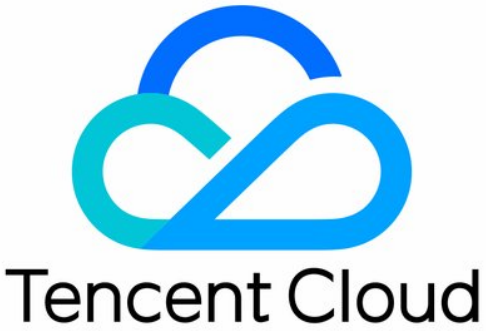
TDMQ for RabbitMQ

Product Overview



Copyright Notice

Trademark Notice

Service Notice

This document provides an overview of the as-is details of Tencent Cloud's products and services in their entirety or part. The descriptions of certain products and services may be subject to adjustments from time to time.

The commercial contract concluded by you and Tencent Cloud will provide the specific types of Tencent Cloud products and services you purchase and the service standards. Unless otherwise agreed upon by both parties, Tencent Cloud does not make any explicit or implied commitments or warranties regarding the content of this document.

Contact Us

We are committed to providing personalized pre-sales consultation and technical after-sale support. Don't hesitate to contact us at 4009100100 or 95716 for any inquiries or concerns.

Contents

Product Overview
Overview

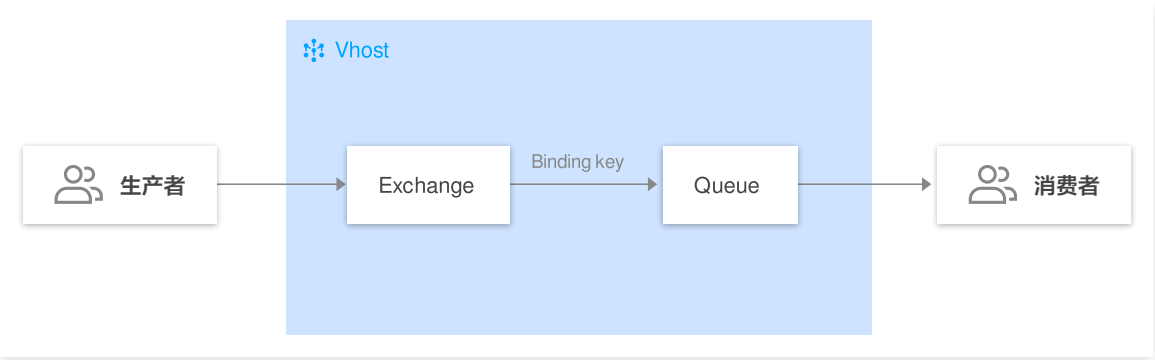
Last updated：2024-08-02 11:42:46

TDMQ for RabbitMQ is a proprietary message queue service developed by Tencent. It supports the AMQP 0-9-1 protocol and is fully compatible with all components and principles of Apache RabbitMQ. It also has the underlying benefits of computing-storage separation and flexible scaling.
TDMQ for RabbitMQ has extremely flexible routing to adapt to the message delivery rules of various businesses. It can buffer the upstream traffic pressure and ensure the stable operations of the message system. It is often used to implement async communication and service decoupling between systems to reduce their mutual dependency, making it widely applicable to distributed systems in finance and government affairs industries.

Architecture

The basic concepts of TDMQ for RabbitMQ are as follows:

- Producer: Sends messages to exchanges.
- Vhost: Is used for logical isolation. Exchanges and queues of different vhosts are isolated from each other.
- Exchange: Receives messages from producers and routes them to queues.
- Queue: Caches messages for consumers to consume them.
- Consumer: Pulls messages from queues for consumption.



For more concepts of TDMQ for RabbitMQ, see Relevant Concepts.

Strengths

Last updated：2024-08-02 11:44:41

Open-Source Version Compatibility

TDMQ for RabbitMQ supports the AMQP 0-9-1 standard protocol and is fully compatible with the open-source RabbitMQ community and its queue, exchange, and vhost components, so that you can quickly migrate the metadata from RabbitMQ to Tencent Cloud at zero costs.

Comprehensive features

TDMQ for RabbitMQ supports various messaging patterns of native RabbitMQ as well as dead letter switch and standby switch, eliminating your concerns over message loss caused by message expiration or routing failure.

Stable and Reliable

TDMQ for RabbitMQ features a persistent storage mechanism for high availability. The persistence of exchanges, queues, and messages ensures that the metadata and message content after service restart will never get lost. It stores messages in three replicas. When a physical machine is faulty, the data can be quickly migrated to guarantee the availability of three data replicas, achieving a 99.95% service availability.

