Introduction to SOFAStack MQ

Middleware 1.10.0

Document version: V20200810

Ant Technology

Trademarks and Permissions

and other Ant Technology service trademarks are trademarks of Ant Technology.

All other registered trademarks mentioned in this document are the property of their respective holders.

Disclaimer

The content of this document may be updated due to product version upgrade, product adjustment, or other reasons. Ant Technology reserves the right to update the content of this document without notice and will release the updated document through channels authorized by Ant Technology. Please pay attention to the document version change and obtain the latest document through channels authorized by Ant Technology. Ant Technology is not liable for any direct or indirect losses caused by improper use of this document.

Contents

1. What is SOFAStack MQ? 1

1.1 Core concepts 1
1.2 Messaging pattern 1
1.2.1 Producer cluster 2
1.2.2 Consumer cluster 2

2. Advantages 3

3. Product architecture 4

3.1 System deployment architecture 4

4. Features 6

4.1 TCP protocol access 6
4.2 Management tools 6
4.3 Message type 7
4.4 Features 7

5. Scenarios 8

5.1 Asynchronous decoupling 8
5.1.1 Conventional processing 8
5.1.2 Asynchronous decoupling 10
5.2 Data consistency of distributed transactions 10
5.3 Load shifting 12

6. Limitations 13

7. Appendix: Basic terminologies 14

1. What is SOFAStack MQ?

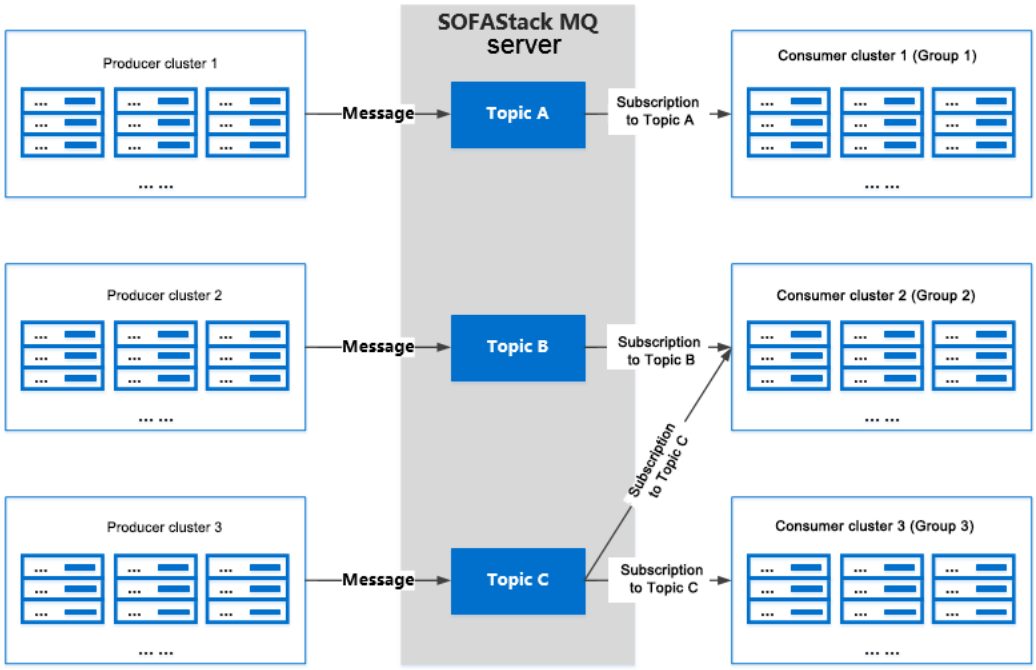
SOFAStack MQ (SOFAMQ) is a distributed message middleware built based on Apache RocketMQ. It is deeply integrated with the financial distributed architecture SOFAStack. SOFAMQ provides distributed application systems with the capabilities of asynchronous decoupling and load shifting. It supports multiple message types such as transactional, sequential, and timed messages. SOFAMQ provides financial-level features such as high reliability, high throughput, and low latency.

1.1 Core concepts

- Topic: the primary message type, to which producers send messages.
- Producer: also known as message publisher, responsible for producing and sending messages to topics.
- Consumer: also known as message subscriber, responsible for receiving and consuming messages from topics.
- Message: a combination of data and (optional) attributes sent by a producer to a topic and finally delivered to a consumer.
- Message attribute: the attributes that can be defined by the producer for a message, including the message key and tag.
- Group: a type of producers or consumers, which usually produce or consume the same type of messages, with the same logic in publishing or subscribing to messages.

