TDMQ for CMQ

Product Introduction

Product Documentation

Contents

Product Introduction

Overview

Last updated：2024-01-03 10:15:44

TDMQ for CMQ is a distributed message queue service that features a reliable message-based async communication mechanism. It enables message sending/receiving among different applications deployed in a distributed manner (or different components of the same application) and stores the delivered messages in reliable and valid message queues to prevent message loss. It supports multi-process simultaneous read/write, so that message sending and receiving do not interfere with each other, eliminating the need for the applications or components to keep running. TDMQ for CMQ supports queue and topic patterns and is suitable for various scenarios, such as async notification, remote call, and topic message delivery. It is widely used in actual businesses, including order processing, time-consuming event callback, and operations system logging. Moreover, it can retain millions of messages to guarantee that messages will not get lost.

TDMQ for CMQ currently supports connection over HTTP and TCP.

Connection Method	HTTP Connection	TCP Connection
Application scenarios	It provides sync HTTP-based connection and can be simply and easily integrated through RESTful APIs and SDKs for multiple programming languages.	It supports sync/async TCP-based connection and SDKs for multiple programming languages, which improve the producer and consumer efficiency and increase the performance of the message queue service.
Strengths	It features unlimited message retention, finance-grade horizontal scalability, and high reliability, and messages can be stored in disks in real time.	It features unlimited message retention, finance-grade horizontal scalability, and high reliability, and messages can be stored in disks in real time. In addition, messages can be received and sent in an async non-blocking way over TCP, which improves the efficiency.

Features

Last updated：2024-01-03 10:15:44

Async Communication Protocol

The message sender can immediately get the returned result after sending a message to TDMQ for CMQ without the need to wait for the recipient's response. The message will be saved in the queue until fetched out by the recipient. The sending and processing of the message are completely async.

Improved Reliability

In traditional modes, a message request may fail due to long waits; however, if the recipient is unavailable when a message is sent through TDMQ for CMQ, it will retain the message until successfully delivered.

Process Decoupling

TDMQ for CMQ helps reduce the degree of coupling between two processes. As long as the message format stays unchanged, no changes will be made to the sender even if the recipient's API, location, or configuration changes. Moreover, the message sender does not have to know who the recipient is, making the system design clearer; in contrast, if a remote procedure call (RPC) or socket connection is used between processes, when one party's API, IP, or port changes, the other party must modify the request configuration.

Message Routing

A direct connection is not required between the sender and the recipient, as TDMQ for CMQ guarantees that the message can be routed from the former to the latter. Message routing is even available for two services that are not easily interconnectable.

Multi-Device Interconnection

Messages can be sent or received among multiple parts in a system, and TDMQ for CMQ controls the availability of messages through message status.

Diversity

Each queue may be configured separately, and they do not have to be identical. Queues in different business scenarios can be customized. For example, if a queue's message processing time is greater than expected, its queue properties can be optimized.

Strengths

Last updated：2024-01-03 10:15:44

Compared with traditional open-source message queue applications, TDMQ for CMQ has the following strengths:

Item	TDMQ for CMQ	Traditional Open-Source MQ Application
High performance	High performance and reliability can be guaranteed at the same time, and the QPS of a single TDMQ for CMQ instance can reach 5,000.	High performance and reliability cannot be guaranteed at the same time.
High scalability	The number of queues and queue storage capacity are highly scalable. The underlying system can be automatically scaled based on the business volume, which is imperceptible to upper-layer businesses. Hundreds of millions of messages can be received, sent, pushed, and retained efficiently with an unlimited capacity. The message service is provided in multiple regions: Beijing, Shanghai, and Guangzhou.	The numbers of queues and retained messages are limited. Devices must be purchased and deployed for each IDC, which is a time-consuming process.
High reliability	TDMQ for CMQ ensures that data is replicated to different physical servers in three copies before a successful write of each message is returned to the user, and the backend data replication mechanism guarantees that data can be quickly migrated when any physical server fails, so that the three copies of user data are always available with a reliability of 99.999999%. The improved Raft consistency algorithm is integrated to delivery a strong data consistency. The business availability is guaranteed at 99.95%.	Data is stored in single servers or a simple primary/secondary architecture, where data cannot be rewound once lost due to single points of failure. The open-source replica algorithm will cause rebalancing of global data when a server is added to or removed from the cluster, drastically bringing down the availability. If Kafka flushes and replicates data asynchronously, strong data consistency cannot be ensured.
Business security	Multidimensional security protection and anti-DDoS services are provided. Each message service has an independent namespace to strictly isolate data of different customers. HTTP access is supported.	The security protection features are limited. To avoid public network threats, it is usually not possible to provide cross-region and cross-IDC services over the public network.

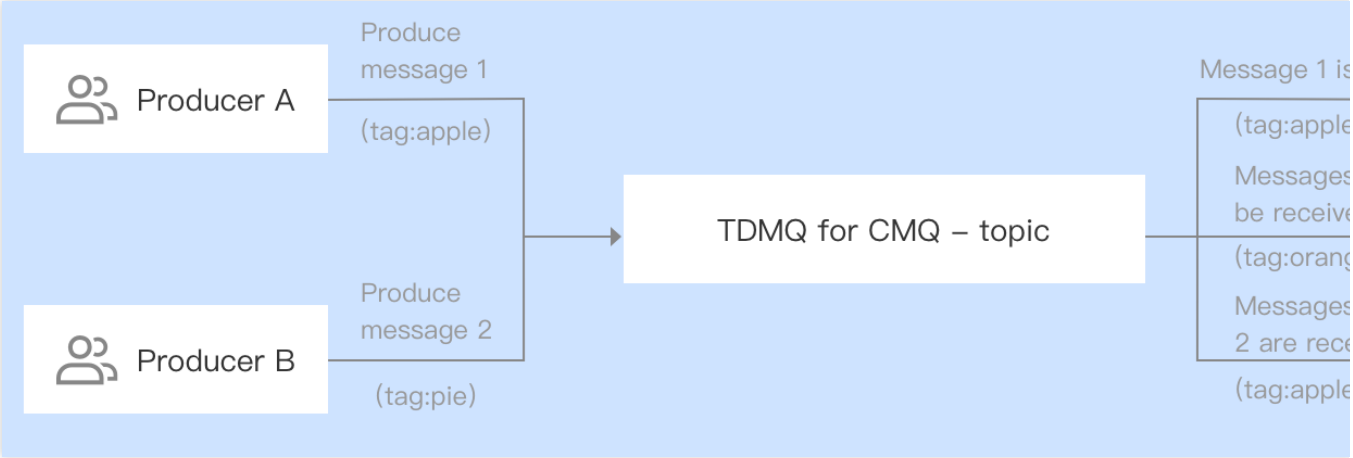
	Cross-region secure message service is supported.	