High scalability

TDMQ for RabbitMQ supports more queues and has higher scalability than open-source RabbitMQ. Its underlying system can automatically scale clusters based on the business scale in a way imperceptible to users.

Easy to use and maintenance-free

TDMQ for RabbitMQ provides access APIs and open-source SDKs for all programming languages on all versions. It offers the entire set of Ops services of the Tencent Cloud platform and monitors alarms in real time to help you quickly discover and solve problems and guarantee the service availability.

Application Scenario

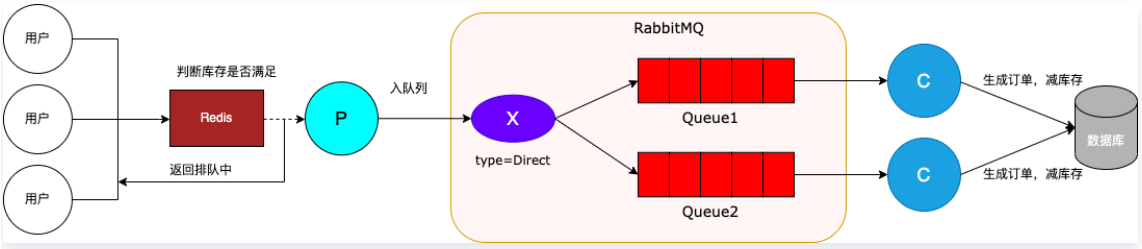
Last updated：2024-08-02 12:03:30

Flash Sale Scenario

Flash sales are a very common scenario in e-commerce systems. There are many solutions, but using TDMQ for RabbitMQ is a good approach.
If there is complex inventory deduction (such as involving product information itself or affecting other systems), it is recommended to use a database to deduct the inventory amount. An asynchronous method can be used to handle this high-concurrency inventory update.

1. When the user places an order, the order is not immediately generated, but all orders are placed in a queue sequentially.
2. The order placement module retrieves orders from the queue in sequence for the "place order and deduct inventory" operation based on its processing speed.
3. After the order is successfully generated, the user can proceed with the payment operation.

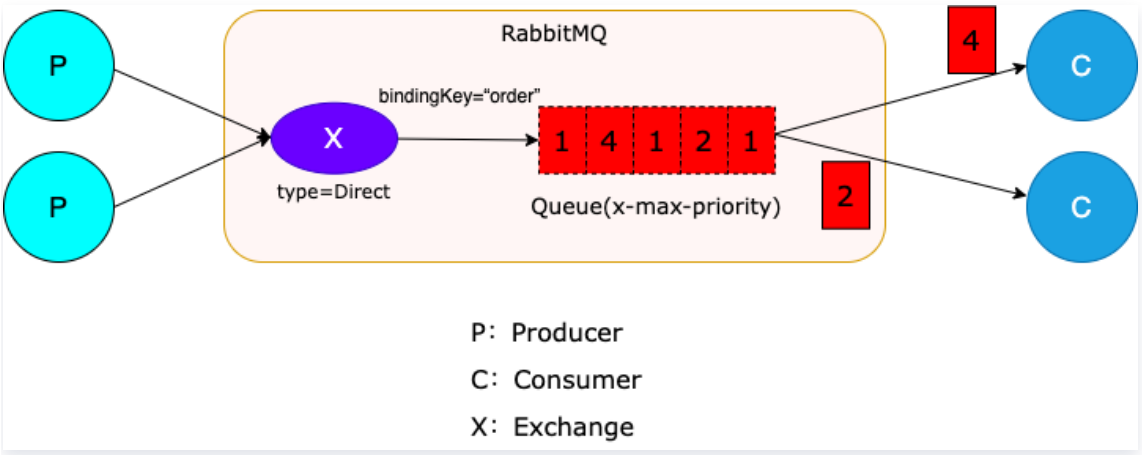
This method is designed for "flash sale" scenarios, ensuring fairness and justice based on the "first come, first served" principle. All users have the opportunity to purchase and then wait for order processing. Finally, an order is generated (if the inventory is insufficient, the order generation fails).



Priority Messages

When consuming messages, if the importance of the messages differs and more important messages need to be consumed first, TDMQ for RabbitMQ's priority queue capability can be used to prioritize high-priority messages.
Refer to the following business scenario:
For example, in the system, there is an order reminder scenario. When customers place orders in the e-commerce system, the system promptly pushes the orders to customers. If the payment is not completed within the set time, an SMS reminder is sent to the customer. However, the e-commerce system distinguishes between key accounts and small customers.

