Message Queue

Advanced Features

Advanced Features

Message retry

Retries for ordered messages

For ordered messages, when the consumer failed to consume a message, MQ will automatically retry sending the message continuously at an interval of one second. This may lead to the occurrence of consumption blocking. Therefore, when you use ordered messages, ensure that the application can monitor and handle the consumption failures promptly to prevent blocking of message consumption.

Retries for unordered messages

Unordered messages include normal, scheduled, delayed, and transactional messages. When a consumer fails to consume such messages, you can set the returned status to achieve the same effect as message retry.

Unordered message retry takes effect only in clustering consumption mode. In broadcasting consumption mode, if message consumption fails, failed messages are not re-consumed, and new messages are consumed instead.

Note: The following information is applicable to unordered messages only.

Number of retries

MQ allows a maximum of 16 retries for each message by default, and the intervals for each retry is as follows:

Retry number	Interval	Retry number	Interval
1	10s	9	7 min
2	30s	10	8 min
3	1 min	11	9 min
4	2 min	12	10 min
5	3 min	13	20 min

6	4 min	14	30 min
7	5 min	15	1 h
8	6 min	16	2 h

If the message fails after 16 retries, it will not be delivered. Strictly according to the intervals above, if the consumption of a message continuously fails, there will be 16 retries within 4 hours and 46 minutes, after which the message will not be delivered.

Note: No matter how many retries there are for a message, the message ID will not be changed.

Configuration method

Configure retries after a message consumption failure

In clustering consumption mode, a message retry is expected after the message consumption failure, which needs to be configured in the implementation of the message listener interface (three methods available):

- Return Action.ReconsumeLater (recommended).
- Return null.
- Throw an exception.

Sample code

```
public class MessageListenerImpl implements MessageListener {

@Override
public Action consume(Message message, ConsumeContext context) {
//Method 3: The message process logic throws an exception, and the message will be retried.
doConsumeMessage(message);
//Mode 1: Return Action.ReconsumeLater, and the message will be retried.
return Action.ReconsumeLater;
//Mode 2: Return null, and the message will be retried.
return null;
//Mode 3: Directly throw an exception, and the message will be retried.
throw new RuntimeException("Consumer Message exception");
}
}
```

Disable retries after a message consumption failure

In clustering consumption mode, no message retry is expected after a message consumption failure. The possible exception thrown by the consumption logic needs to be captured and Action.CommitMessage is returned. After that, the message will not be retried.

Sample code

```
public class MessageListenerImpl implements MessageListener {

@Override
public Action consume(Message message, ConsumeContext context) {
try {
doConsumeMessage(message);
} catch (Throwable e) {
//Capture all exceptions in the consumption logic, and return Action.CommitMessage;
return Action.CommitMessage;
}
//The message processing is normal, and Action.CommitMessage is directly returned
return Action.CommitMessage;
}
}
```

Customize the maximum number of retries

To customize the log configuration of the MQ client, you must upgrade the TCP Java SDK to 1.2.2 or later.

MQ allows you to set the maximum number of retries when the consumer is started. The retry interval complies with the following policies:

- If the maximum number of retries is less than or equal to 16, the retry interval is as described in the preceding table.
- If the maximum number of retries is greater than 16, the interval of the 17th or later retry is 2 hours.

The configuration method is as follows:

```
Properties properties = new Properties();
//Set the maximum number of message retries for the corresponding group ID to 20
properties.put(PropertyKeyConst.MaxReconsumeTimes,"20");
Consumer consumer =ONSFactory.createConsumer(properties);
```

Note:

- The configuration of the maximum number of message retries applies to all consumer instances with the same group ID.
- If the maximum number of message retries is set for only one of the two consumer instances with the same group ID, the configuration applies to both consumer instances.
- The configuration takes effect by overwriting, that is, the last started consumer instance will overwrite the configuration of previously started instances.

Obtain the number of retries

Upon receiving the message, the consumer can obtain the number of retries with the following method:

```
```

```
public class MessageListenerImpl implements MessageListener {

@Override
public Action consume(Message message, ConsumeContext context) {
//Obtain the number of retries
System.out.println(message.getReconsumeTimes());
return Action.CommitMessage;
}
}
```

Message filtering

This topic describes how consumers filter messages on a MQ broker according to tags. For more information about topics and tags, see Topic and tag best practices.

A tag is a label that classifies messages into different types under a topic. MQ allows consumers to filter messages according to tags, ensuring that the consumers consume messages of types they are concerned with.

The following figure shows an example in the e-commerce transaction scenario. The process from placing an order to receiving the product by the customer will produce a series of messages, such as order message, payment message, and logistics message. These messages will be sent to the queue with the topic Trade_Topic and received by different systems, such as the payment system, logistics system, transaction success rate analysis system, and real-time computing system. Among these systems, the logistics system only receives the logistics message, and the real-time computing system receives all the messages related to the transaction (order, payment, and logistics).