1.2 Messaging pattern

SOFAMQ supports the publishing/subscribing pattern, where a message producer application creates a topic and sends messages to it. Then a consumer application subscribes to the topic to receive messages from it. The communication mode can be one-to-many (fan-out), many-to-one (fan-in), and many-to-many.



1.2.1 Producer cluster

A producer cluster is used to indicate the message-sending applications. A producer cluster contains multiple producer instances, which can be multiple servers, or multiple processes of a server, or multiple producer objects of a process.

One producer cluster can send messages to multiple topics. If a producer is down when sending a distributed transactional message, the broker will proactively call back any server from the producer cluster to confirm the transaction status.

1.2.2 Consumer cluster

A consumer cluster is used to indicate message-consuming applications. A consumer cluster contains multiple consumer instances, which can be multiple servers, multiple processes, or multiple consumer objects of a process.

Generally, consumers in a consumer cluster consume an equal share of all the messages. However, in the case of broadcast consumption, each instance in the consumer cluster consumes all the messages.

One consumer cluster corresponds to one group ID, and one group ID can subscribe to multiple topics, as shown in Group 2 in the preceding figure. You can directly set the subscription relationships between groups and topics in SOFAMQ.

2. Advantages

Main advantages of SOFAMQ are as follows:

- **Comprehensive features**
  - Multiple message types: such as normal messages, scheduled messages, partitionally ordered messages, and transactional messages
  - Multiple consumption patterns: Pub/Sub and tag filtering
  - TCP Java SDK
- **Convenient operation and maintenance system**
  - Multi-dimensional queries of messages
  - Full-link message tracing
- **High performance**
  - Accumulation of large amounts of messages
  - End-to-end latency in milliseconds
  - Capability of processing tens of millions of concurrent tasks
- **Service reliability**
  - 99.9% service availability
  - 99.99999% data reliability
  - Message re-delivery mechanism

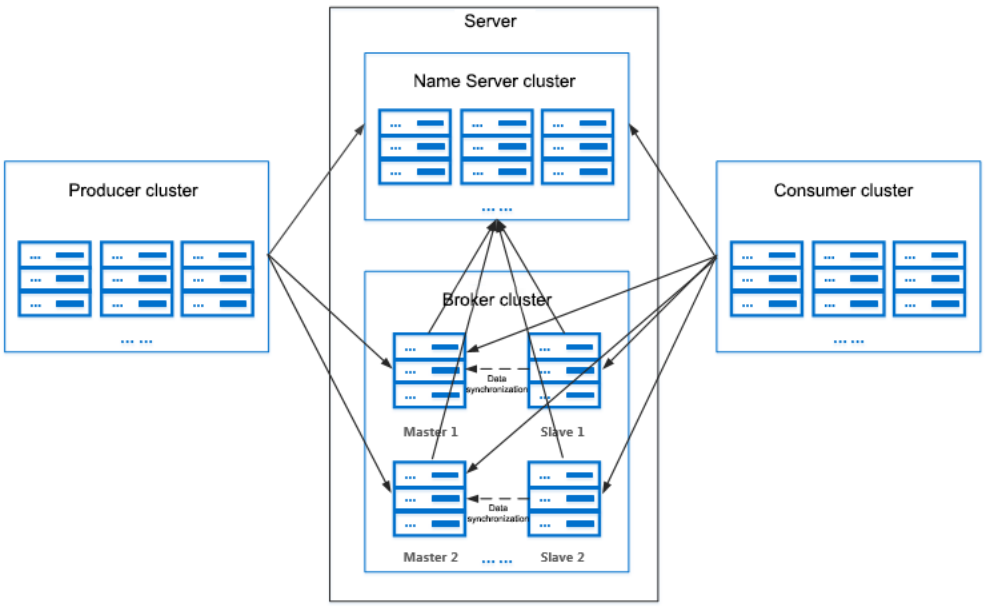
3. Product architecture

SOFAMQ is scalable in any environments. Producers, message servers, and consumers are deployed in clusters respectively. SOFAMQ distinguishes itself from other message servers in its cluster-level high availability. When a message producer sends a message to the message server, the message server randomly selects a consumer. As long as the consumer consumes the message, the message is considered successfully consumed.

