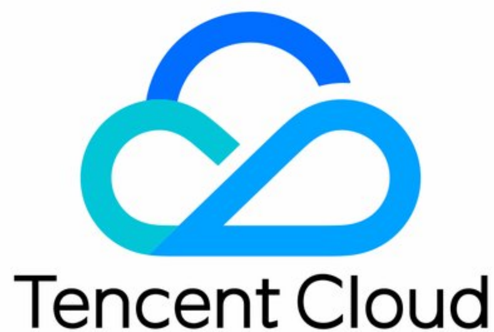
TDMQ for RocketMQ

Product Introduction

Product Documentation



©2013-2022 Tencent Cloud. All rights reserved.

Copyright Notice

©2013-2019 Tencent Cloud. All rights reserved.

Copyright in this document is exclusively owned by Tencent Cloud. You must not reproduce, modify, copy or distribute in any way, in whole or in part, the contents of this document without Tencent Cloud's the prior written consent.

Trademark Notice

All trademarks associated with Tencent Cloud and its services are owned by Tencent Cloud Computing (Beijing) Company Limited and its affiliated companies. Trademarks of third parties referred to in this document are owned by their respective proprietors.

Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

Contents

- Product Introduction
  - Overview
  - Strengths
  - Message Type
    - General Message
    - Scheduled Message and Delayed Message
    - Sequential Message
  - Features
    - Message Filtering
    - Message Retry and Dead Letter Mechanisms
    - Clustering Consumption and Broadcasting Consumption
  - Use Cases
  - Use Limits
  - Relevant Concepts

©2013-2022 Tencent Cloud. All rights reserved.

Product Introduction

Overview

Last updated：2022-02-11 14:33:12

TDMQ for RocketMQ is Tencent's proprietary message queue service. It is compatible with all components and principles of Apache RocketMQ, and supports connection to RocketMQ 4.6.1 or above without any modifications. It also has the underlying benefits of computing-storage separation and flexible scaling.

TDMQ for RocketMQ helps you better tackle traffic spikes in various marketing events. It is very suitable for scenarios with high requirements for sequential and transactional messages and widely used in ecommerce transaction, financial settlement, and other fields.

©2013-2022 Tencent Cloud. All rights reserved.

Strengths

Last updated：2022-02-11 14:33:25

Open-Source Component Compatibility

TDMQ for RocketMQ is compatible with open-source RocketMQ 4.3.0 and above and supports access from open-source clients in Java, C, C++, Go, and other programming languages.

Computing-Storage Separation

The service (broker) and storage (bookie) of TDMQ for RocketMQ are separated from each other. Its overall architecture adopts a stateless cloud native design to natively support on-demand usage and scaling, delivering a more serverless use experience with underlying resources imperceptible to users.

Resource Isolation

TDMQ for RocketMQ offers a multi-level resource structure that allows for both virtual isolation based on namespace and physical isolation at the cluster level. You may enable namespace-level permission verification to distinguish clients in different environments, which is simple and flexible.

Segmented Storage

As TDMQ for RocketMQ stores message data as segments, issues like data skew are rare. Rebalancing will not be triggered when nodes are added or deleted due to scaling or server fault, thus avoiding a significant reduction in cluster throughput.

Rich Diversity of Message Types

TDMQ for RocketMQ supports multiple message types such as general, sequential, delayed, and distributed transaction messages. It also supports message retry and the dead letter mechanism, fully meeting the requirements in various business scenarios.

©2013-2022 Tencent Cloud. All rights reserved.

High Performance

A single TDMQ for RocketMQ server can sustain a production/consumption throughput of up to 10,000 messages. With the distributed architecture and stateless services, the cluster can be scaled horizontally to increase the cluster throughput.

©2013-2022 Tencent Cloud. All rights reserved.

Message Type

General Message

Last updated：2022-02-11 14:33:40

General message is a basic message type, where a message is delivered to the specified topic by the producer and then consumed by the consumer subscribed to the topic. As general messages are not sequential in a topic, you can use multiple partitions to improve the message production/consumption efficiency, and they deliver the best performance when the throughput is huge.