Use Cases

Last updated：2024-01-03 10:15:44

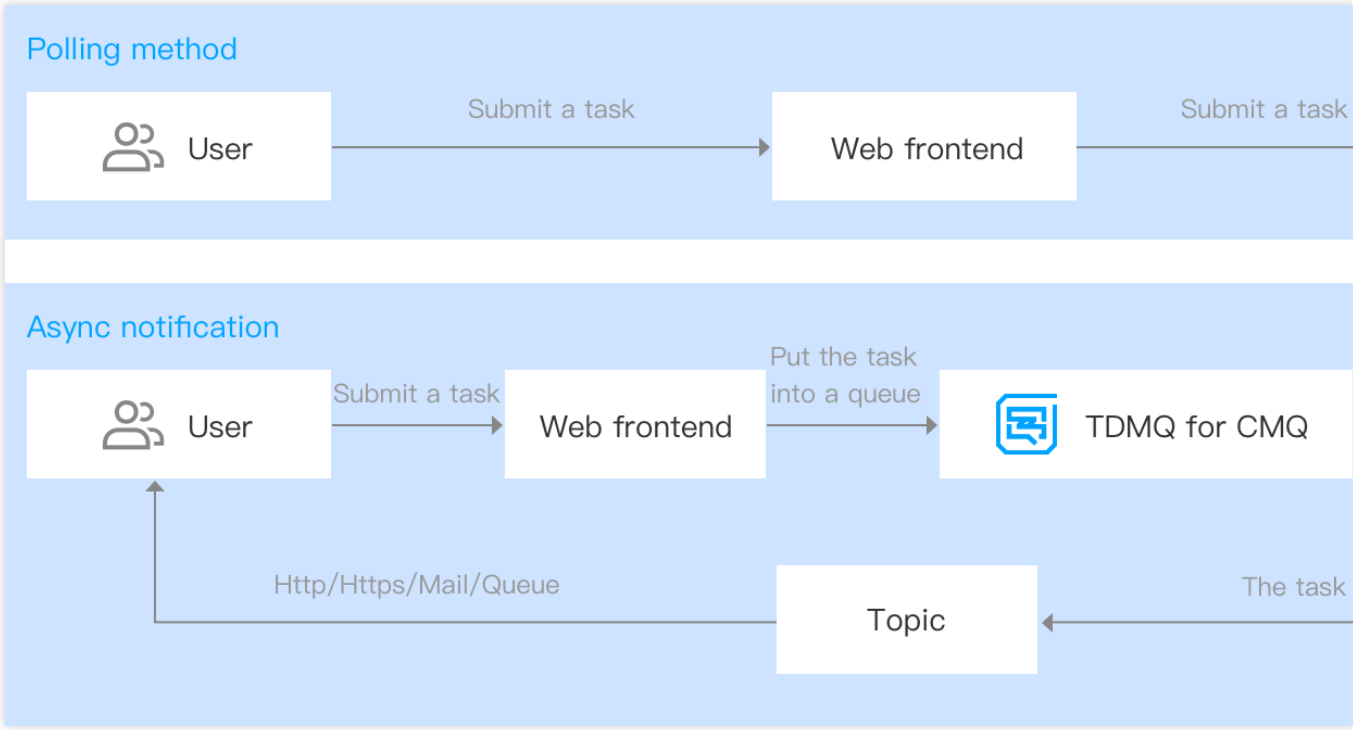
One-to-Many Production

There are many communications between system components or applications, each of which needs to maintain their own network connections. Plus, the types of communicated content are diverse, further making it difficult and costly to implement such communications. In this case, TDMQ for CMQ can support a single producer for several subscribers and deliver messages asynchronously. It can also enable clients to consume certain messages through message filtering.