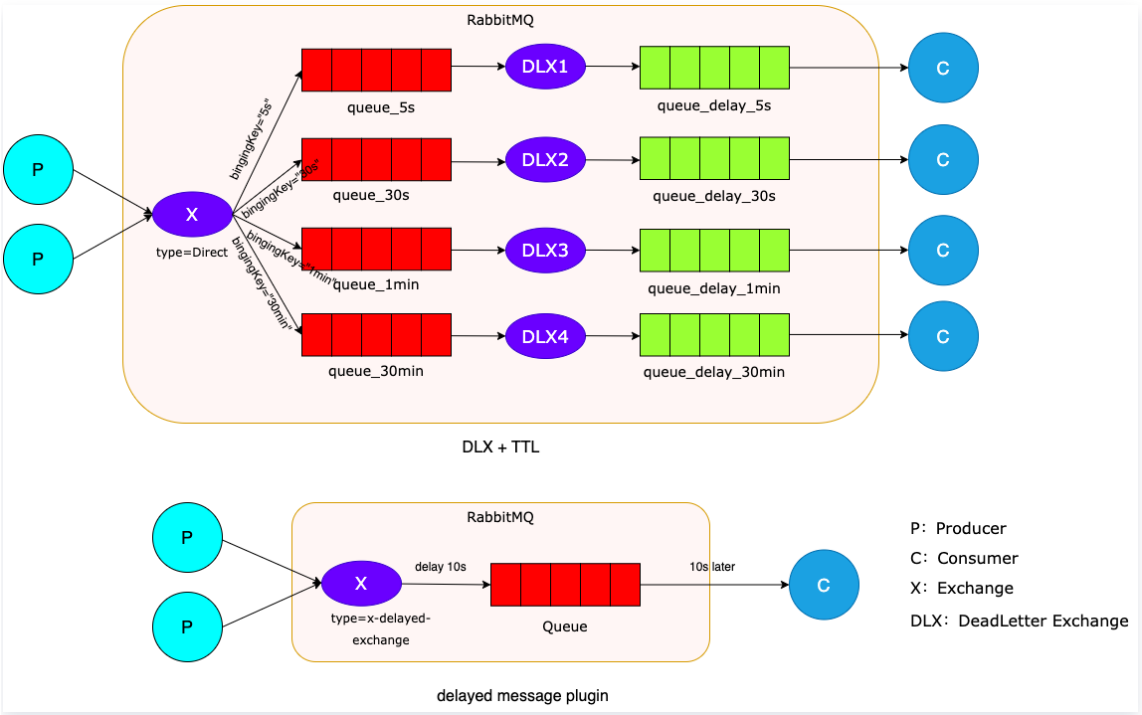
The orders for key accounts need to be prioritized, while the reminders for small customers have relatively lower priority.
Using TDMQ for RabbitMQ's priority queue can support this scenario well, ensuring that high-priority messages are not delayed for too long. If the order is from a key account, a relatively high priority is assigned for prompt processing; otherwise, the default priority is used.



Delayed Message Scenario

In actual business systems, there is a need to send delayed messages. Implementing the delay logic on your own makes it difficult to ensure reliability and delay precision. Using message middleware, such as TDMQ for RabbitMQ, can handle such demands well. The use scenarios for delayed messages include:

