TDMQ for RabbitMQ

Product Introduction

Product Documentation

Contents

Product Introduction

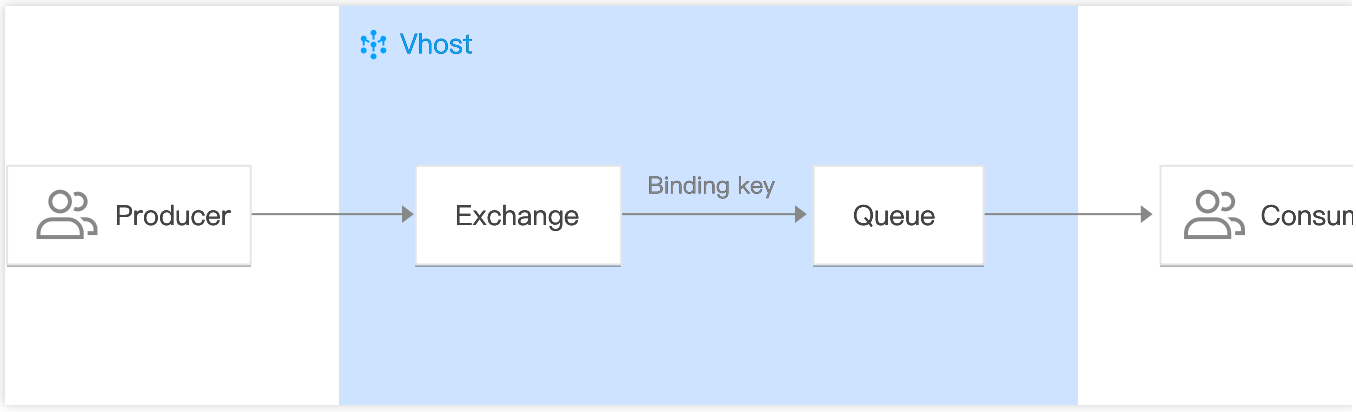
Overview

Last updated：2024-01-03 11:39:16

TDMQ for RabbitMQ is Tencent's proprietary message queue service. 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.

Infrastructure

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 Basic Concepts.

Strengths

Last updated：2024-01-03 11:39:16

Open-Source Component 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.

Diverse 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.

Stability and Reliability

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.

Ease of Ops-Free Use

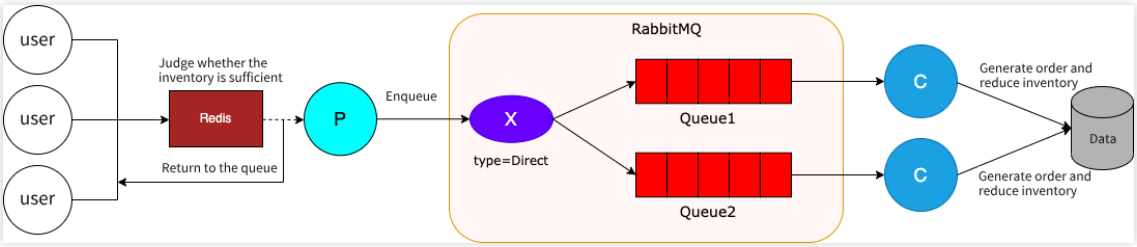
TDMQ for RabbitMQ provides access APIs and open-source SDKs for all programming languages on all versions. It offers the entire set of Ops services of the Tencent Cloud platform and monitors alarms in real time to help you quickly discover and solve problems and guarantee the service availability.