General message is different from scheduled, delayed, sequential, and transactional message. The topics corresponding to these types of messages cannot be mixed and can only be used to send and receive messages of the same type. For example, a general message topic can only be used to send and receive general messages but not other types of messages.

©2013-2022 Tencent Cloud. All rights reserved.

Scheduled Message and Delayed Message

Last updated：2022-06-07 11:25:55

This document describes the concepts and usage of scheduled message and delayed message in TDMQ for RocketMQ.

Concepts

- **Scheduled message**: after a message is sent to the server, the business may want the consumer to receive it at a later time point rather than immediately. This type of message is called "scheduled message".
- **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".

Actually, delayed message can be regarded as a special type of scheduled message, which is essentially the same thing.

Usage

Apache RocketMQ does not provide an API for you to freely set the delay time. In order to ensure compatibility with the open-source RocketMQ client, TDMQ for RocketMQ has designed a method to specify the message sending time by adding the property key-value pair to the message. You only need to add the `__STARTDELIVERTIME` property value to the `property` of the message that needs to be sent at a scheduled time (within 40 days). For delayed messages, you can first calculate the time point for scheduled sending and then send them as scheduled messages.

A code sample is given below to show how to use scheduled and delayed messages in TDMQ for RocketMQ. You can also view the complete sample code >>

Scheduled message

To send a scheduled message, simply write a standard millisecond timestamp to the `__STARTDELIVERTIME` property before sending it.

```
Message msg = new Message("test-topic", ("message content").getBytes(StandardCharsets.UTF_8));
// Set the message to be sent at 00:00:00 on 2021-10-01
try {
long timeStamp = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").parse("2021-10-01 00:00:00").getTime();
```

©2013-2022 Tencent Cloud. All rights reserved.

```
// Set `__STARTDELIVERTIME` into the property of `msg`
msg.putUserProperty("__STARTDELIVERTIME", String.valueOf(timeStamp));
SendResult result = producer.send(msg);
System.out.println("Send delay message: " + result);
} catch (ParseException e) {
// TODO adds the method for handling the timestamp parsing failure
e.printStackTrace();
}
```

Delayed message

For a delayed message, its scheduled sending time point is first calculated by `System.currentTimeMillis() + delayTime`, and then it is sent as a scheduled message.

```
Message msg = new Message("test-topic", ("message content").getBytes(StandardCharsets.UTF_8));
// Set the message to be sent after 10 seconds
long delayTime = System.currentTimeMillis() + 10000;
// Set `__STARTDELIVERTIME` into the property of `msg`
msg.putUserProperty("__STARTDELIVERTIME", String.valueOf(delayTime));
SendResult result = producer.send(msg);
System.out.println("Send delay message: " + result);
```

Use Limits

- When using delayed messages, make sure that the time on the client is in sync with the time on the server; otherwise, there will be a time difference.
- There is a precision deviation of about 1 second for scheduled and delayed messages.
- The maximum time range for scheduled and delayed messages are both 40 days.
- When using scheduled messages, you need to set a time point after the current time; otherwise, the message will be sent to the consumer immediately.
- Delayed message in TMDQ for RocketMQ is not compatible with delayed message of RocketMQ ONS client, but they achieve the same results.

©2013-2022 Tencent Cloud. All rights reserved.

Sequential Message

Last updated：2022-02-11 14:34:05

This document describes the concept and usage of sequential message in TDMQ for RocketMQ.

Concepts

Sequential messages include globally sequential messages and partitionally sequential messages:

- **Globally sequential message**: the most distinctive characteristic of the globally sequential message type is that messages are consumed strictly in the order they are delivered by the producer. Therefore, it uses a single partition to process messages, and you cannot customize the number of partitions. It has a lower performance.

- **Partitionally sequential message**: compared with general messages, partitionally sequential messages are sequential in a particular partition; that is, in the same partition, the consumer consumes messages strictly in the order they are delivered to the partition by the producer. This message type retains the partitioning mechanism to improve the performance while guaranteeing a certain sequence, but it cannot guarantee the sequence across different partitions.

