



APUSIC
固若长城
睿比世界

開發手冊

金蝶Apusic分布式消息队列V2.0.6_for_rabbitmq

版權所有 © 深圳市金蝶天燕雲計算股份有限公司2026。保留所有權利。

版權聲明

本文件所涉及的軟件著作權、版權等知識產權已依法進行了註冊，由金蝶天燕雲計算股份有限公司合法擁有。受《中華人民共和國著作權法》《計算機軟件保護條例》《知識產權保護條例》和相關國際版權條約、法律、法規以及其它知識產權法律和條約的保護。未經授權許可，不得非法使用。

免責聲明

本文件包含的版權資訊由金蝶天燕雲計算股份有限公司合法擁有，受法律的保護，金蝶天燕雲計算股份有限公司對本文件可能涉及到的非金蝶天燕雲計算股份有限公司的資訊不承擔任何責任。在法律允許的範圍內，您可以查閱並僅能夠在《中華人民共和國著作權法》規定的合法範圍內複製和打印本文件。任何單位和個人未經金蝶天燕雲計算股份有限公司書面授權許可，不得使用、修改、再發佈本文件的任何部分和內容，否則將被視為侵權，金蝶天燕雲計算股份有限公司有依法追究其責任的權利。

本文件如有更新，不另行通知。對本文件中的問題您可向金蝶天燕雲計算股份有限公司告知或查詢。未經本公司明確授予的任何權利均予保留。

商標聲明

 是深圳市金蝶天燕雲計算股份有限公司向中華人民共和國國家商標局申請註冊的註冊商標，註冊商標專用權由金蝶天燕合法擁有，受法律保護。未經金蝶天燕的書面許可，任何單位及個人不得以任何方式或理由對該商標的任何部分進行使用、複製、修改、傳播、抄錄或與其它產品捆綁使用銷售。凡侵犯金蝶天燕商標權的，金蝶天燕將依法追究其法律責任。本文件提及的其他所有商標或註冊商標，由各自的所有人擁有。

目錄

- 1 前言
 - 1.1 適用對象
 - 1.2 相關文件
 - 1.3 技術支援
- 2 開發準備
 - 2.1 連線參數
 - 2.2 客戶端依賴
 - 2.2.1 Java
 - 2.2.2 Python
 - 2.2.3 Go
 - 2.2.4 Node.js
- 3 Java 開發範例
 - 3.1 生產者
 - 3.2 消費者
 - 3.3 事務訊息
 - 3.4 Confirm 模式
- 4 Python 開發範例
 - 4.1 生產者
 - 4.2 消費者
- 5 Go 開發範例
 - 5.1 生產者
 - 5.2 消費者
- 6 Node.js 開發範例
 - 6.1 生產者
 - 6.2 消費者
- 7 進階功能
 - 7.1 延遲訊息
 - 7.2 優先級佇列
 - 7.3 批次消費
- 8 最佳實踐
 - 8.1 連線管理
 - 8.2 訊息可靠性

- 8.3 效能最佳化
- 8.4 異常處理

1 前言

本文檔為金蝶Apusic分散式訊息佇列for RabbitMQ（Apusic Distributed Message Queue，簡稱：ADMQ for RabbitMQ）的開發使用說明，幫助使用者快速學習如何使用金蝶Apusic分散式訊息佇列for RabbitMQ進行開發。

