ALIBABA CLOUD

Alibaba Cloud

Alibaba Cloud Message Queue for MQTT

Function Overview

Document Version: 20211217

Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	**Danger:** Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	**Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	**Notice:** If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	**Note:** You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.Send and receive messages

Message Queue for MQTT provides a fundamental feature that allows you to send and receive messages. Clients and backend service applications can be connected to the Message Queue for MQTT broker. This allows you to implement messaging between clients or between clients and backend service applications.Message Queue for MQTT This topic describes the basic scenarios of this feature and provides demos that you can use to send and receive messages.

Background information

In a message interaction scenario in Message Queue for MQTT, both clients and backend service applications can serve as message producers or consumers. Clients must use the client SDK and backend service applications must use the cloud SDK of Message Queue for MQTT to be connected to Message Queue for MQTT to send and receive messages.

Message Queue for MQTT provides the following SDKs:

- Client SDK: an open source SDK of Message Queue Telemetry Transport (MQTT), which supports the MQTT 3.1.1 protocol and is applicable to client-related development.
- Cloud SDK: an SDK developed by Alibaba Cloud, which is applicable to the development of backend service applications. The cloud SDK can be used to send and receive messages, and receive status notifications from the client SDK.

Message interaction between clients

In this scenario, messages are sent and received in the mobile environment. The clients are connected to Message Queue for MQTT based on the MQTT protocol. They can use the open source client SDK to be connected to Message Queue for MQTT to send and receive messages.

- **Instant messaging (IM)**: For example, two mobile phones on which a chat app is installed are connected to the Message Queue for MQTT broker to send and receive messages.
- **Smart device management**: For example, an app on a smart phone instructs a power bank that is connected to the Message Queue for MQTT broker to be ejected. The power bank is automatically ejected after it receives the instruction.

Examples:

Message interaction between clients and backend service applications

In this scenario, the message producer and consumer are distributed in the mobile environment and the cloud. Clients and backend service applications communicate with each other by using Message Queue for MQTT. Clients use the client SDK to be connected to the Message Queue for MQTT broker, whereas backend service applications use the cloud SDK to be connected to the Message Queue for MQTT broker.

- **Device status report**: In this case, clients send messages, whereas backend service applications receive messages. For example, a large number of digital price tags in the mobile environment regularly report their display statuses and power consumption. The backend service applications in the cloud analyze the statuses of the digital price tags based on the reported data and make adjustment based on business requirements.
- **System message push**: In this case, backend service applications send messages, whereas clients receive messages. For example, a game application in the cloud sends an announcement indicating that the game service will be suspended for updates. The Message Queue for MQTT broker pushes the announcement to all the installed mobile clients of the game application to notify game users of the announcement in the form of notifications.
- **Message receiving**: The cloud SDK supports the clustering consumption mode. In other words, consumers in the same group can subscribe to and consume different messages.

Examples:

Demos for you to use the client SDK

> **Note** The following demos provide examples on how to use SDKs to send and receive messages. The programming language that is used in a demo provided for a scenario is not the only supported programming language in the scenario. You can modify parameter settings for different programming languages based on the demo in Java.

- **Use the client SDK to send and receive messages ❯**
- **Use WebSocket to send and receive web application messages ❯**

When a client is connected to the Message Queue for MQTT broker to send and receive messages, you can use one of the following methods to authenticate the client. For more information, see 鉴权概述.

> **Note** If you use the signature authentication mode to send and receive messages, you need to set the Username and Password parameters. For more information about how to obtain the values of the parameters to verify signatures, see Signature authentication.

- **Use the signature authentication mode of Message Queue for MQTT ❯**
- **Use the token-based authentication mode of Message Queue for MQTT ❯**
- **Use SSL encryption of Message Queue for MQTT ❯**

Send and receive messages

Client authentication

Demos for you to use the cloud SDK

> **Note** Only Java is supported for the cloud SDK.

- **Use the client SDK to send and receive messages ❯**

2.Receive offline messages

To simplify the mechanism used to receive offline messages, Message Queue for MQTT automatically loads offline messages and delivers them to a Message Queue for MQTT client after the client establishes a connection to the Message Queue for MQTT broker and passes permission verification.