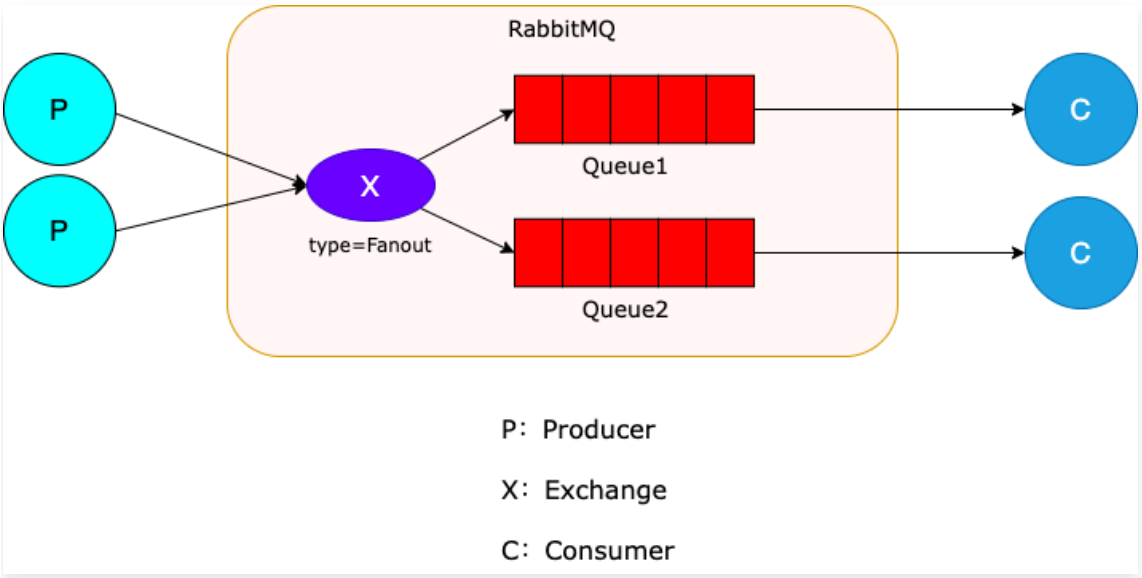
- In the order system, after the user places an order, they have 30 minutes to complete the payment. If the payment is not successful within 30 minutes, the order will go through exception handling. Here, TDMQ for RabbitMQ's delayed message capability can be used to handle these timeout orders.
- In an Internet of Things System, users want to remotely control smart devices via their phones to operate at a specific time. The command can be placed in a latency queue, and when the specified time arrives, the control command is pushed to the smart device.



Message Broadcasting

Many business systems need to broadcast information to downstream systems. Using RPC results in heavy coupling and significant stress on the upstream business system. At this point, TDMQ for RabbitMQ's Fanout Exchange can handle such demands. Use cases include:

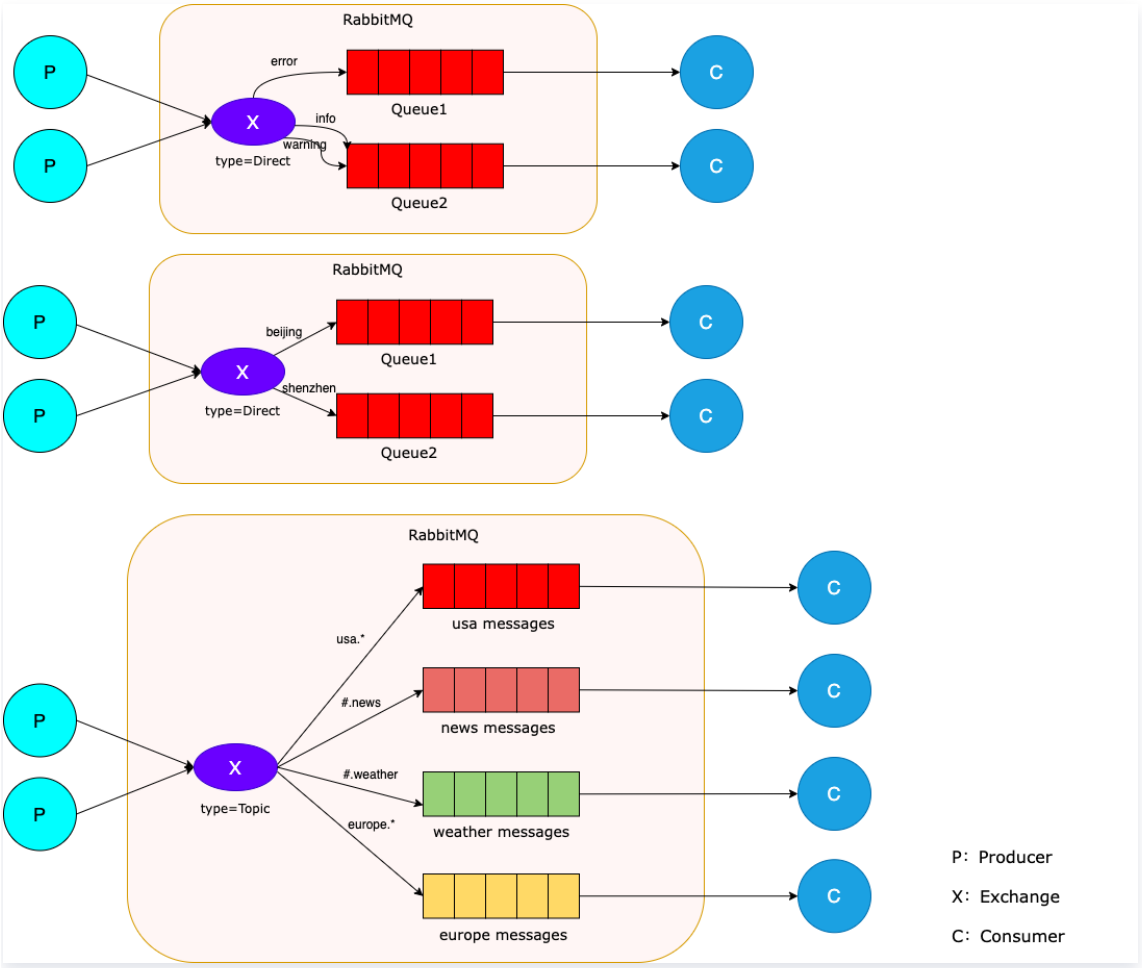
- Massive Multiplayer Online Games (MMO) can use it for leaderboard updates or other global events;
- Sports news websites can use the fan-out switch to distribute score information to clients in near real-time;
- Distributed systems use it to broadcast various status and configuration updates;
- Group chats can use it to distribute messages among participants.