***Note**: The SOFAMQ server mentioned here include the name server and the broker. A server is not equivalent to a broker.*

3.1 System deployment architecture

The following figure shows the deployment architecture of SOFAMQ.



Concepts in the figure are described as follows:

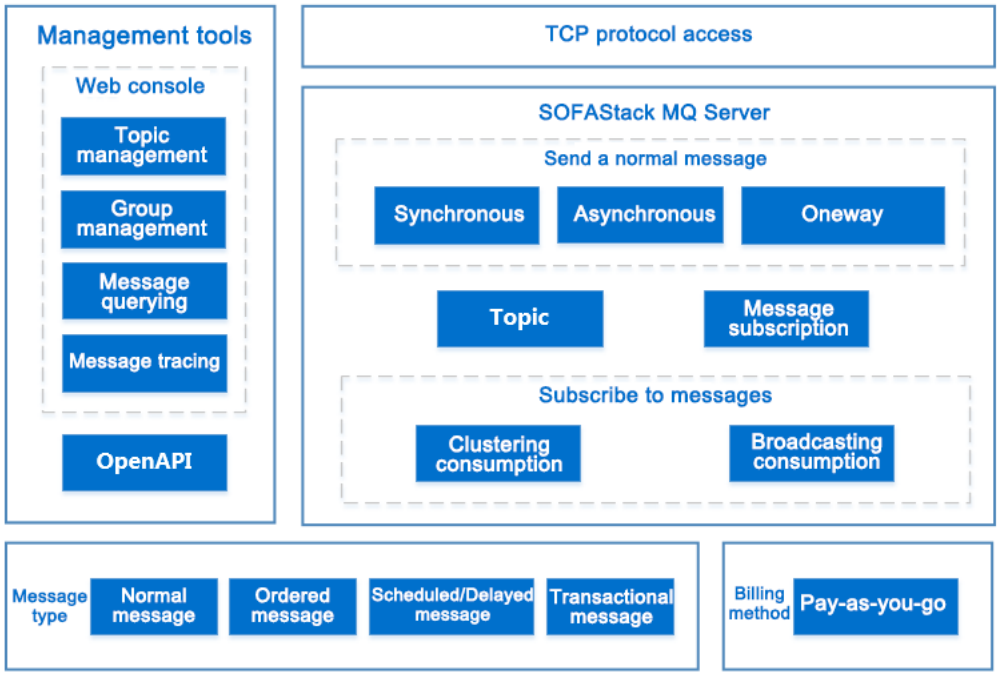
- **Name Server**: a stateless node that can be deployed in clusters, responsible for providing naming service and updating and discovering the broker service.
- **Broker**: the message-transfer role, responsible for storing and forwarding messages. SOFAMQ has master brokers and slave brokers. One master broker can correspond to multiple salve brokers, but one slave broker can only correspond to one master broker. A broker must register itself to the name server after startup. After that, it must report the routing information of topics every 30 seconds to the name server.

- **Producer**: the message producer that establishes a persistent connection (Keep-alive) with a random node in the name server cluster. A producer regularly reads the routing information of topics from the name server, and establishes a persistent connection to the master broker that provides the topic service. The producer also regularly sends heartbeats to the master broker.
- **Consumer**: the message consumer that establishes a persistent connection with a random node in the name server cluster. A consumer regularly pulls the routing information of topics from the name server, and establishes a persistent connection to the master broker and slave broker that provide the topic service. The consumer also regularly sends heartbeats to them. A consumer can subscribe to messages from either the master broker or the slave broker. The subscription rule is determined by the configuration of brokers.

4. Features

SOFAMQ provides highly available messaging services in multiple regions of Alibaba Cloud. Multiple data centers are deployed in a single region, ensuring very high availability. Even if none of the data centers is available, the message publishing service is still available to applications.

SOFAMQ provides access based on the TCP protocol, which ensures quick access of applications to the cloud-based messaging services of SOFAMQ. As long as you are connected to the network, you can deploy your application in an Alibaba Cloud ECS instance or your cloud server. Then you can connect your application to SOFAMQ to send messages from or receive messages to SOFAMQ.



4.1 TCP protocol access

SOFAMQ provides more professional, reliable, and stable SDK access through TCP protocol access. SOFAMQ supports Java.

4.2 Management tools

Web console: The SOFAMQ console supports topic management, group management, message query, and message trace display and query.