Use Cases

Last updated：2024-10-18 17:35:37

Flash Sale Scene

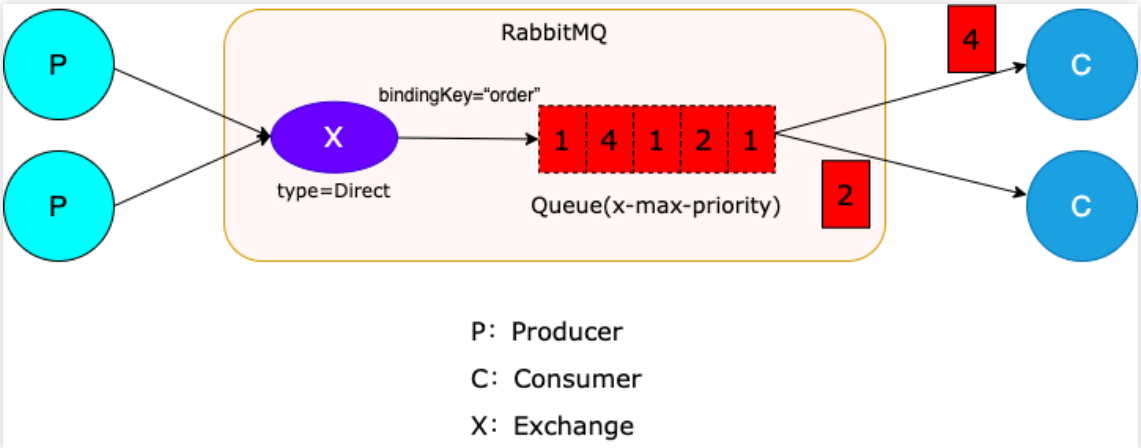
Flash sale is a very common scene in e-commerce systems. There are various solutions, but using TDMQ for RabbitMQ is an effective approach.
If the inventory deduction is complex (such as involving product information itself or affecting other systems), it is recommended to use a database for inventory quantity deduction. You can use an asynchronous approach to handle this high-concurrency inventory update.
1. When a user places an order, the order is not generated immediately but is sequentially placed into a queue.
2. The order module processes orders according to its speed, retrieving them from the queue one by one to perform the inventory deduction operation.
3. Once the order is successfully generated, the user can proceed with the payment.
This method is designed for the flash sale scene, following the first-come, first-served principle to ensure fairness. All users can try to purchase, and then wait for order processing. Finally, the order is generated (if stock is insufficient, the order fails).



Priority Messages

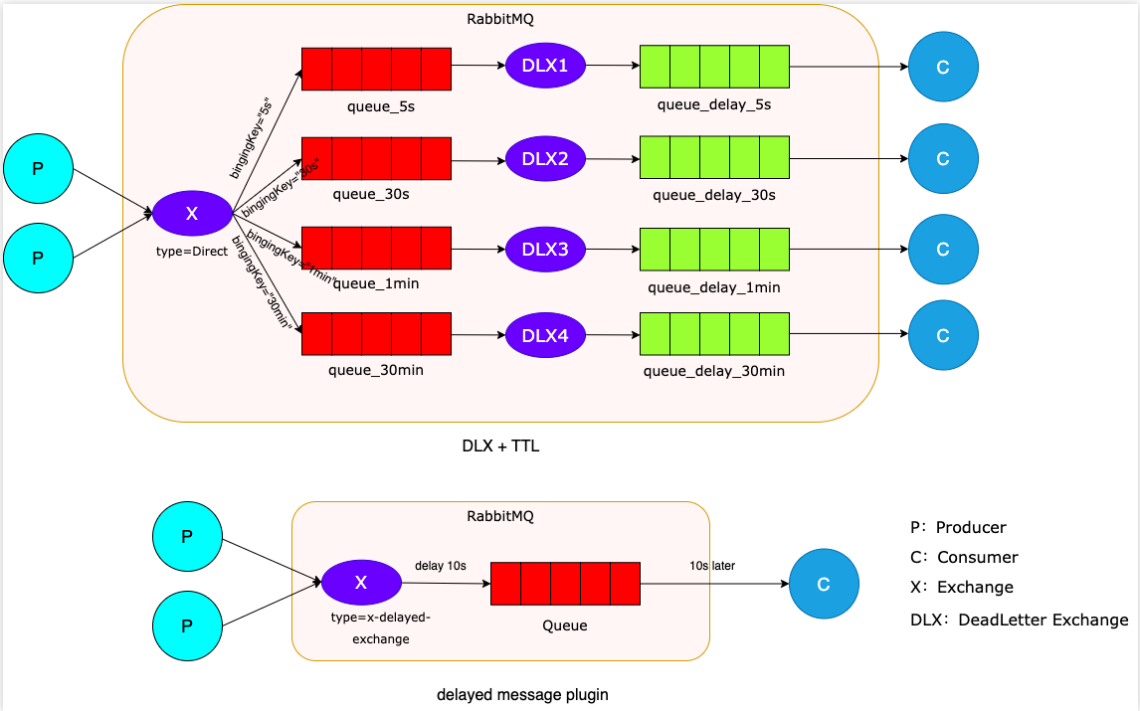
When messages are consumed, if their level of importance varies, and higher-priority messages need to be consumed first, you can use the priority queue feature of TDMQ for RabbitMQ. This ensures that higher-priority messages are consumed ahead of others.
Business scene example:
For instance, in an order payment reminder scene, when a customer places an order on an e-commerce system, the system promptly sends the order to the customer. If the payment is not completed within the specified time, a reminder SMS will be sent to the customer. However, the e-commerce system distinguishes between major and minor customers. Payment reminders for major customers need to be prioritized, while reminders for smaller customers will have a lower priority.