Note: To classify messages, you can create multiple topics, or create multiple tags under the same topic. However, in general, there is no relationship between messages in different topics, and tags are used to distinguish related messages within the same topic, such as the relationship between the full set and the subset, and the relationship between the processes in sequence.

Sample code

Sending messages

A tag must be specified for each message before it is sent:

```
Message msg = new Message("MQ_TOPIC","TagA","Hello MQ".getBytes());
```

Consumption method - 1

If a consumer needs to subscribe to all types of messages under a topic, the * symbol can be used to

represent the tags:

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

Consumption method - 2

If a consumer needs to subscribe a certain type of messages under a topic, the tag should be specified:

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

Consumption method - 3

If a consumer needs to subscribe to messages of multiple types under a topic, separate tags with separators (||):

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

Consumption method - 4 (error example)

If a consumer subscribes to messages of the tags under a topic for multiple times, the tags subscribed to the previous time prevail:

```
//In the following error code, a consumer can receive only messages with TagB under MQ_TOPIC and cannot
receive messages with TagA:
consumer.subscribe("MQ_TOPIC", "TagA", new MessageListener() {
public Action consume(Message message, ConsumeContext context) {
System.out.println(message.getMsgID());
return Action.CommitMessage;
}
});
consumer.subscribe("MQ_TOPIC", "TagB", new MessageListener() {
public Action consume(Message message, ConsumeContext context) {
System.out.println(message.getMsgID());
return Action.CommitMessage;
}
```

```
});
```

Exactly-Once

This topic describes the concept and typical scenarios of the Exactly-Once delivery semantics of MQ.

Note: For more information about how to send and receive messages through the Exactly-Once delivery semantics, see Use Exactly-Once delivery semantics.

What is Exactly-Once delivery semantics?

Exactly-Once means that a message sent to a message system is processed only once by the consumer, even if the producer re-sends the message.

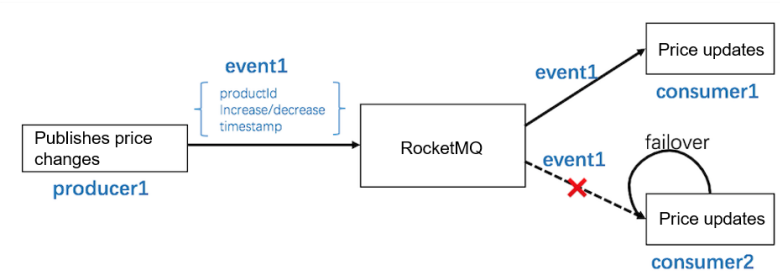
Exactly-Once delivery semantics is an ideal state of message transfer in message systems and stream computing systems. However, this ideal state is rarely implemented in the industry, because it depends on the coordination between the message broker, message client, and user consumption logic. For example, if your consumer client fails after processing a message, the client may process the message again after a restart, because the consumer offset is not synchronized to the message broker.

Exactly-Once delivery semantics is controversial in the industry. Many refer to FLP Impossibility theory or other consistency laws to challenge the validity of Exactly-Once semantics. In fact, in certain scenarios, the Exactly-Once delivery semantics is not difficult to implement. Its implementation only seems complex because the essence of the topic is not described accurately.

To make the consumption result of each message take effect only once in your business system, ensure the consumption idempotence of the same message. The use of Exactly-Once delivery semantics by MQ ensures that the consumption result of a message (the message processing result on the consumer) exists and takes effect only once in the database system. This is the most common business requirement.

Typical scenarios

In an e-commerce system, the upstream real-time computing module releases product price change messages and asynchronously sends them to the downstream product management module. Then the downstream product management module changes the product prices. In this case, consumption idempotence must be ensured for each message. That is, duplicate price change messages take effect only once, preventing repeated changes of prices.



Clustering consumption and broadcasting consumption

This topic introduces concepts of Alibaba MQ clustering consumption and broadcasting consumption, their scenarios, and precautions for use.