OpenAPI: SOFAMQ provides open APIs to help you integrate SOFAMQ management tools into the SOFAMQ console.

4.3 Message type

Normal messages: featureless messages in the message queue, different from scheduled, delayed, ordered, and transactional messages that have specific features.

Transactional messages: messages that implement distributed transaction functions similar to those of X/Open XA, to ensure the final consistency of transactions.

Scheduled and delayed messages: messages that allow message producers to perform scheduled (or delayed) delivery of specified messages. A maximum period of 40 days is supported.

Ordered messages: messages that allow message consumers to consume messages based on the message-sending order.

4.4 Features

Message query: SOFAMQ provides three message query methods: query by message ID, message key, and topic.

Query a message trace: You can use a message trace to track the lifecycle of a message: starting from generation by the producer, to passing through the SOFAMQ server, and to delivery to message consumers. Message trace facilitates the identification of problems and troubleshooting.

Clustering consumption and broadcasting consumption: In clustering consumption, SOFAMQ only needs to ensure that a message is consumed by any consumer in the consumer cluster. In broadcasting consumption, SOFAMQ pushes every message to each of the registered consumers in the consumer cluster to ensure that every message is consumed at least once by every consumer.

Reset consumption offset: You can reset the consumption progress based on the time or consumption offset to re-consume messages or discard accumulated messages.

LDC architecture and message routing: SOFAMQ supports deployment in logical data centers (LDCs) and message routing. This makes logical separation of LDCs and cross-LDC message delivery and consumption much easier.

5. Scenarios

In the scenario of Internet finance, a lot of businesses are involved, such as: payment transfer, billing and interest calculation, business settlement, business marketing, member points, and risk review. Meanwhile, there are many business peaks, such as huge online promotions, flash sales, and anniversary promotions. All these campaigns bring great challenges to the processing performance of microservice applications in distributed systems.

As an important component of a distributed system, SOFAMQ can handle such scenarios well.

Payment transfer is used as an example to show how SOFAMQ implements the following functions:

- Asynchronous decoupling
- Data consistency of distributed transactions
- Load shifting

5.1 Asynchronous decoupling

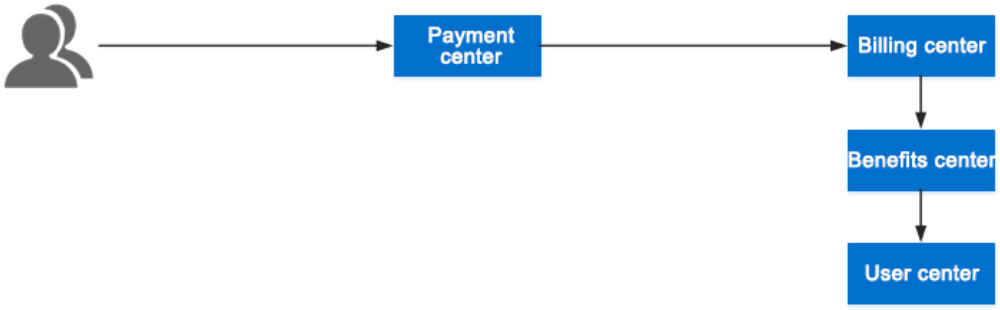
The message publisher and the message consumer can be different applications. One message can be subscribed and consumed by multiple consumers at the same time. SOFAMQ can be used to implement the decoupling of applications, to improve the scalability and availability of their architecture, and improve their user experience. For example, money transfer is a common scenario in finance. Each transfer involves attention from multiple downstream business systems, such as client notification and anti-fraud service event tracking systems. In this scenario, you can create a topic named "transfer", and the downstream services can subscribe to and consume messages from this topic. In this case, the transfer system only needs to focus on its account-related operations, and let SOFAMQ handle the subsequent transfer-related business logic of the downstream services.

5.1.1 Conventional processing

The most common scenario after a transfer succeeds is that the system must generate bills for both parties to the transaction, update their benefits, and send them a notification. There are two common practices as follows:

- **Serial processing**

The flowchart of serial processing is shown in the following figure.