The priority queue feature of TDMQ for RabbitMQ can effectively support this scene, ensuring that high-priority messages don't remain queued for too long. If an order is identified as coming from a major customer, it can be assigned a relatively higher priority for faster processing; otherwise, it will follow the default priority.



Delayed Message Scene

In real-world business systems, there is often a need to send delayed messages. If you implement the delay logic yourself, ensuring reliability and delay accuracy can be challenging. Using a messaging middleware like TDMQ for RabbitMQ can effectively handle such requirements. Scenes for using delayed messages include:
In an order system, after a user places an order, they have 30 minutes to complete the payment. If the payment is not successfully made within 30 minutes, the order will undergo exception handling. In this case, the delayed message capability of TDMQ for RabbitMQ can be used to handle these timed-out orders.
In an IoT system, users may want to remotely control smart devices to operate at a specified time using their mobile phones. In this case, the control instructions can be sent to a delayed queue, and once the specified time arrives, the control instructions will be pushed to the smart device.



Message Broadcasting

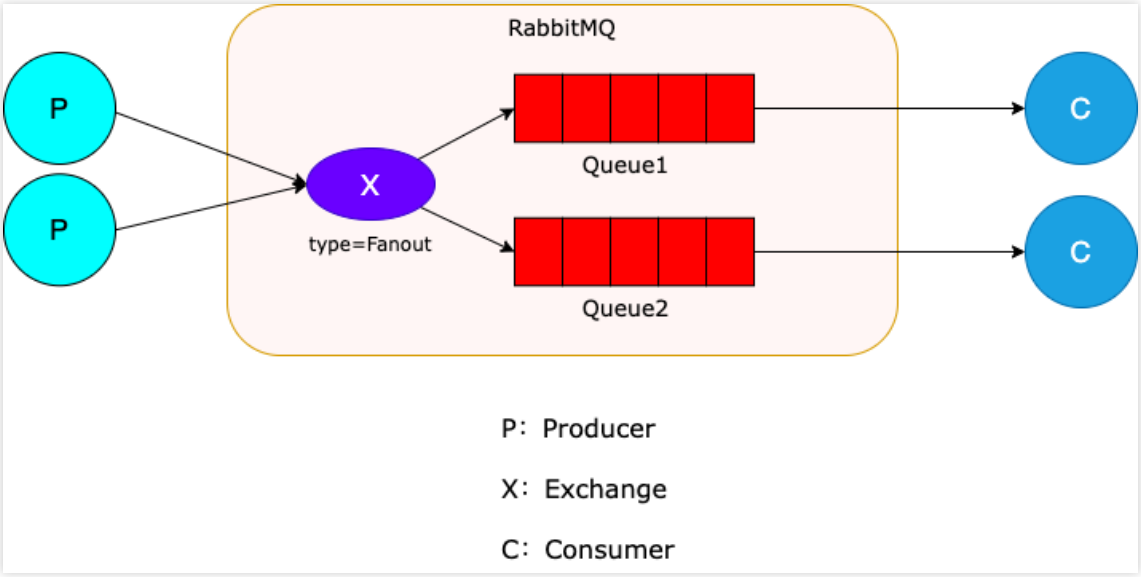
Many business systems need to broadcast information to downstream systems. Using RPC for this purpose can lead to high coupling and increased pressure on the upstream system. In this case, the Fanout Exchange feature of TDMQ for RabbitMQ can be used to handle such requirements. Applicable scenes include:

Massively Multiplayer Online (MMO) games can use it for leaderboard updates or other global events;

Sports news websites can use it to distribute live score updates to clients in near real-time;

Distributed systems can use it to broadcast various status and configuration updates;

Group chats can use it to distribute messages among participants.



