Apusic 金蝶天燕

生成时间：2026-07-13

开发手册

金蝶Apusic分布式消息队列V2.0.6_for_rabbitmq

版权声明

本文档所涉及的软件著作权、版权等知识产权已依法进行了注册，由金蝶天燕云计算股份有限公司合法拥有。受《中华人民共和国著作权法》《计算机软件保护条例》《知识产权保护条例》和相关国际版权条约、法律、法规以及其它知识产权法律和条约的保护。未经授权许可，不得非法使用。

免责声明

本文档包含的版权信息由金蝶天燕云计算股份有限公司合法拥有，受法律的保护，金蝶天燕云计算股份有限公司对本文档可能涉及到的非金蝶天燕云计算股份有限公司的信息不承担任何责任。在法律允许的范围内，您可以查阅并仅能够在《中华人民共和国著作权法》规定的合法范围内复制和打印本文档。任何单位和个人未经金蝶天燕云计算股份有限公司书面授权许可，不得使用、修改、再发布本文档的任何部分和内容，否则将被视为侵权，金蝶天燕云计算股份有限公司有依法追究其责任的权利。

本文档如有更新，不另行通知。对本文档中的问题您可向金蝶天燕云计算股份有限公司告知或查询。未经本公司明确授予的任何权利均予保留。

商标声明

Apusic是深圳市金蝶天燕云计算股份有限公司向中华人民共和国国家商标局申请注册的注册商标，注册商标专用权由金蝶天燕合法拥有，受法律保护。未经金蝶天燕的书面许可，任何单位及个人不得以任何方式或理由对该商标的任何部分进行使用、复制、修改、传播、抄录或与其它产品捆绑使用销售。凡侵犯金蝶天燕商标权的，金蝶天燕将依法追究其法律责任。本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

目录

1 前言

本文档为金蝶Apusic分布式消息队列for RabbitMQ（Apusic Distributed Message Queue，简称：ADMQ for RabbitMQ）的开发使用说明，帮助用户快速学习如何使用金蝶Apusic分布式消息队列for RabbitMQ进行开发。

1.1 适用对象

本文档适用于IT信息化业务负责人、研发经理、软件项目经理、软件架构师、运维工程师。

1.2 相关文档

了解更多ADMQ for RabbitMQ产品相关的信息，请参阅以下ADMQ for RabbitMQ产品手册文档集：

序号	手册文档	说明
1	金蝶Apusic分布式消息队列for RabbitMQ 快速使用手册	简单介绍了如何快速上手使用ADMQ for RabbitMQ 。
2	金蝶Apusic分布式消息队列for RabbitMQ 安装手册	详细介绍如何在各操作系统上安装ADMQ for RabbitMQ，以及ADMQ for RabbitMQ服务启停等操作。
3	金蝶Apusic分布式消息队列for RabbitMQ 消息引擎用户手册	详细介绍 ADMQ for RabbitMQ 消息引擎相关功能的使用、配置、管理及配套工具的使用方法。
4	金蝶Apusic分布式消息队列for RabbitMQ 管控台用户手册	详细介绍ADMQ for RabbitMQ管控台相关功能的使用和操作说明。
5	金蝶Apusic分布式消息队列for RabbitMQ 开发手册	详细介绍基于各开发语言进行ADMQ for RabbitMQ客户端应用开发的说明。
6	金蝶Apusic分布式消息队列for RabbitMQ 迁移手册	详细介绍从RabbitMQ迁移到ADMQ for RabbitMQ的说明。
7	金蝶Apusic分布式消息队列for RabbitMQ 运维手册	详细介绍ADMQ for RabbitMQ的监控、运维、安全加固等运维说明。
8	金蝶Apusic分布式消息队列for RabbitMQ 性能优化手册	详细介绍ADMQ for RabbitMQ性能调优的说明。

1.3 技术支持

ADMQ for RabbitMQ产品提供全面的技术支持服务，您可以通过以下方式获得技术支持：

- 网址：www.apusic.com
- 电话：400-855-5800
- 邮箱：support@apusic.com
- 金蝶云社区：https://vip.kingdee.com/?productId=73&productLineId=14&lang=zh-CN

您在取得技术支持时，请提供如下信息：

1. 您的姓名
2. 公司信息与联系方式
3. 操作系统及其版本
4. 产品版本号
5. 出现异常及错误的日志、截图等详细信息

2 开发准备

2.1 连接参数

参数	说明	示例
host	服务器地址	localhost
port	端口	5672（AMQP）/ 5671（AMQP over SSL）
username	用户名	guest
password	密码	guest
virtualHost	虚拟主机	/

2.2 客户端依赖

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 开发示例

3.1 生产者

```
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 消费者

```
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(QUEUE_NAME, false, deliverCallback,
consumerTag -> {});
    }
}
```

3.3 事务消息

```
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 开发示例

4.1 生产者

```
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 消费者

```
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 开发示例

5.1 生产者

```
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 消费者

```
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 开发示例

6.1 生产者

```
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 消费者

```
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 高级特性

7.1 延迟消息

使用死信交换机 + TTL 实现延迟队列：

```
// 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 优先级队列

```
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 批量消费

```
// 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 最佳实践

8.1 连接管理

1. **使用连接池**：避免频繁创建和关闭连接
2. **合理设置心跳**：防止网络异常导致连接假死
3. **异常处理**：捕获异常并进行重连

8.2 消息可靠性

1. **使用手动确认**：确保消息被正确处理后确认
2. **消息持久化**：重要消息设置 delivery_mode=2
3. **队列持久化**：重要队列设置 durable=true
4. **使用 Confirm 模式**：确保消息成功发送到交换机

8.3 性能优化

1. **批量操作**：批量发送消息、批量确认
2. **合理设置 QoS**：避免消费者积压过多消息
3. **使用多通道**：单个连接上创建多个通道并发处理
4. **选择合适的队列类型**：Quorum Queue 适合高可用场景，Stream Queue 适合高吞吐场景

8.4 异常处理

```
// 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
```

全国统一服务热线

4008-555-800

金蝶天燕云计算股份有限公司(简称“金蝶天燕云”)成立于2000年，前身为“金蝶中间件公司”，是金蝶集团旗下新一代软件基础云平台服务商，云计算国家标准制定企业，国家信创产业核心软件企业。金蝶天燕是国家863重点研发计划与核高基重大专项承接企业，也是“两网一站四库十二金”国家重点工程的基础平台提供商，产品广泛应用于政府、军工、金融、能源等关键行业，累计服务客户总数超过10万家。

Apusic 金蝶天燕

云计算国家标准制定企业

金蝶集团旗下基础软件企业

信息技术应用创新核心企业

官网：www.apusic.com