Usage notes

- After a Message Queue for MQTT client establishes a connection to the broker, the client must pass the permission verification to enable the automatic loading of offline messages. For example, if a Message Queue for MQTT client adopts the token-based verification mode, you must upload a token and the client must pass the verification before it can receive offline messages.
- An offline message takes a specific amount of time to generate. This is because the pushed message can be determined as an offline message only after the acknowledgment of the Message Queue for MQTT client times out. Therefore, if the Message Queue for MQTT client experiences transient disconnection and reconnection, the client may not be able to immediately receive the latest offline message. The latency is generally 5 to 10 seconds.
- If the number of offline messages exceeds 30, Message Queue for MQTT sends offline messages in batches. Each batch contains 30 messages. Batches are sent at intervals of 5 seconds.

> **Note** The automatic loading mechanism allows you to receive offline messages without the need to pull the messages. The mechanism takes effect even if you still use the message pulling method to receive offline messages.

Configuration

You can set the QoS and cleanSession parameters in a consumer to determine whether the consumer can receive offline messages.

Combinations of QoS levels and the cleanSession parameter

QoS level	cleanSession=true	cleanSession=false
QoS0	Offline messages are not delivered. Only one delivery attempt is made for online messages.	Offline messages are not delivered. Only one delivery attempt is made for online messages.
QoS1	Offline messages are not delivered. Online messages are guaranteed to reach the intended Message Queue for MQTT clients.	Offline messages are delivered. Both offline and online messages are guaranteed to reach the intended Message Queue for MQTT clients.
QoS2	Offline messages are not delivered. Online messages are delivered once.	Not supported.

For more information about the QoS and cleanSession parameters, see Terms.

3.Obtain the status of a Message Queue for MQTT client

Message Queue for MQTT allows you to perform synchronous queries on or subscribe to asynchronous status notifications of a Message Queue for MQTT client to obtain the status of the client. This topic describes the principles, scenarios, and implementation of the two methods. This topic also describes the differences between the two methods.

Principles

The Message Queue for MQTT broker allows you to use the following methods to obtain the status of a Message Queue for MQTT client:

- **Synchronous query**
  This method is relatively simple. You can make an HTTP or HTTPS request by using a public endpoint to call an API operation to query the status of a specific client. You can also use this method to query the statuses of multiple clients at a time.
- **Asynchronous status notification**
  This is a notification-based method. If a Message Queue for MQTT client goes online or offline, the Message Queue for MQTT broker sends a notification about the status change event to a backend service application that is deployed on an Alibaba Cloud server.
  In this case, the backend service application is asynchronously notified of the status change event instead of the real-time status of the client. The backend service application must determine the client status based on the timeline of a series of events.

Scenarios

The preceding two methods apply to the following scenarios:

- **Synchronous query**
  - The subsequent operation logic depends on whether the specified Message Queue for MQTT client is online during the main service process.
  - O&M engineers need to determine whether the specified Message Queue for MQTT client is online.
- **Asynchronous status notification**
  - The Message Queue for MQTT broker needs to trigger specific predefined actions when the specified Message Queue for MQTT client goes online or offline.
  - The Message Queue for MQTT broker needs to collect and analyze the data recorded in the status change events of the specified Message Queue for MQTT client, and sends notifications based on the events.

Differences

The two methods have the following differences:

- Synchronous queries can be performed to query the real-time status of a specific Message Queue for MQTT client, which is theoretically more accurate than the other method.
- Asynchronous status notifications are based on message decoupling. This method is more complex and prone to misjudgment when a client status is being determined. However, this method can be used to analyze the status traces of multiple Message Queue for MQTT clients based on events. You can use this method to collect statistics on the statuses of a large number of Message Queue for MQTT clients.