Concepts

Alibaba MQ is a messaging system that is based on message publishing and subscription. Consumers subscribe to a topic to retrieve and consume messages. As the subscribers are usually distributed systems that consist of multiple machines deployed in a cluster, Alibaba MQ defines the following terms:

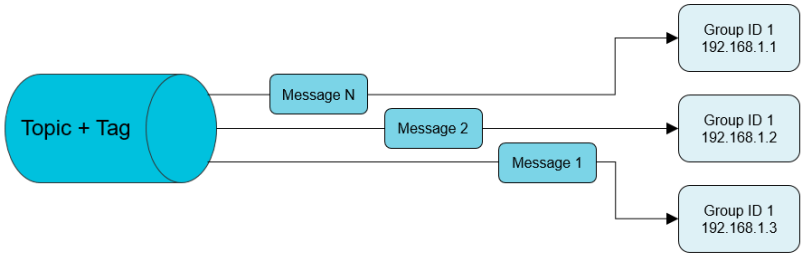
Cluster: Consumers using the same group ID belong to the same cluster. These consumers have identical consumption logic (including tag usage) and can be considered logically as one consumption node.

Clustering consumption: In this model, any message only needs to be processed by any consumer in the cluster.

Broadcasting consumption: In this model, Alibaba MQ broadcasts each message to all clients registered in the cluster to ensure that the message is consumed by each machine at least once.

Scenario comparison

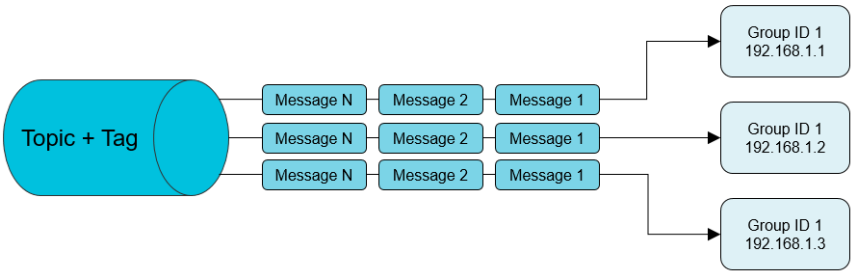
Clustering consumption mode:



Scenarios and usage instructions

- Consumer instances are deployed in a cluster and each message needs to be processed only once.
- The consumption progress is maintained on the broker, so the reliability is high.
- In clustering consumption mode, each message is delivered to only one machine in the cluster for processing. If a message needs to be processed by every machine in the cluster, use the broadcasting consumption mode.
- In clustering consumption mode, there is no guarantee that the re-delivery of a failed message will be routed to the same machine, so no definitive assumptions should be made when processing messages.

Broadcasting consumption mode:



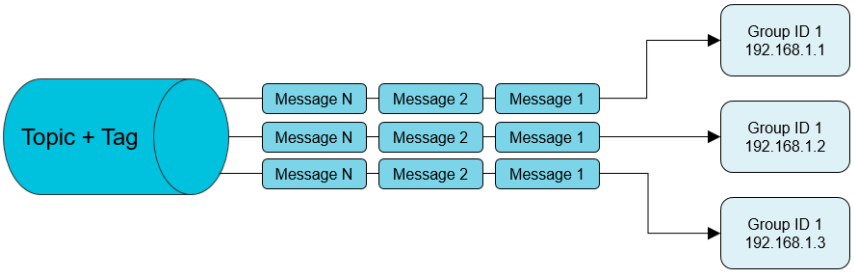
Scenarios and usage instructions

- Ordered messages are not supported in broadcasting consumption mode.
- Resetting consumption offsets is not supported in broadcasting consumption mode.
- Each message needs to be processed by multiple machines with the same logic.
- The consumption progress is maintained at the client. The ratio of repetition is higher than that of the clustering consumption mode.
- In broadcasting consumption mode, Alibaba MQ ensures that each message is consumed by each client at least once, but does not resend messages that fail to be consumed. Therefore, the business side needs to pay attention to consumption failures.
- In broadcasting consumption mode, the consumption starts from the latest message by default when the consumer is started everytime, skipping the messages sent to the Alibaba

MQ server when the consumer is stopped. Consider thouroughly before using this mode.
- In broadcasting consumption mode, each message is processed repeatedly by many clients. Therefore, the clustering consumption mode is recommended.
- Currently, only Java clients support the broadcasting consumption mode.
- In broadcasting consumption mode, the broker does not maintain the consumption progress, so you cannot query message accumulation, set message accumulation alarms, or query subscription in the Alibaba MQ console.

Use the clustering consumption mode to simulate the broadcasting consumption mode:

If the broadcasting consumption mode is needed for your business, you can create multiple group IDs to subscribe to the same topic.



Scenarios and usage instructions

- Each message needs to be processed by multiple machines and the logic of each machine can be either the same or different.
- The consumption progress is maintained on the broker, and the reliability is higher than that in the broadcasting consumption mode.
- For one group ID, one or more consumer instances can be deployed. When multiple consumer instances are deployed, the instances form a cluster to work together for message consumption. Assume that three consumer instances C1, C2, and C3 are deployed for group ID 1. The three instances share the messages sent from the broker to group ID 1. These instances must subscribe to the same topics with the same tags.