，那么消息会丢失！

4、demo代码

消息的结构体

```go
type Publishing struct {
    // Application or exchange specific fields,
    // the headers exchange will inspect this field.
    Headers Table

    // Properties
    ContentType     string    // MIME content type
    ContentEncoding string    // MIME content encoding
    DeliveryMode    uint8     // Transient (0 or 1) or Persistent (2)
    Priority        uint8     // 0 to 9
    CorrelationId   string    // correlation identifier
    ReplyTo         string    // address to to reply to (ex: RPC)
    Expiration      string    // message expiration spec
    MessageId       string    // message identifier
    Timestamp       time.Time // message timestamp
    Type            string    // message type name
    UserId          string    // creating user id - ex: "guest"
    AppId           string    // creating application id

    // The application specific payload of the message
    Body []byte
}
```

4.1、延迟队列+工作队列

生成逻辑

```go
package test

import (
    "context"
    "fmt"
    "github.com/rabbitmq/amqp091-go"
    "log"
    "rabbit/tools"
```

```
    "testing"
)

func failOnError(err error, msg string) {
    if err != nil {
        log.Printf("%s: %s", msg, err)
    }
}

func TestSendMsg(t *testing.T) {
    conn, err := amqp091.Dial("amqp://guest:guest@localhost:5672/")
    failOnError(err, "Failed to connect to RabbitMQ")
    defer conn.Close()

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

    //参数:
    //1.queue:队列名称
    //2.durable: 是否持久化，当mq重启之后，还在
    //3.exclusive: 参数有两个意思 a)是否独占即只能有一个消费者监听这个队列 b)当connection关闭时，是否删除队列
    //4.autoDelete: 是否自动删除。当没有Consumer时，自动删除掉
    //5.argument: 参数。配置如何删除
    q, err := ch.QueueDeclare(tools.BizQueue, false, false, false, false, nil)
    failOnError(err, "Failed to declare q queue")

    //发送普通消息
    for i := 0; i < 10; i++ {
        msg := fmt.Sprintf("send msg %v", i)
        err = ch.PublishWithContext(context.Background(), "", q.Name, false, false, amqp091.Publishing{
            ContentType:  "test/plain",
            DeliveryMode: amqp091.Persistent,
            Body:         []byte(msg),
        })
        failOnError(err, "Failed to publish a message")
        log.Printf(" [x] Sent %s,%v", msg, i)
    }

    //发送延迟消息
    for i := 0; i < 5; i++ {
        body := fmt.Sprintf("这是一个延迟队列的消息2023:%v", i)
        var exp string
        if i < 2 {
            exp = `10000`
        } else {
            exp = `30000`
        }
        // 将消息发送到延迟队列上
        err = ch.PublishWithContext(
            context.Background(),
            "",               // exchange 这里为空则不选择 exchange
            tools.DelayQueue, // routing key
            false,            // mandatory
            false,            // immediate
            amqp091.Publishing{
                ContentType: "text/plain",
                Body:        []byte(body),
                Expiration:  exp, // 设置10秒的过期时间
            })
        failOnError(err, "Failed to publish a delay message")
    }
}