1.1 適用對象

本文檔適用於IT資訊化業務負責人、研發經理、軟體專案經理、軟體架構師、維運工程師。

1.2 相關文件

了解更多ADMQ for RabbitMQ產品相關的資訊，請參閱以下ADMQ for RabbitMQ產品手冊文件集：

序號	手冊文件	說明
1	金蝶Apusic分散式訊息佇列for RabbitMQ 快速使用手冊	簡單介紹如何快速上手使用ADMQ for RabbitMQ。
2	金蝶Apusic分散式訊息佇列for RabbitMQ 安裝手冊	詳細介紹如何在各作業系統上安裝ADMQ for RabbitMQ，以及ADMQ for RabbitMQ服務啟停等操作。
3	金蝶Apusic分散式訊息佇列for RabbitMQ 訊息引擎使用者手冊	詳細介紹ADMQ for RabbitMQ訊息引擎相關功能的使用、設定、管理及配套工具的使用方法。
4	金蝶Apusic分散式訊息佇列for RabbitMQ 管控台使用者手冊	詳細介紹ADMQ for RabbitMQ管控台相關功能的使用和操作說明。
5	金蝶Apusic分散式訊息佇列for RabbitMQ 開發手冊	詳細介紹基於各開發語言進行ADMQ for RabbitMQ客戶端應用開發的說明。
6	金蝶Apusic分散式訊息佇列for RabbitMQ 遷移手冊	詳細介紹從RabbitMQ遷移到ADMQ for RabbitMQ的說明。
7	金蝶Apusic分散式訊息佇列for RabbitMQ 維運手冊	詳細介紹ADMQ for RabbitMQ的監控、維運、安全強化等維運說明。
8	金蝶Apusic分散式訊息佇列for RabbitMQ 效能最佳化手冊	詳細介紹ADMQ for RabbitMQ效能調校的說明。

1.3 技術支援

ADMQ for RabbitMQ產品提供全面的技術支援服務，您可以透過以下方式獲得技術支援：

- 網址: www.apusic.com
- 電話: 400-855-5800
- 郵箱: support@apusic.com
- 金蝶雲社群: <https://vip.kingdee.com/?productId=73&productLineId=14&lang=zh-CN>

您在取得技術支援時，請提供如下資訊：

1. 您的姓名
2. 公司資訊與聯絡方式
3. 作業系統及其版本
4. 產品版本號
5. 出現異常及錯誤的日誌、截圖等詳細資訊

2 開發準備

2.1 連線參數

參數	說明	範例
host	伺服器位址	localhost
port	埠號	5672 (AMQP) / 5671 (AMQP over SSL)
username	使用者名稱	guest
password	密碼	guest
virtualHost	虛擬主機	/

2.2 客戶端依賴

2.2.1 Java

```
<dependency>
  <groupId>com.rabbitmq</groupId>
  <artifactId>amqp-client</artifactId>
  <version>5.21.0</version>
</dependency>
```

2.2.2 Python

```
pip install pika
```

2.2.3 Go

```
go get github.com/rabbitmq/amqp091-go
```

2.2.4 Node.js

```
npm install amqpplib
```

3 Java 開發範例

3.1 生產者

```
import com.rabbitmq.client.*;

public class Producer {
    private static final String EXCHANGE_NAME = "demo.exchange";
    private static final String ROUTING_KEY = "demo.key";

    public static void main(String[] args) throws Exception {
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("localhost");
        factory.setPort(5672);
        factory.setUsername("guest");
        factory.setPassword("guest");
        factory.setVirtualHost("/");

        try (Connection connection = factory.newConnection();
            Channel channel = connection.createChannel()) {

            // 声明交换机
            channel.exchangeDeclare(EXCHANGE_NAME, "direct", true);

            // 发送消息
            String message = "Hello ADMQ for RabbitMQ!";
            channel.basicPublish(EXCHANGE_NAME, ROUTING_KEY,
                MessageProperties.PERSISTENT_TEXT_PLAIN,
                message.getBytes("UTF-8"));

            System.out.println("Sent: " + message);
        }
    }
}
```

3.2 消費者

```
import com.rabbitmq.client.*;

public class Consumer {
    private static final String QUEUE_NAME = "demo.queue";
    private static final String EXCHANGE_NAME = "demo.exchange";
    private static final String ROUTING_KEY = "demo.key";

    public static void main(String[] args) throws Exception {
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("localhost");
        factory.setPort(5672);
        factory.setUsername("guest");
        factory.setPassword("guest");
        factory.setVirtualHost("/");

        Connection connection = factory.newConnection();
        Channel channel = connection.createChannel();

        // 声明队列
        channel.queueDeclare(QUEUE_NAME, true, false, false, null);

        // 绑定队列到交换机
        channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, ROUTING_KEY);

        // 创建消费者
        DeliverCallback deliverCallback = (consumerTag, delivery) ->
        {
            String message = new String(delivery.getBody(), "UTF-8");

            System.out.println("Received: " + message);

            // 手动确认
        }
    }
}
```

```
channel.basicAck(delivery.getEnvelope().getDeliveryTag(), false);
    };

    // 消费消息 (关闭自动确认)
    channel.basicConsume(QueueName, false, deliverCallback,
        consumerTag -> {});
    }
}
```

