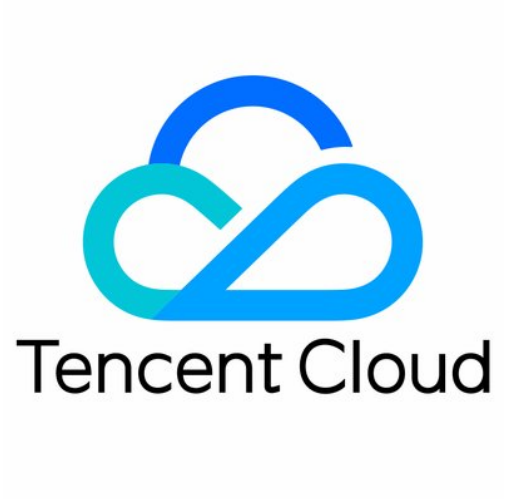
# TDMQ for RabbitMQ

# Development Guide

## Product Documentation



# Contents

# Development Guide
# Delayed Message

Last updated：2024-10-18 17:38:12

This document describes the feature, use cases, and usage of delayed message in TDMQ for RabbitMQ.

## Glossary

**Delayed message**: After a message is sent to the server, the business may want the consumer to receive it after a period of time rather than immediately. This type of message is called "delayed message".

## Use Cases

Use case 1: Scenarios with requirements for message consumption time. For example, if a user fails to pay within 30 minutes after placing an order in an ecommerce system, the order will be automatically canceled.
Use case 2: Scenarios where delayed tasks need to be triggered by messages. For example, if a user hasn't placed an order after viewing an item for over 20 minutes after logging in to an app, an item review message will be automatically pushed, and a coupon for the item will be issued.

## Implementation Methods

### Method I: Achieving Delayed Messages Through Setting Message Expiration Time and Dead Letter Queues

#### How It Works

The sender sets the message expiration time, triggering the expired and unconsumed messages to be delivered to the dead letter queue under the dead letter exchange.
The consumer processes delayed messages by consuming messages from the dead letter queue.

#### Usage Examples

The code below is for example purposes only. Parameters such as the exchange type and routingKey should be replaced with values that meet your actual business requirements.
The code example for the sender is as follows. The consumer can subscribe to the dead letter queue.

```
// Declare a dead letter exchange
```

```
channel.exchangeDeclare("${dlxExchangeName}", "direct", true);
// Declare an exchange for sending delayed messages
channel.exchangeDeclare("${delayExchangeName}", "direct", true);
// Declare a queue for sending delayed messages, and specify its dead letter exchan
Map<String, Object> args = new HashMap<>();
args.put("x-dead-letter-exchange", "${dlxExchangeName}");
channel.queueDeclare("${delayQueueName}", true, false, false, args);
channel.queueBind("${delayQueueName}", "${delayExchangeName}", "");

// Declare a dead letter queue
channel.queueDeclare("${delayQueueName}", true, false, false, null);
channel.queueBind("${dlxQueueName}", "${dlxExchangeName}", "");

// Send delayed messages
int delayInSeconds = 10; // Message delay of 10 seconds
AMQP.BasicProperties props = new AMQP.BasicProperties.Builder().expiration(String.v
channel.basicPublish("${delayExchangeName}", "", props.build(), "delayed payload".g
```

The parameter descriptions are as follows

| Parameter | Description |
| --- | --- |
| ${dlxExchangeName} | Name of the dead letter exchange for delayed messages. Replace it with the name found in the console's Exchange list. |
| ${delayExchangeName} | Name of the exchange for sending delayed messages. Replace it with the name found in the console's Exchange list. |
| x-dead-letter-exchange | Queue parameter key, which is used to set the dead letter exchange corresponding to the queue. |
| ${dlxQueueName} | Name of the dead letter queue for delayed messages. |
| ${delayQueueName} | Name of the queue for sending delayed messages. |

## Method II: Achieving Delayed Messages through the Built-in rabbitmq_delayed_message_exchange Plugin

### Usage Limitations

**Note:**

For more details, see RabbitMQ Official Plugin Use Limitations.

The current plugin design is not suitable for scenes involving a large number of delayed messages (hundreds of thousands or even millions of unscheduled messages). In production environments, carefully evaluate the scale of the messages to avoid unexpected long delays, message loss, and other issues.

Delayed messages have only one persistent replica on each node. If a node fails to operate normally (for example, continuous OOM due to message heap and subsequent restart without recovery), the delayed messages on that node will not be consumed by the consumer.

Delayed exchanges do not support setting `mandatory`, meaning the producer cannot detect unroutable messages through the `basic.return` event. Therefore, make sure the corresponding exchanges, queues, and routing relationships exist before delayed messages are sent.++

Usage Examples

1. The delayed message feature in TDMQ for RabbitMQ can be used in the same way as the delayed message plugin of RabbitMQ, so you don't need to modify your business when migrating it.

```
Declare an exchange and specify its route type.
Map<String, Object> args = new HashMap<String, Object>();
args.put("x-delayed-type", "direct");
```

channel.exchangeDeclare("ExchangeName", "x-delayed-message", true, false, args);

| The parameters are described as follows: | Parameter |
| --- | --- |
| Description | x-delayed-type<br>Exchange type, which specifies the routing rule. Valid values:<br>direct<br>fanout<br>topic<br>For more information, see<br>Exchange |
| . | ExchangeName |
| Exchange name, which can be obtained from the exchange list in the console. | x-delayed-message |

2. Specified exchange type for delayed message delivery.

```
Send a delayed message. Add a key-value pair (key: x-delay; value: number of millis
byte[] messageBodyBytes = "delayed payload".getBytes("UTF-8");
Map<String, Object> headers = new HashMap<String, Object>();
headers.put("x-delay", 4000);
AMQP.BasicProperties.Builder props = new AMQP.BasicProperties.Builder().headers(hea
```

channel.basicPublish("ExchangeName", "", props.build(), messageBodyBytes);