Implementation

- **Synchronous query**

You can call the following operations to query the status of one or more clients:

- QuerySessionByClientId
- BatchQuerySessionByClientIds

- **Asynchronous status notification**
  You can call the cloud SDK that is provided by Message Queue for MQTT to obtain the status notifications of a Message Queue for MQTT client. For more information about how to download the cloud SDK, see Release notes.
  The following sample code shows how to receive the status notifications of a Message Queue for MQTT client:

```
package com.aliyun.openservices.lmq.example;
import com.alibaba.fastjson.JSONObject;
import com.alibaba.mqtt.server.ServerConsumer;
import com.alibaba.mqtt.server.callback.StatusListener;
import com.alibaba.mqtt.server.config.ChannelConfig;
import com.alibaba.mqtt.server.config.ConsumerConfig;
import com.alibaba.mqtt.server.model.StatusNotice;
public class MQTTClientStatusNoticeProcessDemo {
    public static void main(String[] args) throws Exception {
         /**
          * The endpoint of the Message Queue for MQTT instance that you created.
          * To obtain the endpoint for accessing the instance by using the cloud SDK, .
          * Use a domain name instead of an IP address in the endpoint. Otherwise, a broker exception may occur.
          */
         String domain = "domain";
          /**
          * The port that is used by the cloud SDK. The protocol and port that are used by the cloud SDK must match. Set the port number to 5672.
          */
         int port = 5672;
          /**
          * The ID of the Message Queue for MQTT instance that you created.
          */
         String instanceId = "instanceId";
          /**
          * The AccessKey ID that you created in the Alibaba Cloud RAM console for identity authentication. For information about how to obtain your AccessKey ID, see Obtain an AccessKey pair.
          */
         String accessKey = "accessKey";
          /**
          * The AccessKey secret that you created in the Alibaba Cloud RAM console for identity authentication. The AccessKey secret is required only for signature verification. For information about how to obtain your AccessKey secret, see Obtain an AccessKey pair.
          */
         String secretKey = "secretKey";
          /**
          # The group ID that you created in the Message Queue for MQTT console.
          * */
         String mqttGroupId = "mqttGroupId";
         ChannelConfig channelConfig = new ChannelConfig();
         channelConfig.setDomain(domain);
         channelConfig.setPort(port);
```

```
        channelConfig.setInstanceId(instanceId);
        channelConfig.setAccessKey(accessKey);
        channelConfig.setSecretKey(secretKey);
        ServerConsumer serverConsumer = new ServerConsumer(channelConfig, new ConsumerConfi
g());
        serverConsumer.start();
        serverConsumer.subscribeStatus(mqttGroupId, new StatusListener() {
            @Override
            public void process(StatusNotice statusNotice) {
                System.out.println(JSONObject.toJSONString(statusNotice));
            }
        });
    }
}
```

References

Export online and offline events of Message Queue for MQTT clients

4.Data interoperability across cloud products

4.1. Message structure mappings between Message Queue for MQTT and Message Queue for Apache RocketMQ

This topic describes mappings between the message structures and properties involved in interaction with Message Queue for MQTT by using the
Message Queue for Apache RocketMQ
SDK, helping you better understand and use Message Queue for MQTT and Message Queue for Apache RocketMQ.
Message Queue for MQTT is a gateway service that is intended for mobile devices. Message Queue for MQTT can interact with other Alibaba Cloud services such as

Message Queue for Apache RocketMQ

based on data interaction rules. For more information, see Manage rules.

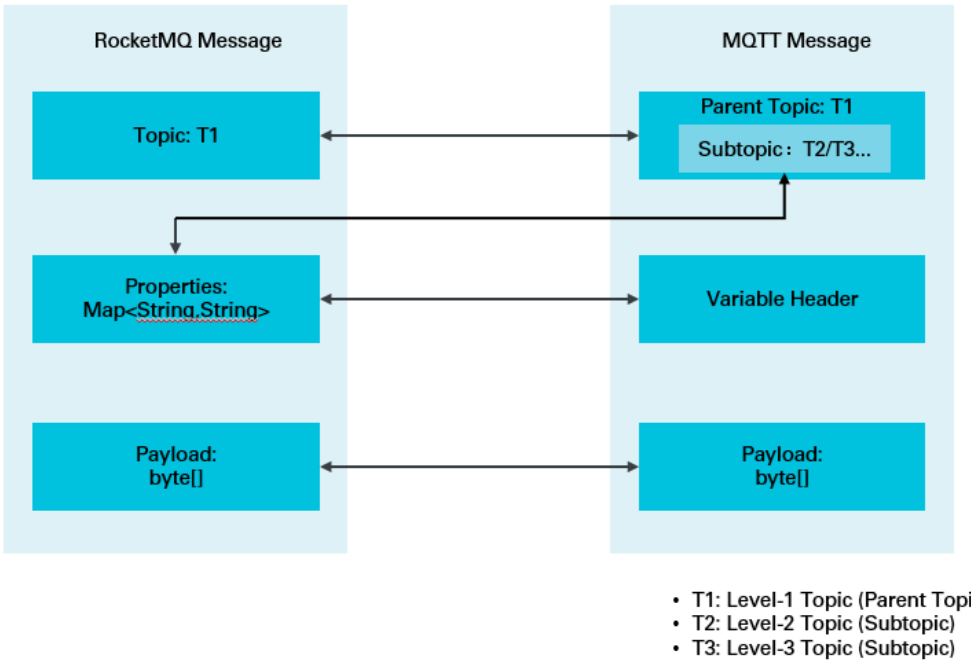
If you use Message Queue for MQTT independently, ignore the mappings described in this topic and observe the Message Queuing Telemetry Transport (MQTT) protocol.