3.3 事務訊息

```
channel.txSelect(); // 开启事务

try {
    channel.basicPublish(EXCHANGE, ROUTING_KEY, null,
        message1.getBytes());
    channel.basicPublish(EXCHANGE, ROUTING_KEY, null,
        message2.getBytes());
    channel.txCommit(); // 提交事务
} catch (Exception e) {
    channel.txRollback(); // 回滚事务
}
```

3.4 Confirm 模式

```
channel.confirmSelect(); // 开启 Confirm 模式

channel.basicPublish(EXCHANGE, ROUTING_KEY, null,
    message.getBytes());

// 同步等待确认
if (channel.waitForConfirms()) {
    System.out.println("Message confirmed");
}
```

// 异步确认

```
channel.addConfirmListener(new ConfirmListener() {  
    @Override  
    public void handleAck(long deliveryTag, boolean multiple) {  
        System.out.println("Message ack: " + deliveryTag);  
    }  
  
    @Override  
    public void handleNack(long deliveryTag, boolean multiple) {  
        System.out.println("Message nack: " + deliveryTag);  
    }  
});
```

4 Python 開發範例

4.1 生產者

```
import pika

connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost', port=5672,

credentials=pika.PlainCredentials('guest', 'guest')))
channel = connection.channel()

channel.exchange_declare(exchange='demo.exchange',
exchange_type='direct', durable=True)

channel.basic_publish(exchange='demo.exchange',
                      routing_key='demo.key',
                      body='Hello ADMQ for RabbitMQ!',

properties=pika.BasicProperties(delivery_mode=2))

print("Sent message")
connection.close()
```

4.2 消費者

```
import pika

connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost', port=5672,

credentials=pika.PlainCredentials('guest', 'guest')))
channel = connection.channel()
```

```
channel.queue_declare(queue='demo.queue', durable=True)
channel.queue_bind(queue='demo.queue', exchange='demo.exchange',
routing_key='demo.key')

def callback(ch, method, properties, body):
    print(f"Received {body.decode()}")
    ch.basic_ack(delivery_tag=method.delivery_tag)

channel.basic_consume(queue='demo.queue',
on_message_callback=callback, auto_ack=False)
print('Waiting for messages...')
channel.start_consuming()
```

5 Go 開發範例

5.1 生產者

```
package main

import (
    "fmt"
    "github.com/rabbitmq/amqp091-go"
    "log"
)

func main() {
    conn, err := amqp091.Dial("amqp://guest:guest@localhost:5672/")
    if err != nil {
        log.Fatal(err)
    }
    defer conn.Close()

    ch, err := conn.Channel()
    if err != nil {
        log.Fatal(err)
    }
    defer ch.Close()

    err = ch.ExchangeDeclare("demo.exchange", "direct", true, false,
false, false, nil)
    if err != nil {
        log.Fatal(err)
    }

    body := "Hello ADMQ for RabbitMQ!"
    err = ch.Publish("demo.exchange", "demo.key", false, false,
        amqp091.Publishing{
```

```

        DeliveryMode: amqp091.Persistent,
        ContentType:  "text/plain",
        Body:         []byte(body),
    })
    if err != nil {
        log.Fatal(err)
    }

    fmt.Println("Sent:", body)
}

```

5.2 消費者

```

package main

import (
    "fmt"
    "github.com/rabbitmq/amqp091-go"
    "log"
)

func main() {
    conn, err := amqp091.Dial("amqp://guest:guest@localhost:5672/")
    if err != nil {
        log.Fatal(err)
    }
    defer conn.Close()

    ch, err := conn.Channel()
    if err != nil {
        log.Fatal(err)
    }
    defer ch.Close()

    q, err := ch.QueueDeclare("demo.queue", true, false, false,

```

```
false, nil)
    if err != nil {
        log.Fatal(err)
    }

    err = ch.QueueBind(q.Name, "demo.key", "demo.exchange", false,
nil)
    if err != nil {
        log.Fatal(err)
    }

    msgs, err := ch.Consume(q.Name, "", false, false, false, false,
nil)
    if err != nil {
        log.Fatal(err)
    }

    fmt.Println("Waiting for messages...")
    for msg := range msgs {
        fmt.Printf("Received: %s\n", msg.Body)
        msg.Ack(false)
    }
}
```

6 Node.js 開發範例

6.1 生產者

```
const amqp = require('amqplib');

async function produce() {
  const conn = await
amqp.connect('amqp://guest:guest@localhost:5672');
  const ch = await conn.createChannel();

  await ch.assertExchange('demo.exchange', 'direct', { durable:
true });

  const msg = 'Hello ADMQ for RabbitMQ!';
  ch.publish('demo.exchange', 'demo.key', Buffer.from(msg), {
persistent: true });

  console.log('Sent:', msg);

  await ch.close();
  await conn.close();
}

produce().catch(console.error);
```

