



APUSIC
固若长城
睿比世界

Development Manual

金蝶Apusic分布式消息队列V2.0.6_for_rabbitmq

Copyright Notice

All intellectual property rights including software copyrights involved in this document have been legally registered and duly owned by Kingdee Apusic Cloud Computing Co., Ltd. Such rights are protected by Copyright Law of the People's Republic of China, Regulations on Protection of Computer Software, Regulations on Intellectual Property Protection, relevant international copyright conventions, laws, regulations and other intellectual property legislations. Unauthorized use is strictly prohibited.

Disclaimer

All copyright information contained herein is legally owned by Kingdee Apusic Cloud Computing Co., Ltd. and protected by applicable laws. The Company shall bear no liability for any information unrelated to Kingdee Apusic contained in this document. You may view, copy and print this document only within the scope permitted by Copyright Law of the People's Republic of China. No entity or individual may use, modify or republish any part or content of this document without written authorization from Kingdee Apusic Cloud Computing Co., Ltd. Any violation shall constitute infringement, and the Company reserves the right to pursue legal liabilities accordingly.

Updates to this document shall not be notified separately. You may consult the Company regarding issues related to this document. All rights not expressly granted by the Company are reserved.

Trademark Notice

 is a registered trademark filed with the National Intellectual Property Administration of PRC by Kingdee Apusic Cloud Computing Co., Ltd. Shenzhen. Exclusive rights to the registered trademark are legally owned by Kingdee Apusic and protected by law. Without written consent of Kingdee Apusic, no entity or individual shall use, copy, modify, disseminate, transcribe, or bundle and sell any part of this trademark with other products in any manner or for any reason. Kingdee Apusic shall pursue legal liabilities against any infringement of its trademark rights. All other trademarks or registered trademarks mentioned in this document belong to their respective owners.

Contents

- 1 Preface
 - 1.1 Target Audience
 - 1.2 Related Documents
 - 1.3 Technical Support
- 2 Development Preparation
 - 2.1 Connection Parameters
 - 2.2 Client Dependencies
 - 2.2.1 Java
 - 2.2.2 Python
 - 2.2.3 Go
 - 2.2.4 Node.js
- 3 Java Development Example
 - 3.1 Producer
 - 3.2 Consumer
 - 3.3 Transactional Messages
 - 3.4 Confirm Mode
- 4 Python Development Example
 - 4.1 Producer
 - 4.2 Consumer
- 5 Go Development Example
 - 5.1 Producer
 - 5.2 Consumer
- 6 Node.js Development Example
 - 6.1 Producer
 - 6.2 Consumer
- 7 Advanced Features
 - 7.1 Delayed Messages
 - 7.2 Priority Queues
 - 7.3 Batch Consumption
- 8 Best Practices
 - 8.1 Connection Management
 - 8.2 Message Reliability

- 8.3 Performance Optimization
- 8.4 Exception Handling

1 Preface

This document is the development guide for Kingdee Apusic Distributed Message Queue for RabbitMQ (Apusic Distributed Message Queue,简称: ADMQ for RabbitMQ), helping users quickly learn how to develop with Kingdee Apusic Distributed Message Queue for RabbitMQ.

1.1 Target Audience

This document is intended for IT information business leaders, R&D managers, software project managers, software architects, and operations engineers.

1.2 Related Documents

For more information about ADMQ for RabbitMQ products, please refer to the following ADMQ for RabbitMQ product documentation set:

No.	Document	Description
1	Kingdee Apusic Distributed Message Queue for RabbitMQ Quick Start Guide	Briefly introduces how to quickly get started with ADMQ for RabbitMQ.
2	Kingdee Apusic Distributed Message Queue for RabbitMQ Installation Guide	Detailed instructions on installing ADMQ for RabbitMQ on various operating systems, as well as ADMQ for RabbitMQ service start/stop operations.
3	Kingdee Apusic Distributed Message Queue for RabbitMQ Message Engine User Guide	Detailed description of the usage, configuration, management, and supporting tools of the ADMQ for RabbitMQ message engine.
4	Kingdee Apusic Distributed Message Queue for RabbitMQ Console User Guide	Detailed description of the usage and operation of the ADMQ for RabbitMQ management console.
5	Kingdee Apusic Distributed Message Queue for RabbitMQ Development Guide	Detailed instructions on developing ADMQ for RabbitMQ client applications using various programming languages.
6	Kingdee Apusic Distributed Message Queue for RabbitMQ Migration Guide	Detailed instructions on migrating from RabbitMQ to ADMQ for RabbitMQ.
7	Kingdee Apusic Distributed Message Queue for RabbitMQ Operations Guide	Detailed description of ADMQ for RabbitMQ monitoring, operations, security hardening, and other maintenance tasks.

8	Kingdee Apusic Distributed Message Queue for RabbitMQ Performance Optimization Guide	Detailed description of ADMQ for RabbitMQ performance tuning.
---	--	---

1.3 Technical Support

ADMQ for RabbitMQ provides comprehensive technical support services. You can obtain technical support through the following channels:

- Website: www.apusic.com
- Phone: 400-855-5800
- Email: support@apusic.com
- Kingdee Cloud Community: <https://vip.kingdee.com/?productId=73&productLineId=14&lang=zh-CN>

When seeking technical support, please provide the following information:

1. Your name
2. Company information and contact details
3. Operating system and version
4. Product version number
5. Detailed information such as logs and screenshots of exceptions and errors

2 Development Preparation

2.1 Connection Parameters

Parameter	Description	Example
host	Server address	localhost
port	Port	5672 (AMQP) / 5671 (AMQP over SSL)
username	Username	guest
password	Password	guest
virtualHost	Virtual host	/

2.2 Client Dependencies

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 amqplib
```

3 Java Development Example

3.1 Producer

```
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 Consumer

```
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 Transactional Messages

```
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 Mode

```
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 Development Example

4.1 Producer

```
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 Consumer

```
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 Development Example

5.1 Producer

```
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 Consumer

```

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 Development Example

6.1 Producer

```
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 Consumer

```
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 Advanced Features

7.1 Delayed Messages

Use dead letter exchange + TTL to implement delayed queues:

```
// 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 Priority Queues

```
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 Batch Consumption

```
// 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 Best Practices

8.1 Connection Management

1. Use connection pool: Avoid frequently creating and closing connections
2. Set reasonable heartbeat: Prevent connection staleness caused by network anomalies
3. Exception handling: Catch exceptions and perform reconnection

8.2 Message Reliability

1. Use manual acknowledgments: Ensure messages are acknowledged only after correct processing
2. Message persistence: Set delivery_mode=2 for important messages
3. Queue persistence: Set durable=true for important queues
4. Use Confirm mode: Ensure messages are successfully sent to the exchange

8.3 Performance Optimization

1. Batch operations: Batch send messages and batch acknowledgments
2. Set reasonable QoS: Prevent consumers from accumulating too many messages
3. Use multiple channels: Create multiple channels on a single connection for concurrent processing
4. Choose the right queue type: Quorum Queue for high availability, Stream Queue for high throughput

8.4 Exception Handling

```
// 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
```

UNIFIED SERVICE HOTLINE

4008-555-800

Kingdee Apusic Cloud Computing Co., Ltd. (referred to as “Kingdee Apusic”) was founded in 2000. Its predecessor was “Kingdee Middleware Company.” It is a basic software platform provider under Kingdee Group, a national standards-setting enterprise for cloud computing, and a core software enterprise in the national IT application innovation industry. Kingdee Apusic is an enterprise undertaking the National 863 Key R&D Program and the National Major Science and Technology Project for Core Electronic Components, High-end General-purpose Chips and Fundamental Software, and is also a basic platform provider for the national key projects of “Two Networks, One Website, Four Databases, and Twelve Golden Projects.” Its products are widely used in key industries such as government, military, finance, and energy, and it has cumulatively served more than 100,000 customers.



National standard setter for cloud computing
Basic software enterprise under Kingdee Group
Core enterprise for IT application innovation
Website: www.apusic.com