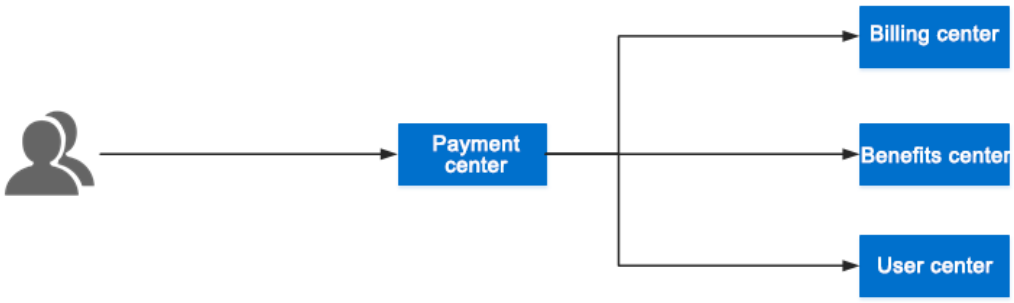


The data flow is described as follows:

1. A user enters the transfer-related information such as the amount to be transferred in the payment center and completes the transfer operation.
2. After the transfer succeeds, a request is sent to the billing center to generate bills for both parties to the transaction.
3. After the successful generation of bills, a request is sent to the benefits center to update their member points
4. After the successful update of the member points, a request is sent to the user center to send a user notification to them.

- **Parallel processing**

The registration process for parallel processing is as shown in the following figure.



The data flow is described as follows:

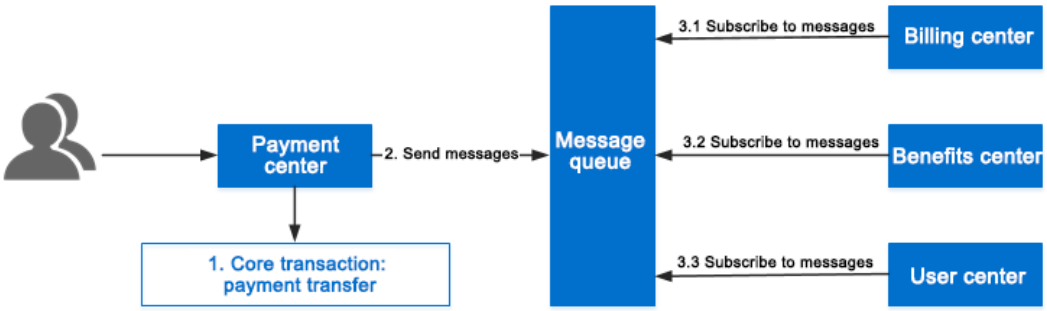
1. A user enters the transfer-related information such as the amount to be transferred in the payment center and completes the transfer operation.
2. After the transfer succeeds, requests are sent to the billing center, user center, and benefits center for corresponding operations at the same time.

The use of SOFAMQ in the payment scenario is described in detail next.

5.1.2 Asynchronous decoupling

Normal message processing

For users, steps such as bill generation, benefits update, and member points update after transfer success do not require instant attention. These operations can be performed by the downstream service systems.



The data flow is described as follows:

1. A user enters the transfer-related information such as the amount to be transferred on the transfer page, and completes the transfer operation.
2. After the transfer succeeds, the transfer system sends a payment message to SOFAMQ. SOFAMQ then immediately returns a response to the payment center, and the transfer is completed.
3. The downstream systems such as the billing center, benefits center, and the user center subscribe to payment messages of SOFAMQ, and complete the subsequent business process.

Asynchronous decoupling is a main feature of SOFAMQ, and its main purposes are to reduce the request-response time and decouple applications. The main scenario is to put time-consuming operations that do not require an instant (synchronous) return of results into SOFAMQ as messages. In addition, SOFAMQ ensures that as long as the message format remains unchanged, the message producer and consumer do not need to contact or be affected by each other. In other words, they are decoupled.