Flexible Routing Scenes

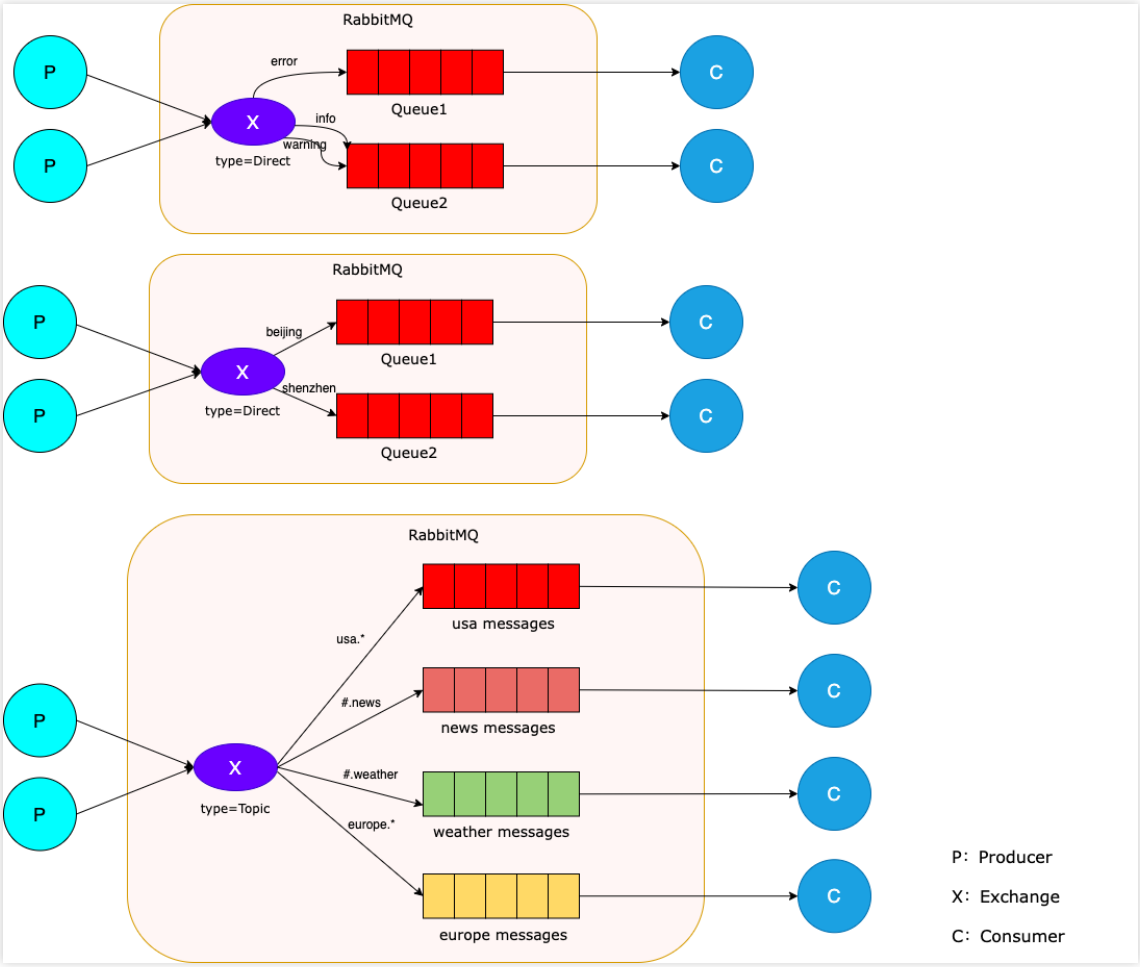
With the rise of microservice architecture, services are often broken down into smaller components, and data from these services is sent as messages to different queues using carefully designed distribution strategies. TDMQ for RabbitMQ's flexible message routing capabilities can be fully used to distribute messages to target Queues.

Applicable scenes include:

Log processing scenes, where logs can be delivered to different Queues based on type. For example, Error logs can have a dedicated queue for priority processing.

In an e-commerce logistics system, logistics information can be distributed by region to different consumers for processing.

Complex routing scenes.



Use Limits

Last updated：2025-01-02 16:23:39

This document lists the cluster and character limits in the TDMQ for RabbitMQ Exclusive Cluster Edition. When you use the product, be careful not to exceed the corresponding limits to avoid exceptions.

Cluster

Limit	Description
Cluster Name Length	A cluster name is a string of 3 to 64 characters.
Cluster Name Specification	A cluster name cannot be empty and can only contain digits, letters, hyphens (-), and underscores (_).