For more information about Message Queue for MQTT, see What is Message Queue for MQTT? and Terms.

Message structure mappings

Message Queue for MQTT and

Message Queue for Apache RocketMQ
are messaging systems that are based on the publish-subscribe model and have similar concepts. The following figure shows the differences in the major concepts and mappings between them.



Message Queue for MQTT supports multi-level topics, whereas

Message Queue for Apache RocketMQ
supports one-level topics, as shown in the preceding figure. Therefore, a level-1 topic in Message Queue for MQTT is mapped to a topic in
Message Queue for Apache RocketMQ
, and level-2 and level-3 topics in Message Queue for MQTT are mapped to the message properties in
Message Queue for Apache RocketMQ
.
The

Message Queue for Apache RocketMQ
protocol supports messages with custom properties, whereas the MQTT protocol does not support properties. However, part of the information in Message Queue for MQTT is mapped to message properties in
Message Queue for Apache RocketMQ
. This facilitates tracing of the headers and device information in the MQTT protocol and allows users of the
Message Queue for Apache RocketMQ
SDK to retrieve such information.

> **Note** For information about how to configure mappings from properties in Message Queue for Apache RocketMQ to parameters in Message Queue for MQTT, see the table in Property mappings.

Message Queue for Apache RocketMQ
and Message Queue for MQTT use the data serialization results of your service messages as the payload.
Message Queue for Apache RocketMQ
and Message Queue for MQTT do not further encode and decode the service messages.

Property mappings

The following table lists the property mappings supported between Message Queue for MQTT and Message Queue for Apache RocketMQ. You can set or retrieve information by reading and writing these properties during interaction between applications that use the Message Queue for Apache RocketMQ and Message Queue for MQTT SDKs.
For more information about QoS, cleanSession, topics, and client IDs, see Terms.

Message Queue for MQTT parameter	Message Queue for Apache RocketMQ property key	Valid property value	Description
QoS	qoslevel	0, 1, and 2	This property can be set when Message Queue for Apache RocketMQ sends messages to Message Queue for MQTT. If it is not set, the default value 1 is used. Message Queue for Apache RocketMQ can directly read the QoS parameter from the messages that are sent from Message Queue for MQTT.
cleanSession	cleansessionflag	true and false	This property can be set when Message Queue for Apache RocketMQ sends P2P messages to Message Queue for MQTT clients. If it is not set, the default value true is used. This property cannot be set for other message types. Message Queue for Apache RocketMQ can directly read the cleanSession parameter from the messages that are sent from Message Queue for MQTT.

Message Queue for MQTT parameter	Message Queue for Apache RocketMQ property key	Valid property value	Description
Subtopic	mqttSecondTopic	A string that indicates a specific subtopic	This property can be set when a subtopic is required to filter the messages that Message Queue for Apache RocketMQ sends to Message Queue for MQTT clients. If it is not set, the default value null is used. Message Queue for Apache RocketMQ can directly read the subtopic from the messages that are sent from Message Queue for MQTT.
Topic in messages received on a client	mqttRealTopic	The sub-level string that services expect message-receiving clients to display	This property can be set when a Message Queue for MQTT client is expected to display the specified subtopic name after it receive messages from Message Queue for Apache RocketMQ . This property is typically applied to P2P messages. If it is not set, P2P messages use a fixed topic name by default. The messages that Message Queue for MQTT sends to Message Queue for Apache RocketMQ do not contain the corresponding parameter.
clientId	clientId	A string that indicates a specific client ID	This property cannot be set. When a Message Queue for MQTT client sends a message to Message Queue for Apache RocketMQ , the clientId parameter is used to trace the ID of the client.