The comparison between sequential message and general message is as follows:

Message Type	Consumption Sequence	Performance	Use Cases
General message	No sequence	Highest	Huge-Throughput use cases with no requirements for production and consumption sequence

©2013-2022 Tencent Cloud. All rights reserved.

Message Type	Consumption Sequence	Performance	Use Cases
Partitionally sequential message	All messages in the same partition follow the First In, First Out (FIFO) rule	High	High-Throughput use cases that require publishing and consuming messages in the same partition in strict accordance with the FIFO rule
Globally sequential message	All messages in the same topic follow the First In, First Out (FIFO) rule	Average	Average-Throughput use cases that require publishing and consuming all messages in strict accordance with the FIFO rule

©2013-2022 Tencent Cloud. All rights reserved.

Features

Message Filtering

Last updated：2022-02-11 14:34:21

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

Feature Overview

In message filtering, a message producer configures message attributes to group messages before sending them to a topic, and a consumer subscribed to the topic uses message attributes to filter the messages so that only eligible messages are delivered to the consumer for consumption.

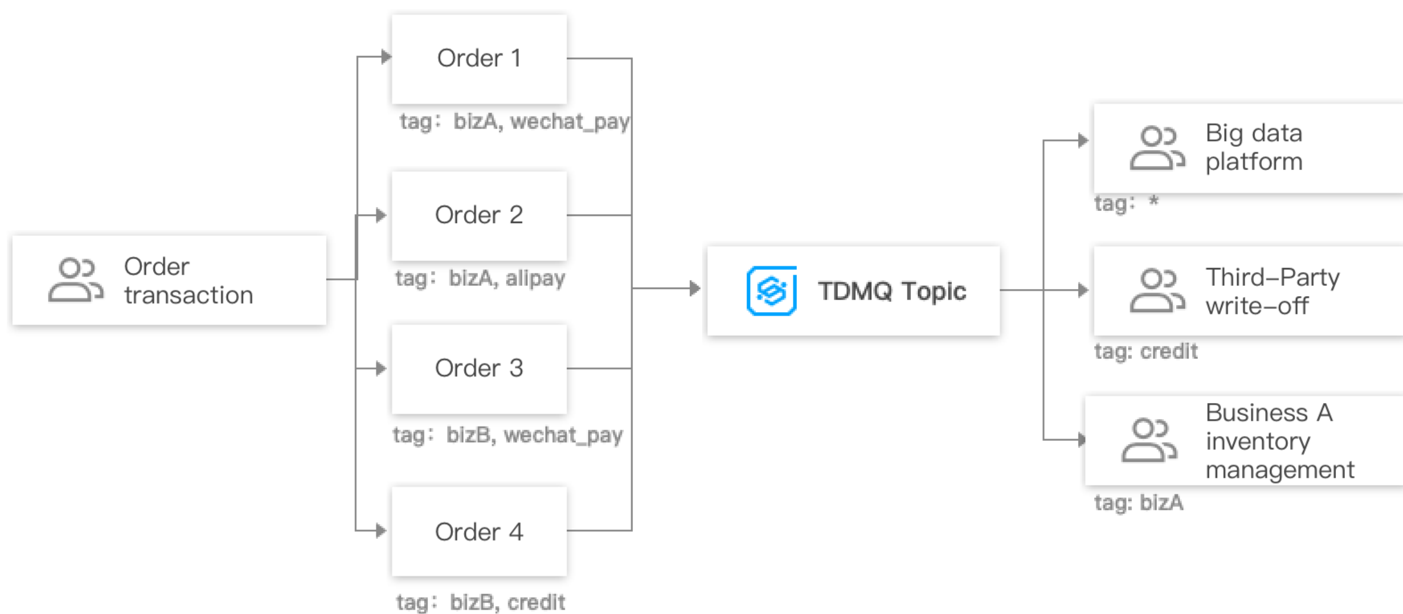
If a consumer sets no filter conditions when subscribing to a topic, no matter whether filter attributes are set during message sending, all messages in the topic will be delivered to the consumer for consumption.