Vhost

Limit	Description
Vhost Name Length	A Vhost name is a string of 1 to 64 characters.
Vhost Name Specification	A Vhost name cannot be empty and can only contain letters, digits, periods (.), hyphens (-), and underscores (_).
Vhost Quantity	Each Vhost can have up to 20 Vhosts.

Exchange

Limit	Description
Exchange Name Length	An exchange name is a string of 1 to 64 characters.
Exchange Name Specification	An exchange name cannot be empty and can only contain letters, digits, periods (.), hyphens (-), and underscores (_).
Exchange Routing Type	Options include **direct**, **fanout**, **topic**, and **headers**. A routing type must be selected when an exchange is created.
Number of Exchanges	Each Vhost can have up to 1000 exchanges.

Queue

Limit	Description

Queue Name Length	A queue name is a string of 1 to 64 characters.
Queue Name Specification	A queue name cannot be empty and can only contain letters, digits, period (.), hyphens (-), and underscores (_).
Number of Queues	Each cluster can have up to 1000 queues.

Channel

Limit	Description
Number of Channels Per Connection	The maximum number of channels per connection is 1,024. If a user creates more channels than this limit on a connection, no more channels can be created.

Message

Limit	Description
Message Size	The maximum size of each message is 128 MB. Messages exceeding this limit will result in an exception.

Relevant Concepts
Basic Concepts

Last updated：2024-10-18 17:35:37

This document mainly introduces common terms and their explanations used in TDMQ for RabbitMQ.

Binding

In RabbitMQ, a Binding associates an Exchange with a Queue, allowing RabbitMQ to know how to correctly route messages to the specified Queue.

Binding key

When you bind an Exchange with a Queue, a binding key is usually specified. When the producer sends a message to the Exchange, it generally includes a routing key. If the binding key matches the routing key, the message will be routed to the corresponding Queue.
When you bind multiple Queues to the same Exchange, these Bindings can use the same binding key.
The binding key is not always effective in all situations; it depends on the Exchange type. For example, a fanout Exchange ignores the binding key and routes messages to all Queues bound to that Exchange.

Channel

Within each physical TCP connection from a client, multiple Channels can be established, with each Channel representing a session task.

Connection

A TCP connection is the physical connection established between the producer or consumer and TDMQ for RabbitMQ.

Exchange

Producers send messages to an Exchange, which then routes the messages to one or more Queues (or discards them). The Exchange routes messages based on their attributes or content.

Exchange Types

RabbitMQ commonly uses Exchange Types such as fanout, direct, topic, and header.
Fanout: A fanout Exchange routes all messages sent to it to every Queue bound to it.
Direct: A direct Exchange routes messages to Queues where the binding key exactly matches the routing key.
Topic: Similar to the direct type, a topic Exchange supports multi-condition and fuzzy matching. It routes messages to the bound Queues by pattern matching and string comparison using the routing key.
Header: This type of Exchange is unrelated to the routing key. Instead, the matching mechanism relies on the Headers attribute information in the message. Before a Queue is bound 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 checked against the key-value pair specified during the binding process. If the headers match exactly, the message is routed to the corresponding Queue; otherwise, it is not.

Message Acknowledgment

Message acknowledgment: After a message is consumed, the consumer sends an acknowledgment to RabbitMQ. RabbitMQ only deletes the message from the queue after receiving this acknowledgment. If RabbitMQ does not receive the acknowledgment and detects that the consumer’s connection has been closed, it will resend the message to another consumer for processing.

Message Durability

Message durability: Message durability ensures that messages are not lost even if the RabbitMQ service restarts. Both the Queue and Message can be set to durable, which guarantees that in most cases, RabbitMQ messages will not be lost.

Prefetch Count

When consumers have message acknowledgment enabled, they can asynchronously acknowledge messages based on their business needs. The prefetch setting allows for specifying the maximum number of unacknowledged messages per consumer. For example, if prefetchCount=10 is set, the Queue will send 10 messages to each consumer (assuming there are two consumers, A and B). The consumers do not need to respond immediately and

can cache the 10 messages locally. Once a consumer processes and acknowledges a message, the Queue will send another message to that consumer. If neither consumer acknowledges any message, the Queue will not continue sending new messages.

Queue

A Queue in RabbitMQ is an internal object used for storing messages. Each message will be placed into one or more Queues.