4.2. Data inflow across cloud products

You can configure the Message Queue for MQTT to customize data flow from other Alibaba Cloud products to Message Queue for MQTT. This topic describes the principles and application scenarios of cross-product data inflows, and Message Queue for MQTT mapping to resources of other Alibaba cloud products.

How it works

To import data from other Alibaba cloud products, Message Queue for MQTT, you need to create a data inflow rule. This rule reads data from the Alibaba Cloud product that you have configured and pushes the data to the MQTT client by using the MQTT Protocol. In this way, the Alibaba Cloud product API is called directly to send data to the MQTT client.

> **Notice** Currently, you can only set Message Queue for Apache RocketMQ data import Message Queue for MQTT.

Scenario

Instruction delivery scenarios

The background control service deployed on the cloud sends commands to

Message Queue for Apache RocketMQ
, Message Queue for MQTT based on the configured rules,
Message Queue for Apache RocketMQ
To MQTT topics and then push the corresponding data to the target MQTT device.

Resource Mapping methods

Data flow rules are mapped at the granularity of Message Queue for MQTT the parent Topic of maxcompute, that is, the data of a resource of other Alibaba Cloud products can be imported to Message Queue for MQTT in the parent Topic of log service. You can use other extension properties to configure subtopics that are pushed by the MQTT protocol.

Mapping

MQTT resources	Other Alibaba Cloud products	Other Alibaba cloud resources	Packet definition
MQTT Topic	Message Queue for Apache RocketMQ	Message Queue for Apache RocketMQ The Topic	Message structure mappings between Message Queue for MQTT and Message Queue for Apache RocketMQ

References

For more information about console operations, see Manage data inbound rules.

4.3. Export data from Message Queue for MQTT to other Alibaba Cloud services

You can configure data outbound rules of Message Queue for MQTT to export data from Message Queue for MQTT to other Alibaba Cloud services. This topic describes the principles, scenarios, and limits of exporting data from Message Queue for MQTT to other Alibaba Cloud services. This topic also describes the resource mappings between Message Queue for MQTT and other Alibaba Cloud services.

Principles

You must create a data outbound rule before you can export data from Message Queue for MQTT to other Alibaba Cloud services. This rule is used to export messages from a Message Queue for MQTT client to another Alibaba Cloud service. This way, you can directly call the API of the Alibaba Cloud service to read messages sent from the Message Queue for MQTT client.

Scenarios

Device data reporting

Massive devices report their status data to Message Queue for MQTT by using the MQTT protocol. After you configure a data outbound rule to map the topics of Message Queue for MQTT to the topics of

Message Queue for Apache RocketMQ
, you can start the consumer data processing service of
Message Queue for Apache RocketMQ
in the backend to process the reported data.

Limits

- You can export data only from Message Queue for MQTT to
  Message Queue for Apache RocketMQ
  .

Rule-based mapping

Data outbound rules support mapping resources to a specific parent topic of Message Queue for MQTT. You can export all messages under a parent topic of Message Queue for MQTT to a resource in other Alibaba Cloud services. You can configure MQTT-based subtopics by using other extension properties.

Mappings

Message Queue for MQTT resource	Alibaba Cloud service	Alibaba Cloud service resource	Packet definition
Topic of Message Queue for MQTT	Message Queue for Apache RocketMQ	Topic of Message Queue for Apache RocketMQ	Message structure mappings between Message Queue for MQTT and Message Queue for Apache RocketMQ

References

For more information about console operations, see Manage data outbound rules.

4.4. Export online and offline events of Message Queue for MQTT clients

By running rules for client status notification of Message Queue for MQTT, you can export the obtained online and offline events of Message Queue for MQTT clients to other Alibaba Cloud services. This is an asynchronous method for status notification. This topic describes the principles and application scenarios of client status notifications. It also describes the resource mappings between Message Queue for MQTT and other Alibaba Cloud services.