```

消费逻辑

Go | 收起

```go
package main

import (
    "bytes"
    "github.com/rabbitmq/amqp091-go"
    "log"
    "rabbit/tools"
    "time"
)

func main() {
    conn, err := amqp091.Dial("amqp://guest:guest@localhost:5672/")
    tools.FailOnError(err, "Failed to connect to RabbitMQ")
    defer conn.Close()

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

    /*- 工作队列 -*/

    q, err := ch.QueueDeclare(tools.BizQueue, false, false, false, false, nil)
    tools.FailOnError(err, "Failed to declare a queue")

    msgs, err := ch.Consume(q.Name, "", true, false, false, false, nil)
    tools.FailOnError(err, "Failed to register a consumer")

    var forever chan struct{}
    //消费工作队列
    go func() {
        for d := range msgs {
            log.Printf("工作队列 [x]: %s", d.Body)
            dotCount := bytes.Count(d.Body, []byte("."))
            t := time.Duration(dotCount)
            time.Sleep(t * time.Second)
            log.Printf("Done")
        }
    }()

    /*- 延迟队列 -*/
    // 声明一个主要使用的 exchange
    err = ch.ExchangeDeclare(
        tools.DelayExchange, // name
        "fanout",            // type广播形式，绑定的都会推送消息
        false,               // durable
        false,               // auto-deleted
        false,               // internal
        false,               // no-wait
        nil,                 // arguments
    )
    tools.FailOnError(err, "Failed to declare an exchange")

    //声明一个普通队列，来处理死信消息
    _, err = ch.QueueDeclare(tools.DelayHandlerQueue, false, false, false, false, nil)
    tools.FailOnError(err, "Failed to declare a queue")

    /**
     * 注意,这里是重点!!!!!
     * 声明一个延时队列，ß我们的延时消息就是要发送到这里
     */
    _, err = ch.QueueDeclare(
```

```
        tools.DelayQueue, // name
        false,            // durable
        false,            // delete when unused
        false,            // exclusive
        false,            // no-wait
        amqp091.Table{
            // 当消息过期时把消息发送到 logs 这个 exchange
            "x-dead-letter-exchange": tools.DelayExchange,
        }, // arguments
    )
    tools.FailOnError(err, "fail to declare a queue")

    err = ch.QueueBind(
        tools.DelayHandlerQueue, // queue name, 这里指的是 test_logs
        "",                      // routing key
        tools.DelayExchange,     // exchange
        false,
        nil,
    )
    tools.FailOnError(err, "Failed to bind a queue")

    // 这里监听的是 test_logs
    msg2, err := ch.Consume(
        tools.DelayHandlerQueue, // queue name, 这里指的是 test_logs
        "",                      // consumer
        true,                    // auto-ack
        false,                   // exclusive
        false,                   // no-local
        false,                   // no-wait
        nil,                     // args
    )
    tools.FailOnError(err, "Failed to consume a queue")

    go func() {
        for d := range msg2 {
            log.Printf("接收延迟消息 [y] %s", d.Body)
        }
    }()

    log.Printf(" [*] Waiting for messages. To exit press CTRL+C")
    <-forever
}

```

代码执行情况

消费者1

```
2023/10/16 10:18:23  [*] Waiting for messages. To exit press CTRL+C
2023/10/16 10:19:39 工作队列 [x]: send msg 0
2023/10/16 10:19:39 Done
2023/10/16 10:19:39 工作队列 [x]: send msg 2
2023/10/16 10:19:39 Done
2023/10/16 10:19:39 工作队列 [x]: send msg 4
2023/10/16 10:19:39 Done
2023/10/16 10:19:39 工作队列 [x]: send msg 6
2023/10/16 10:19:39 Done
2023/10/16 10:19:39 工作队列 [x]: send msg 8
2023/10/16 10:19:39 Done
2023/10/16 10:19:49 接收延迟消息 [y] 这是一个延迟队列的消息2023:0
2023/10/16 10:20:09 接收延迟消息 [y] 这是一个延迟队列的消息2023:2
2023/10/16 10:20:09 接收延迟消息 [y] 这是一个延迟队列的消息2023:4
```

消费者2

```
2023/10/16 10:19:20  [*] Waiting for messages. To exit press CTRL+C
2023/10/16 10:19:39 工作队列 [x]: send msg 1
2023/10/16 10:19:39 Done
2023/10/16 10:19:39 工作队列 [x]: send msg 3
2023/10/16 10:19:39 Done
2023/10/16 10:19:39 工作队列 [x]: send msg 5
2023/10/16 10:19:39 Done
2023/10/16 10:19:39 工作队列 [x]: send msg 7
2023/10/16 10:19:39 Done
2023/10/16 10:19:39 工作队列 [x]: send msg 9
2023/10/16 10:19:39 Done
2023/10/16 10:19:49 接收延迟消息 [y] 这是一个延迟队列的消息2023:1
2023/10/16 10:20:09 接收延迟消息 [y] 这是一个延迟队列的消息2023:3
```

结论

1、延迟队列采用的fifo的方式

```
-> a 10s
-> b 5s
那么10s后，弹出a,b

-> a 10s
-> b 30s
那么10s后，弹出a
那么30s后，弹出b
```