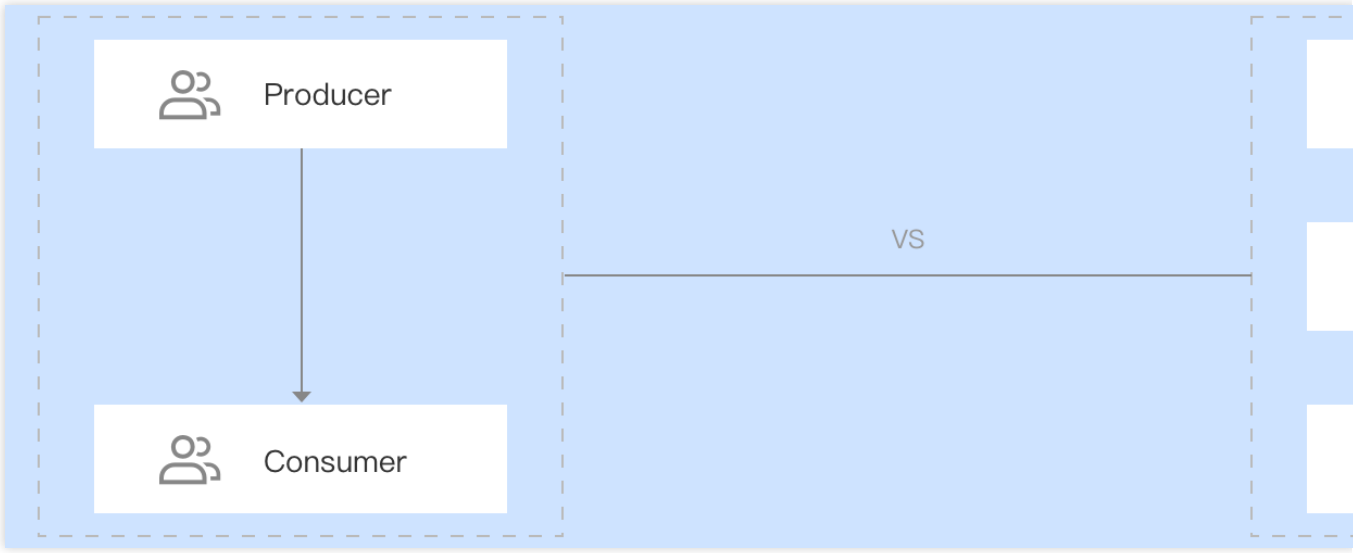
Async Notification

When a message is sent, it cannot be delivered reliably if the recipient is unavailable due to power outage, server downtime, or CPU overload. If TDMQ for CMQ is used, the message will be persistently stored in a queue until the recipient becomes available to consume it successfully.



System Decoupling

Your business receives contents submitted by users, stores some data in your own system, and forwards the processed data to other business applications (such as data analysis and storage systems). These system components or applications are closed coupled with each other. In contrast, if TDMQ for CMQ is used, the recipient and sender are imperceptible to each other's information, which greatly reduces the coupling degree. This is especially helpful when the controllability of dependent components is low.