How it works

When Message Queue for MQTT clients go online or offline, the Message Queue for MQTT broker pushes a status notification message to other backend cloud services based on the configured rules for client status notification. Service applications are generally deployed on Elastic Compute Service (ECS) instances. Service applications can subscribe to the status notification message from backend cloud services to obtain online and offline events of all related Message Queue for MQTT clients.

This method asynchronously perceives the client status and detects online and offline events instead of the status of clients. Therefore, cloud applications must deduce the client status based on the timeline of a series of events.

Asynchronous status notifications are based on message decoupling. Therefore, status determination based on such notifications is more complex and prone to misjudgment. However, this method can be used to analyze the running status traces of multiple Message Queue for MQTT clients based on events.

> **Notice** The only supported backend cloud service is Message Queue for Apache RocketMQ .

Scenarios

Client status notifications are used in the scenarios where service applications need to trigger some predefined actions when Message Queue for MQTT clients go online or offline.

For example, in the scenario of client status aggregation, a Message Queue for MQTT client makes status changes such as going online and going offline. Then, the Message Queue for MQTT instance encapsulates the status changes and forwards them to a

Message Queue for Apache RocketMQ
instance based on the configured rules for client status notification to aggregate and collect client status data.

> **Note** In other scenarios, we recommend that you obtain the client status by calling synchronous query operations. For more information, see Obtain the status of a Message Queue for MQTT client.

Resource mapping methods

Status change notifications of all clients under the same Message Queue for MQTT group ID are forwarded to the resources of the same Alibaba Cloud service that you configure.

Mappings

Message Queue for MQTT resource	Another Alibaba Cloud service	Resource of another Alibaba Cloud service	Packet definition

Message Queue for MQTT resource	Another Alibaba Cloud service	Resource of another Alibaba Cloud service	Packet definition
MQTT Group ID	Message Queue for Apache RocketMQ	Message Queue for Apache RocketMQ topics	Message structure mappings between Message Queue for MQTT and Message Queue for Apache RocketMQ

Procedure

As previously described, if asynchronous status notifications are used, you must create rules for client status notification, and export the status notification messages to backend cloud services. The

Message Queue for Apache RocketMQ
backend cloud service is used in the following example.

1. Create a rule for client status notification.
   When you create a rule in the Message Queue for MQTT console, select the group ID of the group where the Message Queue for MQTT clients to which you want to pay attention are located. For more information about how to create a rule for client status notification, see Create a rule for client status notification.
2. Service applications subscribe to this type of notifications.
   After the service applications adopt the rule created in Step 1, they can receive online or offline events of related Message Queue for MQTT clients. For more information about how to receive notifications in
   Message Queue for Apache RocketMQ
   , see Subscribe to messages. For more information about the sample code, see MQTTClientStatusNoticeProcessDemo.java.
   Event types are specified in the tag of
   Message Queue for Apache RocketMQ
   and represent online or offline events. Data format:
   ```
   MQ Tag: connect/disconnect/tcpclean
   ```
   where:
   - A connect event indicates that the Message Queue for MQTT client goes online.
   - A disconnect event indicates that the Message Queue for MQTT client actively disconnects from the Message Queue for MQTT broker. Based on the Message Queuing Telemetry Transport (MQTT) protocol, the Message Queue for MQTT client sends a disconnect message before it actively closes the TCP connection. The Message Queue for MQTT broker triggers a disconnect message after it receives the disconnect message from the Message Queue for MQTT client. If the SDK for a Message Queue for MQTT client does not send a disconnect message based on the MQTT protocol, the Message Queue for MQTT broker cannot receive the disconnect message.
   - A tcpclean event indicates that the TCP connection is closed. If the current TCP connection is closed, a tcpclean event is triggered regardless of whether the Message Queue for MQTT client has explicitly sent a disconnect message.

> **Note**
> A tcpclean message indicates that the TCP connection of the Message Queue for MQTT client is closed. A disconnect message only indicates that the Message Queue for MQTT client actively sends a disconnect message. Some Message Queue for MQTT clients may fail to send a disconnect message due to unexpected exits. This is subject to the implementation on the Message Queue for MQTT clients. Therefore, determine whether a Message Queue for MQTT client is offline based on the tcpclean event.