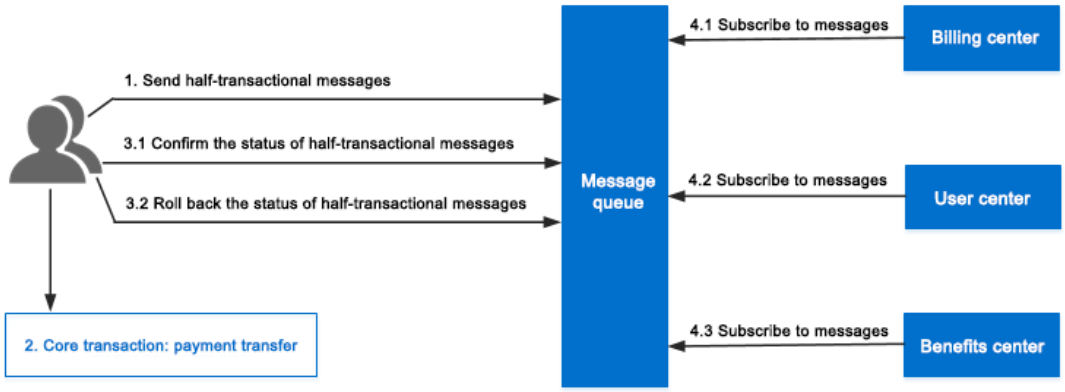
5.2 Data consistency of distributed transactions

In the payment process, the user operation is performed in the payment center, and the billing, benefits update, and notification sending operations are performed in other systems. The data between these systems must be consistent.

In this case, the systems are decoupled and the upstream system does not need to be concerned about the business processing results of downstream systems. However, data consistency, for example, the state consistency between the payment system and its downstream systems is difficult to ensure.

Transactional message processing

To address this issue, SOFAMQ uses transactional messages to ensure the consistency of state data between the systems.



The process is described as follows:

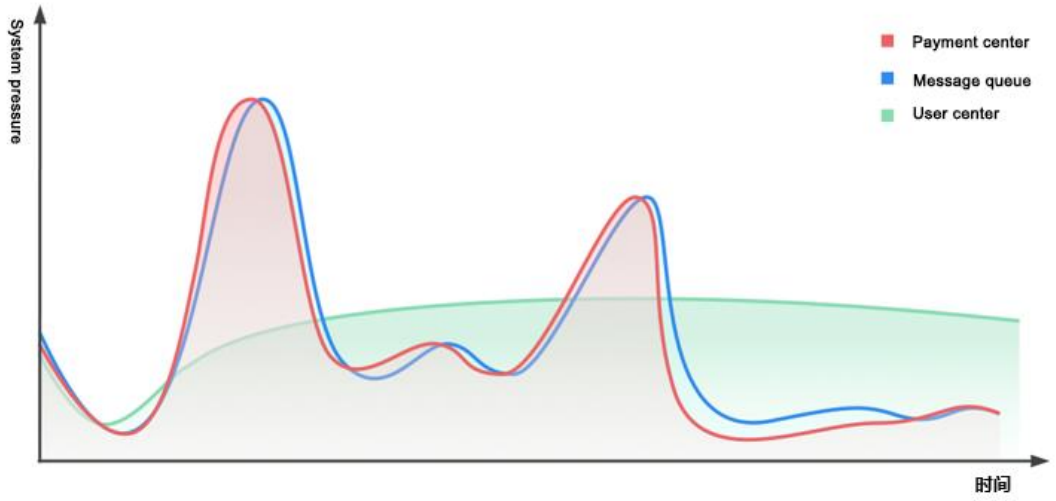
1. The payment center sends a half-transactional message to SOFAMQ.
    1) If the half-transactional message is sent successfully, move to Step 2.
    2) If the half-transactional message failed to be sent, the payment center will not perform the transfer. The process ends. In the end, the data of the payment center and that of the downstream systems are consistent.
2. The payment center starts the transfer process.
    1) If the transfer succeeds, move to Step 3.1.
    2) If the transfer fails, move to Step 3.2.
3. The payment center sends the state of the half-transactional message to SOFAMQ.
    1) If the half-transactional message is committed, generate a payment successful message, and move to Step 4.
    2) If the half-transactional message is rolled back, no payment successful message is generated. The process ends. In the end, the data of the payment center and that of the downstream systems are consistent.
4. The downstream systems receive the payment successful message from SOFAMQ.
5. The downstream systems process the relevant business logic. In the end, the data of the payment center and that of the downstream systems are consistent.

5.3 Load shifting

Load shifting is another commonly used feature of SOFAMQ. This feature is widely used in flash sales or group buying.

Take the payment scenario as an example. During flash sales or group buying, the traffic surges due to the large amounts of user requests. After the payment center processes such large amounts of traffic, the downstream applications' user center may not be able to cope with the load of a massive number of calls. Sometimes, the system may not respond, or may miss some notifications.

When using SOFAMQ, the user center, as a consumer, can consume messages according to the capacity of the application, without being affected by the heavy traffic.