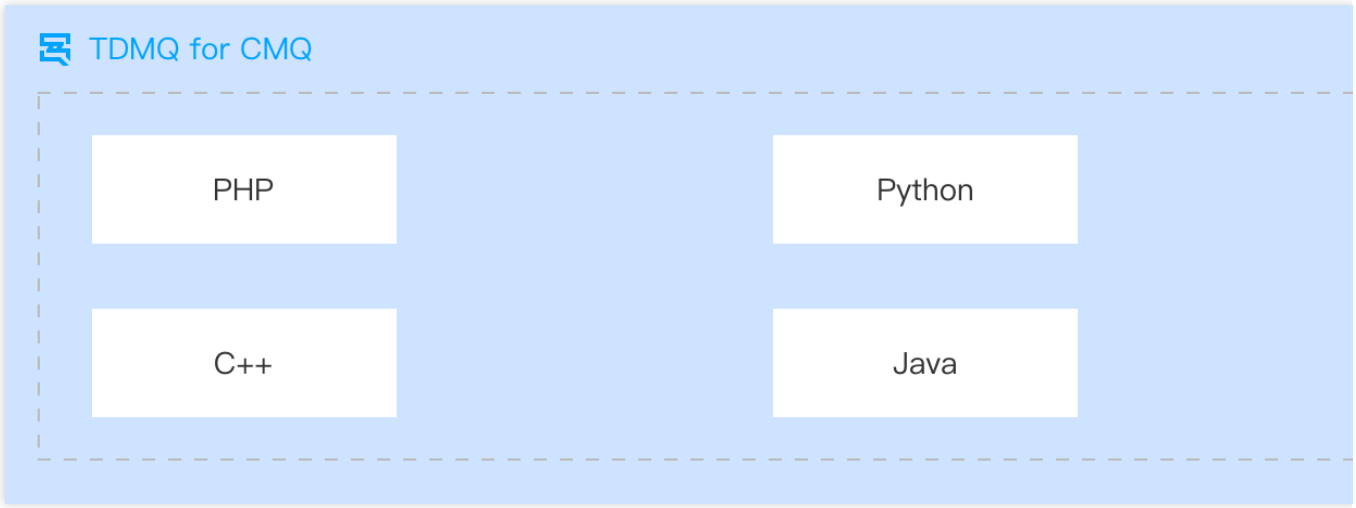


Peak Shifting

During flash or group sales, the high number of users often causes temporary traffic spikes and poses huge challenges to each backend application system. In this case, TDMQ for CMQ can act as a buffer to centrally collect the suddenly increased requests in the upstream, allowing downstream businesses to consume the request messages based on their actual processing capacities.

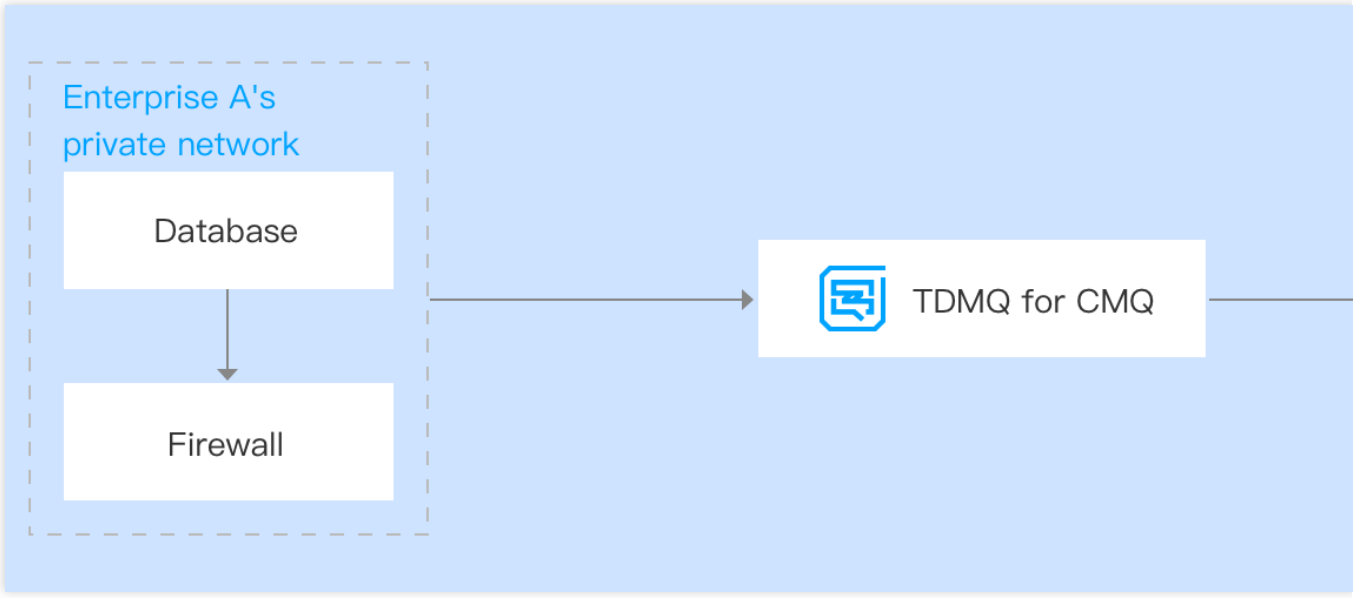
Unified Platform

As an ecommerce system grows over time, if the order system (order_module), inventory system (inventory_module), and shipping system use Java, Erlang, and Python architectures respectively, developers may need to spend much time maintaining some redundant code to sustain the communications between the modules. After TDMQ for CMQ is introduced, such troubles caused by the differences between platforms and between programming languages can be eliminated.