Flexible Routing Scenarios

With the popularity of Microservices Architecture, services are split into smaller parts, and data is sent in the form of messages to different queues using carefully designed distribution strategies. TDMQ for RabbitMQ's flexible message routing capabilities can fully distribute messages to target queues. Use cases include:

- Log processing scenarios, where logs can be delivered to different queues based on their type. For instance, errors can be placed in a dedicated processing queue for priority handling.
- E-commerce logistics system can distribute logistics information by region to different consumer sides for processing.
- Complex Routing Scenarios.



Use Limits

Last updated: 2024-08-02 12:04:07

This document lists the limits for clusters and characters in the TDMQ RabbitMQ Edition dedicated cluster. Please be careful not to exceed these limits to avoid exceptions.

Cluster

Limit	Restrictions
Cluster name length	3-64 characters
Cluster Name Specifications	Cannot be empty; can contain only numbers, letters, "-" and "_"

Vhost

Limit	Restrictions
Vhost name length	1-64 characters
Vhost Name Specifications	Cannot be empty; can contain only letters, numbers, ".", "-" and "_"
Vhost Count	Each cluster is limited to 20 vhosts

Exchange

Limit	Restrictions
Exchange name length	1-64 characters
Exchange Name Specifications	Cannot be empty; can contain only letters, numbers, ".", "-" and "_"
Exchange Routing Type	Options include direct, fanout, topic, and headers. One must be selected during creation
Exchange Count	Each cluster is limited to 1000 exchanges

Queue

Limit	Restrictions

Queue name length	1-64 characters
Queue Name Specifications	Cannot be empty; can contain only letters, numbers, ".", "-" and "_"
Number of Queues	Each cluster is limited to 1000 exchanges

Channel

Limit	Restrictions
Connection Channels	The maximum number of channels per connection is 1024. If the user creates more channels on this connection than this limit, no additional channels can be created

Concepts
Basic Concepts

Last updated: 2024-08-02 12:05:37

This document describes the common terms in TDMQ for RabbitMQ.

Binding

RabbitMQ associates an exchange with a queue through a binding, so that it can know how to correctly route a message to the specified queue.

Binding key

- When binding an exchange to a queue, a binding key is generally specified; when a producer sends a message to the exchange, a routing key is typically designated. If the binding key matches the routing key, the message will be routed to the corresponding queue.
- When binding multiple queues to the same exchange, these bindings can use the same binding key.
- The binding key does not always take effect; it depends on the exchange type. For example, a fanout exchange ignores the binding key and routes messages to all queues bound to it.

Channel

In each physical TCP connection of a client, multiple channels can be established, each of which represents a session task.

Connection

It refers to a physical TCP connection between a producer or consumer and TDMQ for RabbitMQ.

Exchange

A producer sends a message to an exchange, which then routes the message to one or more queues based on its attributes or content (or discards it).

Exchange Types

Common RabbitMQ exchange types include fanout, direct, topic, and header.

- Fanout: A fanout exchange routes all messages sent to it to all queues bound to it.

- Direct: A direct exchange routes messages to queues where the binding key exactly matches the routing key.
- Topic: Similar to direct exchanges, but topic exchanges support multi-condition matching and fuzzy matching, routing messages based on routing key pattern matching and string comparison to the bound queues.
- Header: Independent of the routing key, the matching mechanism relies on matching the headers attributes of messages. Before binding a queue to a headers exchange, a map key-value pair is declared. This map object is used to bind the queue and exchange. When a message is sent to RabbitMQ, its headers are matched against the key-value pairs specified during the exchange binding. If they match exactly, the message is routed to the queue; otherwise, it is not.