Routing key

When a producer sends a message to an Exchange, a routing key is typically specified to define the routing rules for the message. This routing key should be used in conjunction with the Exchange Type and binding key for it to take effect.
In a scene where the Exchange Type and binding key are fixed (which is usually the case during regular use), the producer can determine the destination of the message by specifying the routing key when sending the message to the Exchange.

Vhost

A virtual host (Vhost) is used for logical isolation, allowing the separate management of individual Exchanges, Queues, and Bindings. This ensures that applications run securely on different Vhost instances without interfering with each other. A single instance can have multiple Vhosts, and each Vhost can contain several Exchanges and Queues. When connecting to TDMQ for RabbitMQ, both producers and consumers need to specify a Vhost.

Exchange

Last updated：2024-08-22 10:42:35

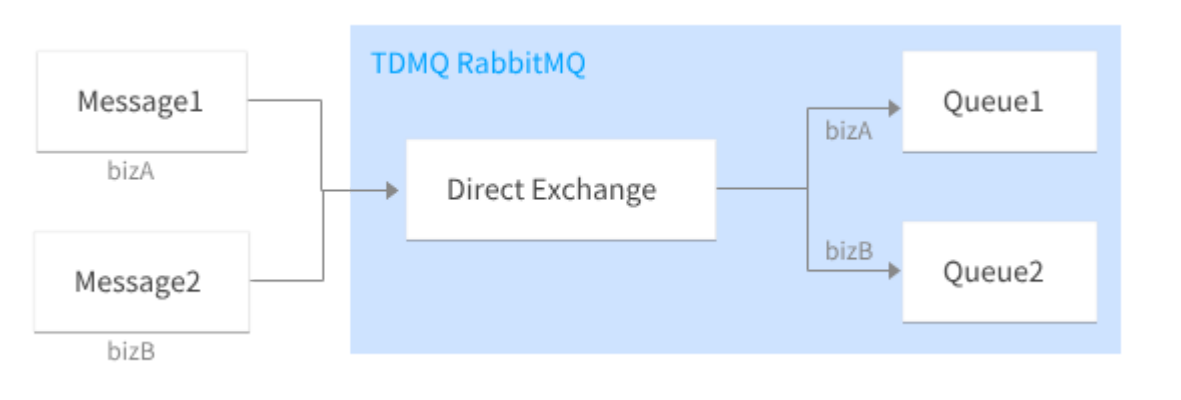
This document describes the concept, types, and usage of exchange in TDMQ for RabbitMQ.

Concept

Exchange is a message routing agent in TDMQ for RabbitMQ. 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). Then, a consumer pulls it from such queues for consumption.
TDMQ for RabbitMQ currently supports four types of exchange: Direct, Fanout, Topic, and Header.
Direct: a direct exchange will route messages to the queue whose binding key exactly matches the routing key.
Fanout: a fanout exchange will route messages to all queues bound to it.
Topic: a topic exchange supports multi-condition match and fuzzy match; that is, it will route messages to the queues bound to it by using routing key pattern match and string comparison.
Header: A header exchange has nothing to do with routing key and matches messages by the `Headers` attribute. When binding a queue to a header exchange, declare a map key-value pair to implement the binding.