Cross-user Data Exchange

Two services need to communicate with each other when their networks cannot interconnect or the application route information (such as IP and port) is variable. For example, if two Tencent Cloud services need communication but don't know each other's address, they can agree on the queue name in TDMQ for CMQ, so that one service can send messages to the queue and the other can receive messages from it and thus implement data exchange.



Use Limits

Last updated：2024-01-03 10:15:44

This document lists the limits of certain metrics and performance in TDMQ for CMQ. Be careful not to exceed the limits during use so as to avoid exceptions.

Queue

Limit	Description
Maximum number of queues per root account	10,000
Message lifecycle	1 minute–15 days
Long polling wait time for message receipt	0–30 seconds
Hidden duration of fetched message	1 second–12 hours
Maximum message size	1 MB
Maximum number of heaped messages	1 million–100 million per queue
Production QPS limit	5,000
Consumption QPS limit	5,000
Traffic limit	400 Mbps

Topic

Limit	Description
Maximum number of topics per root account	10,000
Message lifecycle	24 hours by default, which cannot be modified currently
Maximum message size	1 MB
Production QPS limit	5,000
Consumption QPS limit	5,000
Traffic limit	400 Mbps

Subscriber

Limit	Description
Maximum number of subscribers per topic	1,000

Note:

If you need higher production QPS, consumption QPS, and traffic, you can submit a ticket to apply for increasing the upper limits.