Message acknowledgment

Message Acknowledgment: After consuming a message, consumers send an acknowledgment to RabbitMQ. RabbitMQ deletes the message from the queue only after receiving the acknowledgment. If no acknowledgment is received and the consumer’s RabbitMQ connection is detected as closed, RabbitMQ sends the message to another consumer for processing.

Message durability

Message Persistence ensures that messages are not lost even if the RabbitMQ service restarts. By setting both the queue and messages as durable, most RabbitMQ messages are protected from loss.

Prefetch count

When consumers enable acknowledgment, they can asynchronously acknowledge messages based on business needs. Prefetch allows specifying the maximum number of unacknowledged messages per consumer. For example, if `prefetchCount=10`, each consumer (say A and B) receives 10 messages at a time. Consumers do not need to respond immediately, caching the 10 messages locally. Upon processing one message, the queue sends another message to the consumer. If neither consumer acknowledges any messages, the queue does not send additional messages.

Queue

A queue is an internal object of RabbitMQ used to store messages. Each message will be put into one or more queues.

Routing key

- When a producer sends a message to an exchange, it usually specifies a routing key to set the routing rules for the message. The routing key must work together with the exchange type and binding key to take effect.
- If the exchange type and binding key are fixed (which is generally pre-configured for normal use), producers can determine the message flow by specifying the routing key when sending messages to the exchange.

Vhost

A virtual host (vhost) is used for logical isolation and manages its own exchanges, queues, and bindings separately, so that applications can run securely in different vhosts without interfering with each other. There can be multiple vhosts under one instance, and there can also be multiple exchanges and queues in one vhost. A vhost must be specified when a producer or consumer connects to TDMQ for RabbitMQ.

Exchange

Last updated: 2024-08-02 12:06:04

This article introduces the concepts, types, and usage of Exchange in TDMQ RabbitMQ Edition.

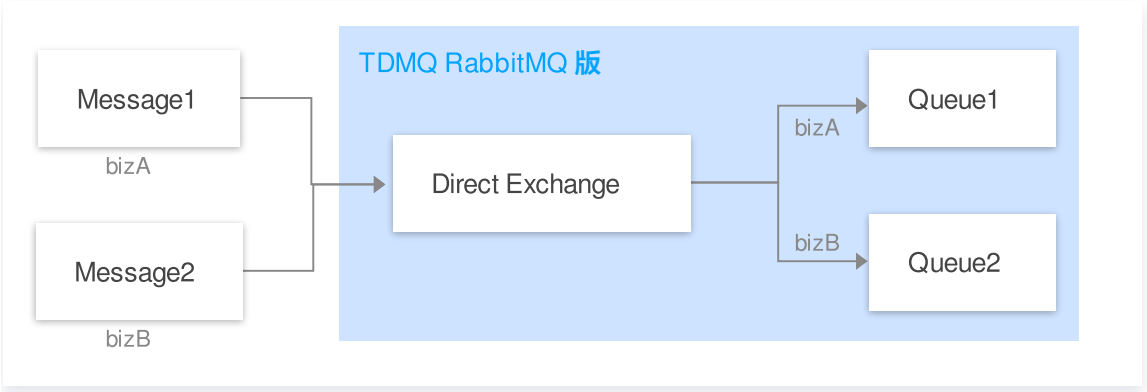
Concept

Exchange is the message routing agent in TDMQ RabbitMQ Edition. The Producer sends messages to the Exchange, and the Exchange routes the messages to one or more Queues (or discards them) based on the attributes or content of the messages. Consumers pull messages from the Queue for consumption.
TDMQ RabbitMQ Edition currently supports four types of Exchanges: Direct, Fanout, Topic, and Header.

- Direct: This type of Exchange routes messages to Queues where the RoutingKey exactly matches the BindingKey.
- Fanout: This type of Exchange routes messages to all Queues bound to it.
- Topic: This type of Exchange supports multi-condition and fuzzy matching. It routes messages to bound Queues based on pattern matching and string comparison using the RoutingKey.
- Header: This type of Exchange is independent of the RoutingKey. The matching mechanism is based on matching the Headers attribute information in the message. Before binding a Queue to a Headers Exchange, declare a map key-value pair. Use this map object to bind the message queue and the Exchange.