Use Cases

Generally, messages with the same business attributes are stored in the same topic; for example, when an order transaction topic contains messages of order placement transactions, payment transactions, and delivery transactions, and if you want to consume only one type of transaction messages in your business, you can filter them on the client, but this will waste bandwidth resources.

To solve this problem, TDMQ supports filtering on the broker. You can set one or more tags during message production and subscribe to specified tags during consumption.

©2013-2022 Tencent Cloud. All rights reserved.



Usage

Sending message

You must specify tags for each message when sending it.

```
Message msg = new Message("TOPIC","TagA","Hello world".getBytes());
```

Subscribing to message

- Subscribe to all tags:
  If a consumer wants to subscribe to all types of messages under a topic, an asterisk (*) can be used to represent all tags.

```
consumer.subscribe("TOPIC", "*", new MessageListener() {
public Action consume(Message message, ConsumeContext context) {
System.out.println(message.getMsgID());
return Action.CommitMessage;
}
});
```

- Subscribe to one tag:
  If a consumer wants to subscribe to a certain type of messages under a topic, the tag should be specified clearly.

©2013-2022 Tencent Cloud. All rights reserved.

```
consumer.subscribe("TOPIC", "TagA", new MessageListener() {
public Action consume(Message message, ConsumeContext context) {
System.out.println(message.getMsgID());
return Action.CommitMessage;
}
});
```

- Subscribe to multiple tags:

  If a consumer wants to subscribe to multiple types of messages under a topic, two vertical bars (||) should be added between two tags for separation.

```
consumer.subscribe("TOPIC", "TagA||TagB", new MessageListener() {
public Action consume(Message message, ConsumeContext context) {
System.out.println(message.getMsgID());
return Action.CommitMessage;
}
});
```

©2013-2022 Tencent Cloud. All rights reserved.

Message Retry and Dead Letter Mechanisms

Last updated：2022-02-11 14:34:32

This document describes the message retry and dead letter mechanisms and their usages in TDMQ for RocketMQ.

Feature Overview

When a message is consumed for the first time by a consumer and fails to get a normal response, TDMQ for RocketMQ will automatically retry delivering this message through the message retry mechanism until it is consumed successfully. When the number of retries reaches the specified value but the message is still not consumed successfully, retry will stop, and the message will be delivered to the dead letter queue.

After the message enters the dead letter queue, TDMQ for RocketMQ can no longer process it automatically. At this point, human intervention is generally required. You can write a dedicated client to subscribe to the dead letter queue to process such messages.

> Note：
> The broker will automatically retry in the cluster consumption mode but not the broadcast consumption mode.

The following results are considered as consumption failure, and the message will be retried accordingly:

1. The consumer returns `ConsumeConcurrentlyStatus.RECONSUME_LATER` .
2. The consumer returns `null` .
3. The consumer actively/passively throws an exception.

Number of Retries

When a message needs to be retried in TDMQ for RocketMQ, set the "messageDelayLevel" parameter as follows to configure the number of retries and retry intervals:

```
messageDelayLevel=1s 5s 10s 30s 1m 2m 3m 4m 5m 6m 7m 8m 9m 10m 20m 30m 1h 2h
```

The number of retries and retry intervals have the following relationships:

Retry Number	Interval Retry	Retry Number	Interval Retry

©2013-2022 Tencent Cloud. All rights reserved.