6.2 消費者

```
const amqp = require('amqplib');

async function consume() {
  const conn = await
amqp.connect('amqp://guest:guest@localhost:5672');
```

```
const ch = await conn.createChannel();

await ch.assertQueue('demo.queue', { durable: true });
await ch.bindQueue('demo.queue', 'demo.exchange', 'demo.key');

console.log('Waiting for messages...');

ch.consume('demo.queue', (msg) => {
    console.log('Received:', msg.content.toString());
    ch.ack(msg);
}, { noAck: false });
}

consume().catch(console.error);
```

7 進階功能

7.1 延遲訊息

使用死信交換器 + TTL 實現延遲佇列：

```
// 创建死信交换机和队列
channel.exchangeDeclare("dlx.exchange", "direct", true);
channel.queueDeclare("dlx.queue", true, false, false, null);
channel.queueBind("dlx.queue", "dlx.exchange", "dlx.key");

// 创建延迟队列 (设置 TTL 和死信参数)
Map<String, Object> args = new HashMap<>();
args.put("x-message-ttl", 60000); // 60秒延迟
args.put("x-dead-letter-exchange", "dlx.exchange");
args.put("x-dead-letter-routing-key", "dlx.key");

channel.queueDeclare("delay.queue", true, false, false, args);

// 发送消息到延迟队列
channel.basicPublish("", "delay.queue", null, message.getBytes());
```

7.2 優先級佇列

```
Map<String, Object> args = new HashMap<>();
args.put("x-max-priority", 10);
channel.queueDeclare("priority.queue", true, false, false, args);

// 发送高优先级消息
AMQP.BasicProperties props = new AMQP.BasicProperties.Builder()
    .priority(9)
    .build();
```

```
channel.basicPublish("", "priority.queue", props,  
message.getBytes());
```

7.3 批次消費

```
// 设置 QoS, 每次预取 100 条消息  
channel.basicQos(100);  
  
// 批量确认  
channel.basicAck(deliveryTag, true); // multiple=true 确认所有小于等于  
该 deliveryTag 的消息
```

8 最佳實踐

8.1 連線管理

1. 使用連線池：避免頻繁建立和關閉連線
2. 合理設定心跳：防止網路異常導致連線假死
3. 異常處理：捕獲異常並進行重新連線

8.2 訊息可靠性

1. 使用手動確認：確保訊息被正確處理後才確認
2. 訊息持久化：重要訊息設定 `delivery_mode=2`
3. 佇列持久化：重要佇列設定 `durable=true`
4. 使用 Confirm 模式：確保訊息成功發送到交換器

8.3 效能最佳化

1. 批次操作：批次傳送訊息、批次確認
2. 合理設定 QoS：避免消費者積壓過多訊息
3. 使用多頻道：單個連線上建立多個頻道並行處理
4. 選擇合適的佇列類型：Quorum Queue 適合高可用場景，Stream Queue 適合高吞吐場景

8.4 異常處理

```
// 连接断开的回调
factory.setExceptionHandler(new DefaultExceptionHandler() {
    @Override
    public void handleConnectionRecoveryException(Connection conn,
Throwable exception) {
        // 记录日志并进行重连
    }
});

// 自动恢复
factory.setAutomaticRecoveryEnabled(true);
factory.setNetworkRecoveryInterval(5000); // 5秒重试间隔
factory.setTopologyRecoveryEnabled(true); // 自动恢复拓扑结构
```

全國統一服務熱線
4008-555-800

金蝶天燕雲計算股份有限公司（簡稱“金蝶天燕”）成立於2000年，前身為“金蝶中間件公司”，是金蝶集團旗下基礎軟件平台提供商，雲計算國家標準制定企業，國家信創產業核心軟件企業。金蝶天燕是國家863重點研發計劃與核高基重大專項承接企業，也是“兩網一站四庫十二金”國家重點工程的基礎平台提供商，產品廣泛應用於政府、軍工、金融、能源等關鍵行業，累計服務客戶總數超過10萬家。



雲計算國家標準制定企業
金蝶集團旗下基礎軟件企業
資訊技術應用創新核心企業
官網：www.apusic.com