Direct Exchange

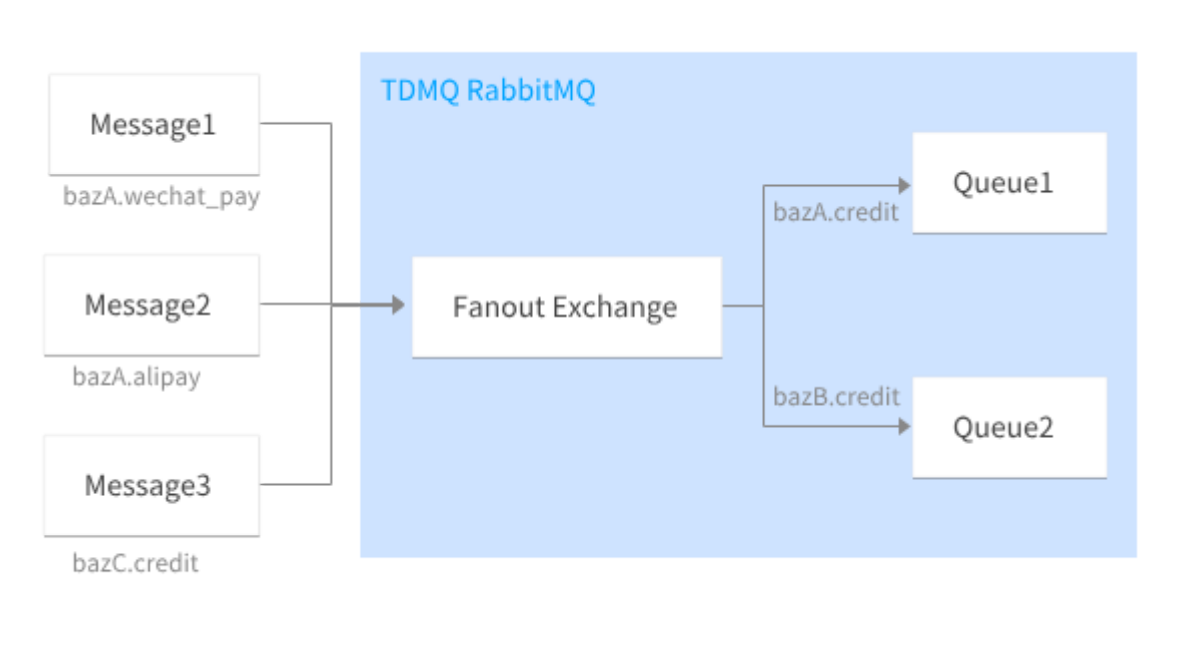
Routing rule: a direct exchange will route messages to the queue whose binding key exactly matches the routing key.
Use case: this type of exchange is suitable for filtering messages by simple character identifiers and is often used for unicast routing.
Example:



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 will route messages to all queues bound to it.
Use case: this type of exchange is suitable for broadcast message scenarios. For example, a distribution system can use a fanout exchange to broadcast various status and configuration updates.
Example



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 match and fuzzy match; that is, it will route messages to the queues bound to it by using routing key pattern match and string comparison.

The wildcards supported by topic exchanges include asterisk "*" and pound sign "#".

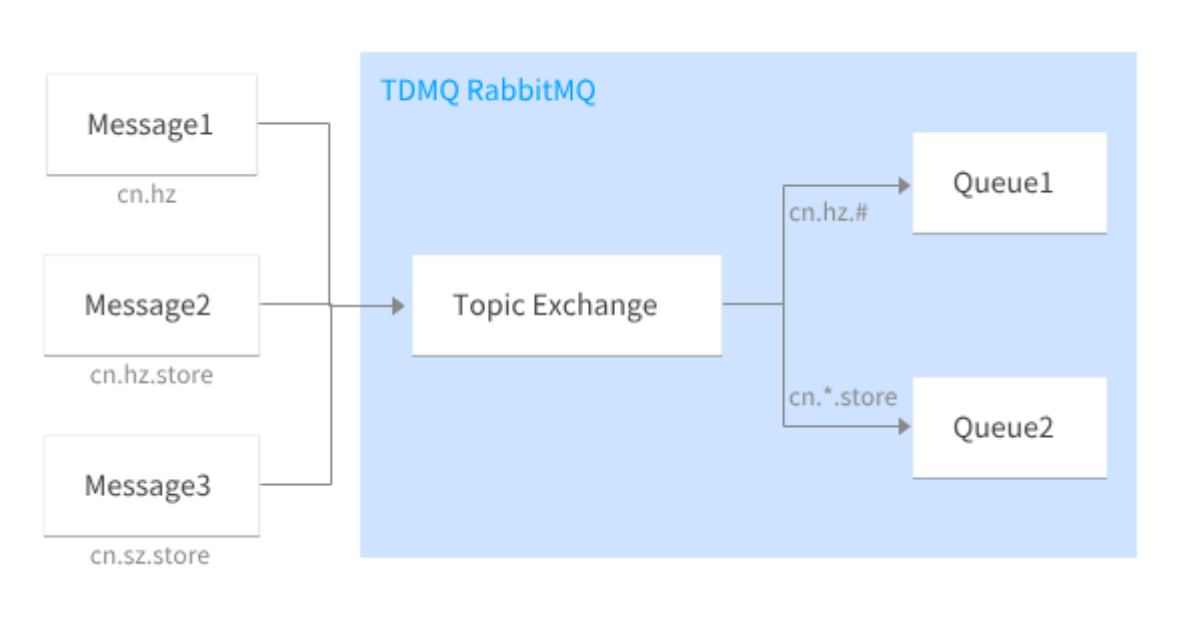
"*" represents a word, such as `sh` .