Data content is in JSON format. The related keys have the following meanings:

- clientId indicates a specific Message Queue for MQTT client.
- time indicates the occurrence time of the event.
- eventType indicates the event type, which is used by the Message Queue for MQTT client to differentiate events.
- channelId uniquely identifies a TCP connection.
- clientIp indicates the public egress IP address used by the Message Queue for MQTT client.

Examples:

```
clientId: GID_XXX@@@YYYYY
time:1212121212
eventType:connect/disconnect/tcpclean
channelId:2b9b1281046046faafe5e0b458e4XXXX
clientIp: 192.168.X.X:133XX
```

To determine whether a Message Queue for MQTT client is online, check the last received message and status notification messages. You cannot depend only on the last received message to determine the client status.
Determine the client status based on the following rules:

- The sequence of online and offline events generated by Message Queue for MQTT clients that have the same clientId is determined by time. This means that a more recent event has a greater timestamp.
- Message Queue for MQTT clients that have the same clientId may be transiently disconnected multiple times. Therefore, when an offline notification message is received, you must check whether the message is related to the current TCP connection based on the channelId field. In short, an offline notification message can overwrite only another offline notification message that has the same channelId. If channelId values of two offline notification messages are different, the offline notification message that has a more recent occurrence time cannot overwrite the other message. A channelId represents the existence of a TCP connection. Only one connect event and one close event exist for a single TCP connection.

Sample Java code for asynchronous status notifications ▼

References

For more information about console operations, see Manage rules for client status notification.

5.P2P Messaging model (MQTT)

In addition to the publish-subscribe messaging modes supported by the standard MQTT protocol, Message Queue for MQTT it also supports the Point-to-Point (Point to Point) mode. This topic describes the concepts, principles, and usage of the P2P model. Message Queue for MQTT Point-to-point message sending and receiving.

What is the P2P model?

P2P, as its name implies, is a one-to-one messaging model where only one message sender and one message receiver are involved. The publish-subscribe model is usually used in one-to-many or many-to-many message transmission scenarios where one or more message senders and many message receivers are involved.

In the P2P model, the sender knows the information about the expected receiver when sending a message, and expects that the message will be consumed only by a specific client. The sender directly specifies the receiver in a topic when sending a message. The receiver can obtain the message without subscribing to the topic in advance.

The P2P model not only saves the subscription registration cost for the receiver but, due to the independently optimized messaging link, also reduces the push latency.

Differences between the P2P model and the publish-subscribe model

In Message Queue for MQTT the differences between the P2P model and the publish-subscribe model for sending and receiving messages are as follows:

- In the publish-subscribe model, the message sender needs to send the messages of the topic to which the receiver has subscribed. In the P2P model, the receiver does not need to subscribe to the topic, and the sender can send messages directly to the target client according to the standards.
- In the publish-subscribe model, the receiver needs to subscribe to the topic in advance to receive messages from the sender. In the P2P model, the receiver does not need to subscribe to the topic in advance. Therefore, the program logic at the receiver is simplified and the subscription cost is saved.

Send P2P messages

When using the MQTT SDK to send P2P messages, you must set the level-2 topic to "p2p" and the level-3 topic to the client ID of the target receiver.

Java example

```
String p2pTopic =topic+"/p2p/GID_xxxx@@@DEVICEID_001";
sampleClient.publish(p2pTopic,message);
```

Use

Message Queue for Apache RocketMQ
when using the [******] SDK to send P2P messages, you only need to set the subtopic attribute to the preceding subtopic string because the parent Topic and subtopic are set separately.
Java example

```
String subTopic="/p2p/GID_xxxx@@@DEVICEID_001";
msg.putUserProperties(PropertyKeyConst.MqttSecondTopic, subTopic);
```

The links to multi-language sample codes for sending P2P messages are shown in the following table.

Language	Sample code
.NET	. NET sample code
C	C sample code
Java	Java sample code
JavaScript	JavaScript sample code
Python	Python sample code
PHP	PHP Sample code

Note The Go SDK cannot be used to send and receive P2P messages.

Receive P2P messages

The client does not need to subscribe to P2P messages. Instead, it can receive P2P messages after initialization.