6. Limitations

Item	Range	Description
Length of the topic name	64 characters	The length of topic names cannot exceed this limit, otherwise it will result in a failure to send or subscribe to messages of this topic.
Message size	Normal and ordered messages: 4 MB Transactional and scheduled/delayed messages: 64 KB	The size of a message cannot exceed the limit for its type, otherwise the message will be discarded.
Message retention time	Three days	Messages are retained for a maximum of three days. After three days, they will be automatically deleted.
Reset consumption offset	Three days	SOFAMQ supports resetting messages published at any time point in the last three days.
Single-instance message receiving and sending in TPS	The standard version: 5,000 messages/second	
The delay time of scheduled/delayed messages	40 days	You can set the msg.setStartDeliverTime parameter to any time (unit: millisecond) within 40 days. If the time is set to a value longer than 40 days, the message sending will fail.

7. Appendix: Basic terminologies

Topic

A topic is the primary message type. SOFAMQ classifies messages by topics.

Message

The carrier of information in the message queue.

Message ID

The global and unique ID that is automatically generated by the SOFAMQ system to uniquely identify a message.

Message key

The business identifier that is set by the producer to uniquely identify a business logic of messages.

Tag

A tag, the secondary message type, is used to further classify messages of a topic.

Producer

A producer, also known as the message publisher, produces and sends messages.

Producer instance

An object instance of the producer. Different producer instances can run in different processes or on different servers. A producer instance is thread safe and can be shared by multiple threads in the same process.

Consumer

A consumer, also known as the message subscriber, receives, and consumes messages.

Consumer instance

An object instance of the consumer. Different consumer instances can run in different processes or on different servers. A consumer instance contains a configured thread pool to consume messages.

Group

A type of producers or consumers that usually produce or consume the same type of messages, and use the same logic to publish or subscribe to messages.

Group ID

The identifier of a group.

Queue

One or more queues are used under each topic to store messages. The number of queues corresponding to each topic is related to the message type and the region of instances.

Clustering consumption

All consumers with the same group ID consume an equal number of messages. Assume that a topic has 9 messages and there are three consumer instances with the same group ID. In the clustering consumption mode, these consumers equally share the messages, and each consumes three of them.

Broadcasting consumption

In this mode, every consumer instance with the same group ID consumes each of the messages once. Assume that, a topic has 9 messages and there are three consumer instances with the same group ID. In broadcasting consumption mode, each instance consumes all 9 messages.

Scheduled message

If a producer sends a message to the SOFAMQ server and wants to deliver the message later at a specified time point, we call this message a scheduled message.

Delayed message

If a producer sends a message to the SOFAMQ server and wants to deliver the message after a specified period, we call this message a delayed message.

Transactional message

SOFAMQ provides distributed transaction functions that are similar to those of X/Open XA. Transactional messages of SOFAMQ help ensure the final consistency of distributed transactions.

Ordered message

A message type provided by SOFAMQ. These messages are published and consumed in order, and are divided into globally ordered messages and partitionally ordered messages. Currently, SOFAMQ only supports partitionally ordered messages.

Partitionally ordered message

All messages of a specified topic are partitioned according to the sharding key. Messages in the same partition are published and consumed in a strict first-in-first-out (FIFO) order. The sharding key is a key field used to distinguish different partitions of ordered messages. The sharding key is different from the message key of normal messages.

Message accumulation

Sometimes, after a producer sends messages to the SOFAMQ server, the consumers may not be able to correctly consume all messages in a short time due to limited consumption

capacity. In this case, the unconsumed messages are stored on the SOFAMQ server. We call this state message accumulation.

Message filtering

Consumers can filter messages by tags, to ensure that they only receive the filtered types of messages. Consumers complete message filtering on the SOFAMQ server.

Message trace

It consists of data such as the time and location of relevant nodes in the lifecycle of a message. Its lifecycle starts from the time and location that the producer publishes it, and ends at the time and location that a consumer consumes it. You can use a message trace to track the lifecycle of a message: starting from generation by the producer, to passing through the SOFAMQ server, and to delivery to message consumers. Message trace facilitates the identification of problems and troubleshooting.

Reset consumption offset

You can reset the consumption progress of a topic that a consumer has subscribed to within the time range of persisted storage (three days by default). Then the consumer receives messages that the producer publishes on the SOFAMQ server after the set time point.


蚂蚁集团
ANT GROUP