Retry Number	Interval Retry	Retry Number	Interval Retry
1	1 second	10	6 minutes
2	5 seconds	11	7 minutes
3	10 seconds	12	8 minutes
4	30 seconds	13	9 minutes
5	1 minutes	14	10 minutes
6	2 minutes	15	20 minutes
7	3 minutes	16	30 minutes
8	4 minutes	17	1 hour
9	5 minutes	18	2 hours

Usage

If you use `DefaultMQProducer` to send a general message, you can set the maximum number retries allowed if message delivery fails and develop the business retry logic flexibly based on the timeout period. The usage is as follows:

```
/**Set the maximum number retries after message sending fails*/
public void setRetryTimesWhenSendFailed(int retryTimesWhenSendFailed) {
this.retryTimesWhenSendFailed = retryTimesWhenSendFailed;
}
/**Send the message synchronously and specify the timeout period*/
public SendResult send(Message msg,
long timeout) throws MQClientException, RemotingException, MQBrokerException, Int
erruptedException {
return this.defaultMQProducerImpl.send(msg, timeout);
}
```

Below is the sample code of retrying for three times when the producer fails to send a message within 3s:

```
/**Send the message synchronously and retry three times if the message fails to b
e sent within 3s*/
DefaultMQProducer producer = new DefaultMQProducer("DefaultProducerGroup");
producer.setRetryTimesWhenSendFailed(3);
producer.send(msg, 3000L);
```

©2013-2022 Tencent Cloud. All rights reserved.

©2013-2022 Tencent Cloud. All rights reserved.

Clustering Consumption and Broadcasting Consumption

Last updated：2022-07-04 16:38:20

This document describes the features and use cases of clustering consumption and broadcasting consumption in TDMQ for RocketMQ.

Feature Description

- Clustering: If the clustering mode is used, any message only needs to be processed by any consumer in the cluster.
- Broadcasting: If the broadcasting mode is used, each message will be pushed to all registered consumers in the cluster to ensure that the message is consumed by each consumer at least once.

Use Cases

Clustering consumption is suitable for scenarios where each message only needs to be processed once.

Broadcasting consumption is suitable for scenarios where each message needs to be processed by each consumer in the cluster.

Sample Code

- **Cluster subscription**
  All consumers identified by the same group ID will evenly share messages for consumption. For example, if a topic has nine messages, and a group ID identifies three consumer instances, then each instance will consume only three messages evenly in the clustering consumption mode.

```
// Set the cluster subscription mode (which is the default mode if you don't specify one)
properties.put(PropertyKeyConst.MessageModel, PropertyValueConst.CLUSTERING);
```

- **Broadcast subscription**
  A message will be consumed once by all consumers identified by the same group ID. In the broadcasting consumption mode, for example, if a topic has nine messages and a group ID identifies three consumer instances, each instance will consume nine messages.

©2013-2022 Tencent Cloud. All rights reserved.

```
// Set the broadcast subscription mode
properties.put(PropertyKeyConst.MessageModel, PropertyValueConst.BROADCASTING);
```

Note：

You need to ensure that all consumer instances under the same group ID have the same subscription relationships.

©2013-2022 Tencent Cloud. All rights reserved.

Use Cases

Last updated：2022-02-11 14:34:54

Async Decoupling

The transaction engine is the core system of Tencent billing. The data of each transaction order needs to be monitored by dozens of downstream business systems, including item price approval, delivery, reward point, and stream computing analysis. Such systems use different message processing logic, making it impossible for a single system to adapt to all associated business. In this case, TDMQ for RocketMQ can implement efficient async communication and application decoupling to ensure the business continuity of the primary site.

Peak Shifting

Companies hold promotional campaigns such as new product launch and festival red packet grabbing from time to time, which often cause temporary traffic spikes and pose huge challenges to each backend application system. In this case, TDMQ for RocketMQ 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.

Sequential Message Sending/Receiving

Sequential messages are used in some business scenarios, such as order creation, payment, delivery, and refund of in-app/game items, which are all strictly executed in sequence. Similar to the First In, First Out (FIFO) principle, TDMQ for RocketMQ offers a sequential message feature dedicated to such scenarios to ensure message FIFO.

Consistency of Distributed Transactions

A billing system often has a long transaction linkage with a significant chance of error or timeout. TDMQ for RocketMQ's automated repush and abundant message retention features can be used to provide transaction compensation, and the eventual consistency of payment tips notifications and transaction pushes can also be achieved through TDMQ for RocketMQ.