"#" represents zero, one, or more words separated by period ".", such as `cn.hz` .

Use case

This type of exchange is often used for multicast routing. For example, you can use a topic exchange to distribute data about specific geographic locations.

Example



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

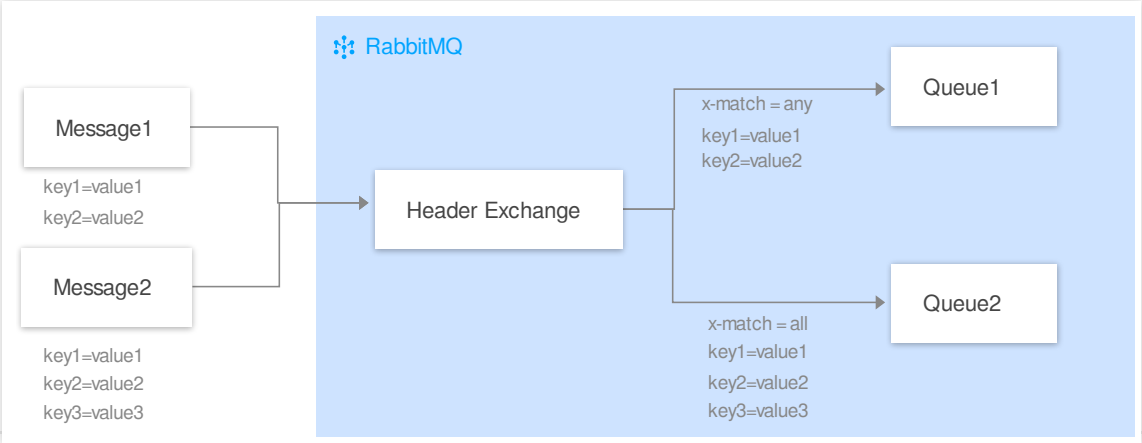
Header Exchange

A header exchange has nothing to do with routing key and matches messages by the `Headers` attribute. When binding a queue to a header exchange, declare a map key-value pair to implement the binding. When a message is sent to RabbitMQ, the `Headers` attribute of the message will be obtained to match the key-value pair specified during exchange binding, and the message will be routed to the queue only if there is a full match.

The x-match rule has two types:

x-match = all: A message can be received only if all key-value pairs are matched.

x-match = any: A message can be received as long as any key-value pair is matched.



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

Regions and Availability Zones

Last updated：2025-03-18 09:59:37

A region is the geographical region of a physical data center. An availability zone (AZ) is the physical data center of Tencent Cloud in the same region, where the power supply and network are isolated from each other. For more information, see CVM - Regions and AZs.

Supported Regions

Chinese Regions

Region		Value
South China	Guangzhou	ap-guangzhou
	Shenzhen Finance	ap-shenzhen-fsi
East China	Nanjing	ap-nanjing
	Shanghai	ap-shanghai
	Shanghai Finance	ap-shanghai-fsi
Hong Kong (China), Macao (China), and Taiwan (China)	Hong Kong (China)	ap-hongkong
North China	Beijing	ap-beijing
Southwest China	Chengdu	ap-chengdu
	Chongqing	ap-chongqing

Other Countries and Regions

Region		Value
Southeast Asia Pacific	Singapore	ap-singapore
	Bangkok	ap-bangkok
Northeast Asia Pacific	Seoul	ap-seoul
	Tokyo	ap-tokyo

West US	Silicon Valley	na-siliconvalley
East US	Virginia	na-ashburn
European regions	Frankfurt	eu-frankfurt
South America	Sao Paulo	sa-saopaulo

How to Select Regions and AZs

When selecting regions and AZs, you need to consider the following factors:
The region where the TDMQ for RabbitMQ cluster is located, your geographical location, and the location of your target users. It is recommended to select the region closest to your customers when the TDMQ for RabbitMQ cluster is purchased to reduce access delay and improve access speed.
The relationship between TDMQ for RabbitMQ and other cloud products. It is recommended to select other cloud products in the same region and AZ to facilitate private network communication between products, reducing access delay, and improving access speed.
Business high availability and disaster recovery. Even in scenarios with only one VPC, it is recommended to deploy your businesses in different AZs to ensure fault isolation between AZs and achieve cross-AZ disaster recovery. There may be network communication delay between different AZs. You should evaluate it based on your actual business needs to find the best balance between high availability and low delay.