Direct Exchange

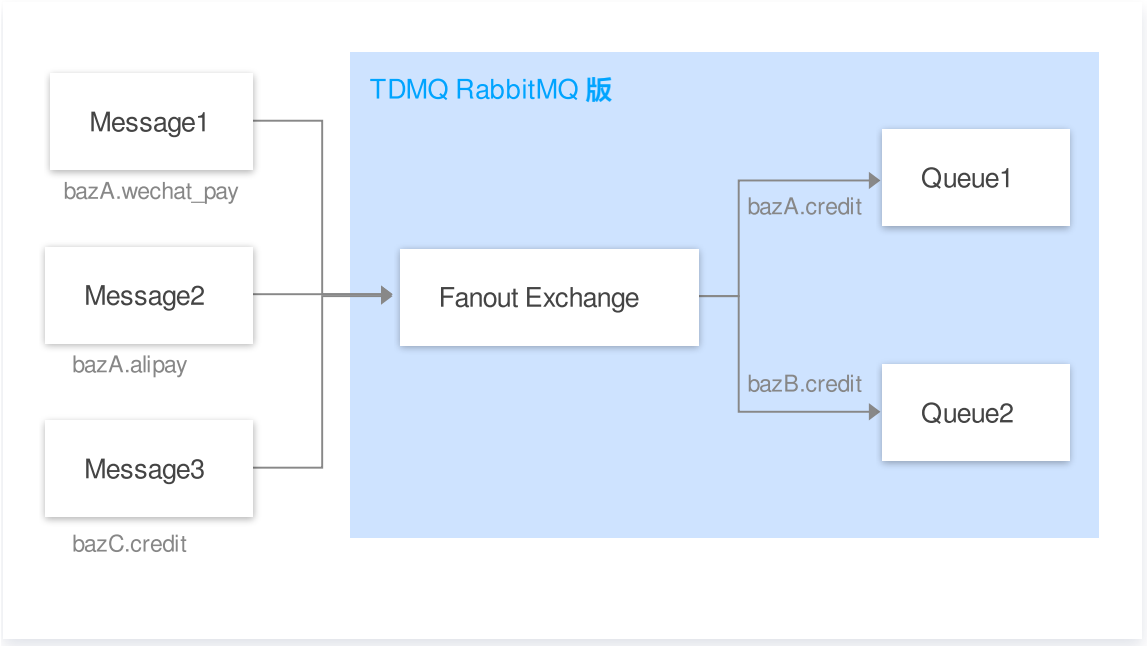
Routing Rule: Direct Exchange routes messages to Queues where the RoutingKey exactly matches the BindingKey.
Scenarios: This type of Exchange is suitable for scenarios where messages are filtered through simple character identifiers, commonly used for Unicast Routing.
Sample:



Message	Routing Key	Binding Key	Queue
Message 1	`bizA`	`bizA`	Queue 1
Message 2	`bizB`	`bizB`	Queue 2

Fanout Exchange

Routing Rule: This type of Exchange routes messages to all bound Queues.
Scenarios: This type of Exchange is suitable for Broadcast Messages scenarios. For example, distribution systems use Fanout Exchange to broadcast various statuses and Configuration Updates.
Sample



Message	Routing Key	Binding Key	Queue
Message 1	`bazA.wechat_pay`	`bazA.credit` , `bazB.credit`	Queue 1,Queue 2

Message 2	`bazA.alipay`	`bazA.credit` , `bazB.credit`	Queue 1,Queue 2
Message 3	`bazC.credit`	`bazA.credit` , `bazB.credit`	Queue 1,Queue 2

Topic Exchange

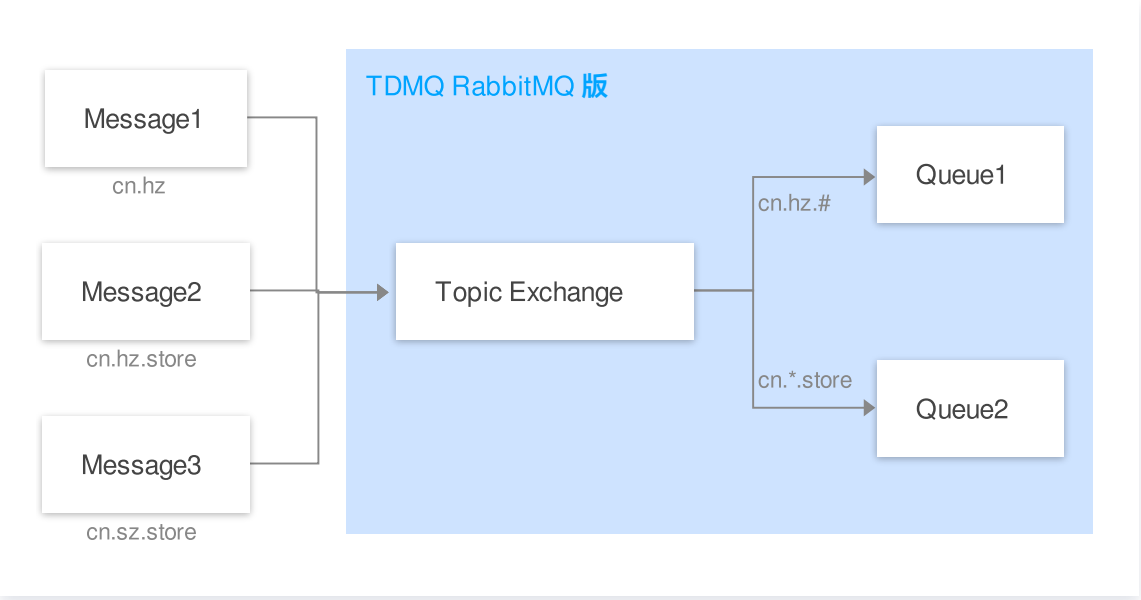
Routing Rule: This type of Exchange supports multi-condition and fuzzy matching. It routes messages to bound Queues based on pattern matching and string comparison using the RoutingKey.