Distributed Cache Sync

During sales and promotions, there are a wide variety of products with frequent price changes. When users query item prices multiple times, the cache server's network interface may be fully loaded, which makes page opening slower. After the broadcast consumption mode of TDMQ for RocketMQ is adopted, a message will be consumed by all nodes once, which is equivalent to syncing the price information to each server as needed in place of the cache.

Big Data Analysis

Data creates value in the "flow". Most traditional data analysis are based on batch computing models, which means they cannot analyze data in real time. In contrast, TDMQ for RocketMQ can easily implement real-time analysis of business data when combined with a stream computing engine.

©2013-2022 Tencent Cloud. All rights reserved.

Use Limits

Last updated：2022-07-04 16:38:20

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

Cluster

Limit	Description
Maximum number of clusters per region	5
Cluster name length	3–64 characters

Namespace

Limit	Description
Maximum number of namespaces per cluster	10
Maximum TPS per namespace	8,000
Maximum bandwidth (production + consumption) per namespace	400 Mbps
Namespace name length	3–64 characters

Topic

Limit	Description
Maximum number of topics per cluster	1,000
Topic name length	3–64 characters
Maximum number of producers per topic	1,000
Maximum number of consumers per topic	500

Group

Limit	Description
Maximum number of groups per cluster	10,000

©2013-2022 Tencent Cloud. All rights reserved.

Limit	Description
Group name length	3–64 characters

Message

Limit	Description
Maximum message retention period	15 days
Maximum message delay	40 days
Maximum message size	5 MB
Consumption offset reset	15 days

©2013-2022 Tencent Cloud. All rights reserved.

Relevant Concepts

Last updated：2022-02-11 14:35:15

This document lists the common concepts in TDMQ for RocketMQ and their definitions.

General Message

General message is a basic message type, where a message is delivered to the specified topic by the producer and then consumed by the consumer subscribed to the topic. As general messages are not sequential in a topic, you can use multiple partitions to improve the message production/consumption efficiency, and they deliver the best performance when the throughput is huge.

Sequential Message

- **Partitionally sequential message**: compared with general messages, partitionally sequential messages are sequential in a particular partition; that is, in the same partition, the consumer consumes messages strictly in the order they are delivered to the partition by the producer. This message type retains the partitioning mechanism to improve the performance while guaranteeing a certain sequence, but it cannot guarantee the sequence across different partitions.

- **Globally sequential message**: the most distinctive characteristic of the globally sequential message type is that messages are consumed strictly in the order they are delivered by the producer. Therefore, it uses a single partition to process messages, and you cannot customize the number of partitions. It has a lower performance than the other two message types.

Dead Letter Message

A dead letter message is a message that cannot be consumed normally. When you create a subscription in TDMQ (to subscribe a consumer to a topic), a dead letter queue will be automatically created to process messages of this type.

Retry Letter Queue

A retry letter queue is designed to ensure that messages are consumed normally. If no normal response is received after a message is consumed by the consumer for the first time, it will enter the retry letter queue, and after a specified number of failed retries, it will be delivered to the dead letter queue.

In actual scenarios, messages may not be processed promptly due to temporary issues such as network jitter and service restart, and the retry mechanism of the retry letter queue can be a good solution in this case.

Dead Letter Queue

©2013-2022 Tencent Cloud. All rights reserved.

A dead letter queue is a special type of message queue used to centrally process messages that cannot be consumed normally. If a message cannot be consumed after a specified number of retries in the retry letter queue, TDMQ will determine that the message cannot be consumed under the current situation and deliver it to the dead letter queue.

In actual scenarios, messages may not be consumed due to service downtime or network disconnection. In this case, they will not be discarded immediately; instead, they will be persisted by the dead letter queue. After fixing the problem, you can create a consumer subscription to the dead letter queue to process such messages.

Cluster Consumption

Cluster consumption is suitable for scenarios where each message only needs to be processed once.

Broadcast Consumption

Broadcast consumption is suitable for scenarios where each message needs to be processed by each consumer in the cluster.

©2013-2022 Tencent Cloud. All rights reserved.