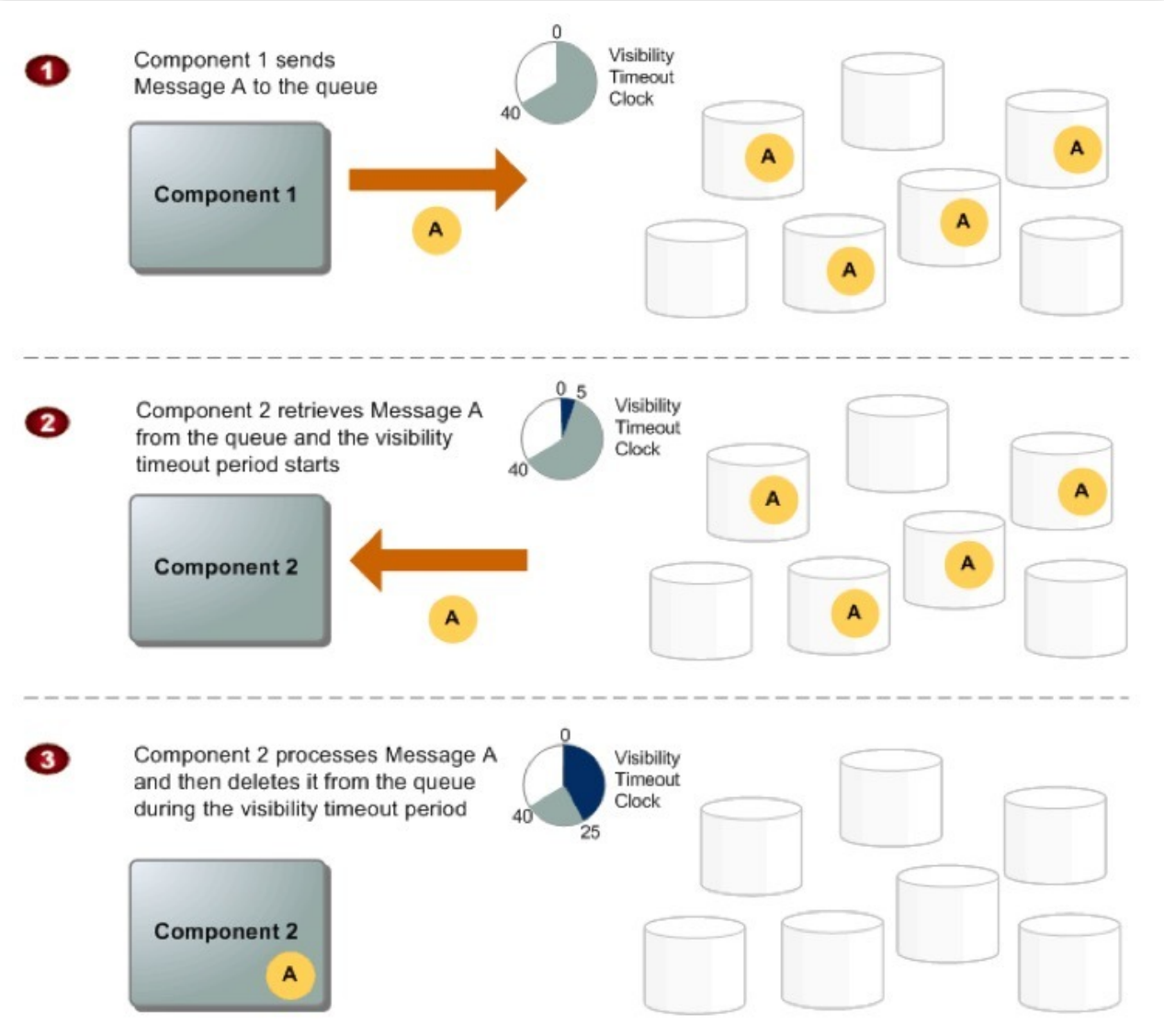
Public Network Bandwidth

Limit	Description
Maximum public network bandwidth	5 Mbps by default

Basic Concepts
Message Lifecycle

Last updated：2024-01-03 10:15:44

When a general message is sent to a general message queue, its initial status is **Active**. After it is fetched out, its status will become **Inactive** within the time period specified by `VisibilityTimeout` . If the message is not deleted after the period specified by `VisibilityTimeout` elapses, its status will become **Active** again; otherwise, its status will become **Deleted**. The maximum retention period of the message is subject to the `MessageRetentionPeriod` attribute value specified when the queue is created. After this period elapses, the message will become **Expired** and be repossessed.
Consumers can read **Active** messages only, which ensures that a message will not be repeatedly consumed simultaneously but can be repeatedly consumed sequentially.



Component 1 sends message A, which has multiple redundancies across TDMQ for CMQ servers, to a queue.

After getting ready to process messages, component 2 will retrieve messages from the queue, and message A will be returned. When being processed, message A still stays in the queue. Within the **hidden duration of fetched messages**, other businesses cannot get message A.

Component 2 can delete message A from the queue to avoid receiving and processing it again after the **hidden duration of fetched messages** elapses. It can also retain message A so that other businesses can consume message A repeatedly.

Queue and Message Identifiers

Last updated：2024-01-03 10:15:44

When using TDMQ for CMQ, you need to get familiar with the following three identifiers: queue name, message ID, and receipt handler.

Queue Name

When creating a queue, you need to give it a unique name in the current region. Queue names can be the same in different regions. TDMQ for CMQ uniquely identifies a queue based on its region and name. When performing an operation on a queue, you always need to provide these two parameters.

Message ID

Each message will receive a message ID in the format of `Msg-XXXXXXXX` assigned by the Tencent Cloud system. It is used to identify a message and can be returned to you through the `SendMessage` API request. It should be noted that the message receipt handler instead of message ID is required during message deletion.

Receipt Handler

Whenever a message is received from a queue, a receipt handler of the message will also be received, which is always relevant to the message receipt operation rather than the message itself. To delete a message or modify message attributes, the receipt handler instead of the message ID needs to be provided, which means that a message can be deleted/modified only after it is received.

Note:

If a message is received more than once, the obtained receipt handler will differ with each receipt. When a message deletion request is initiated, the latest received receipt handler must be provided; otherwise, the message may not be deleted.