- The wildcards supported by Topic Exchange include Asterisk "*" and Pound Sign "#".
- Asterisk "*" represents a single English word, for example sh.
- Pound sign "#" represents zero, one, or multiple English words, separated by English periods ".", for example cn.hz.

Application Scenario

This type of Exchange is commonly used for multicast routing, such as distributing data related to specific geographical locations using a Topic Exchange.

Sample

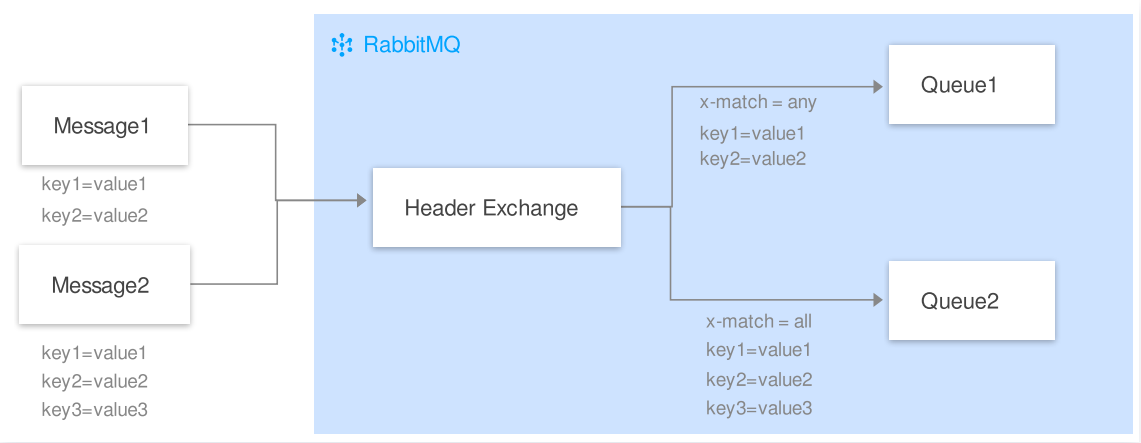


Message	Routing Key	Binding Key	Queue
Message 1	`cn.hz`	`cn.hz.#`	Queue 1
Message 2	`cn.hz.store`	`cn.hz.#` , `cn.*.store`	Queue 1,Queue 2
Message 3	`cn.sz.store`	`cn.*.store`	Queue 2

Header Exchange

It is independent of the Routing Key. The matching mechanism is based on matching the Headers attribute information in the message. Before binding a Queue to a Headers Exchange, declare a map key-value pair. Use this map object to bind the Queue and the Exchange. When a message is sent to RabbitMQ, it will match the Headers of the message with the key-value pairs specified during the Exchange binding. If there is a complete match, the message will be routed to the Queue; otherwise, it will not be routed to the Queue.
There are two types of matching rules for x-match:

- x-match = all: All key-value pairs must match to receive the message
- x-match = any: Any key-value pair match will allow the message to be received



Message	Message Headers Attribute	Binding Headers Attribute	Queue

Message 1	`key1=value1`
	`key2=value2`
		`x-match = any`
		`key1=value1`
		`key2=value2`
			Queue 1

Message 2	`key1=value1`
	`key2=value2`
	`key3=value3`
		`x-match = any`
		`key1=value1`
		`key2=value2`
		,
		`x-match = all`
		`key1=value1`
		`key2=value2`
		`key3=value3`
			Queue 1,Queue 2