消息队列 MQ

微消息队列 MQ for IoT

微消息队列 MQ for IoT

微消息队列概述

本文主要介绍 MQ 微消息队列（Light Message Queue，又称 MQ for IoT），如果说传统的消息队列大多应用于微服务之间，那么物联网微消息队列则实现了端与云之间的消息传递，真正实现万物互联。

如何使用微消息队列请参考 接入准备。

简介

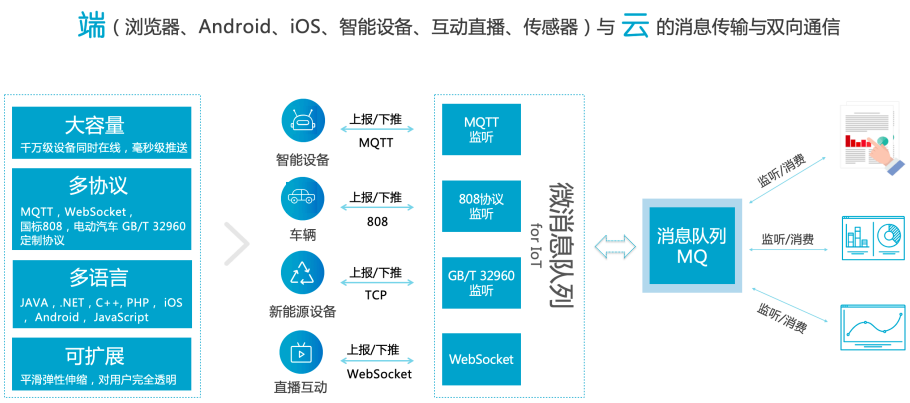
针对用户在移动互联网以及物联网领域的存在的特殊消息传输需求，MQ 开放了 MQTT 协议的完整支持。

消息队列遥测传输（Message Queuing Telemetry Transport，简称 MQTT）是一种轻量的，基于发布订阅模型的即时通讯协议。该协议设计开放，协议简单，平台支持丰富，几乎可以把所有联网物品和外部连接起来，因此在移动互联网和物联网领域拥有众多优势。

MQTT 协议的特点包括：

- 使用发布/订阅消息模式，提供一对多的消息分发，解除了应用程序之间的耦合；
- 对负载内容屏蔽的消息传输；
- 使用 TCP/IP 提供基础的网络连接；
- 有三种级别的消息传递服务；
- 小型传输，开销很小（头部长度固定为 2 字节），协议交换最小化，以降低网络流量。

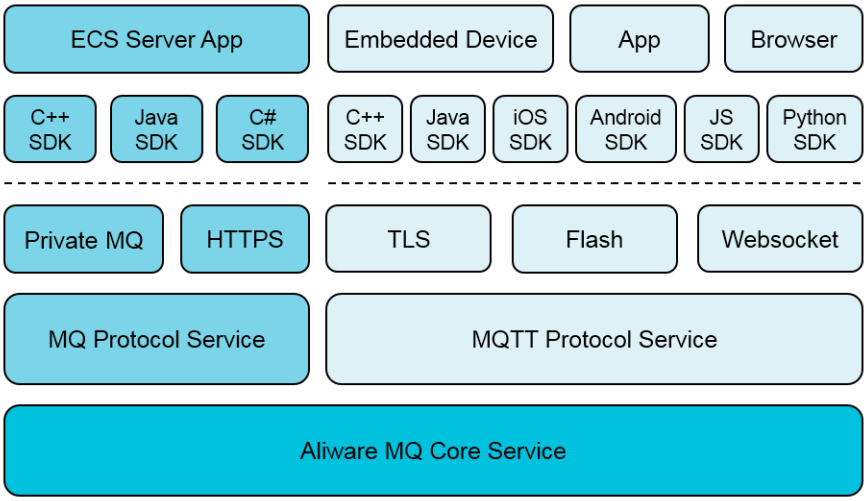
产品优势



微消息队列 MQ for IoT 广泛应用移动互联网以及物联网领域，覆盖覆盖动直播、车联网、金融支付、智能餐饮、即时聊天等多种应用场景，采用分布式理念进行设计，系统无单点瓶颈，各组件之间均可以无限水平扩展，保证容量可以随着您的在线使用量进行调整，并且对用户完全透明；

系统结构

MQ 完整支持 MQTT 3.1.1 协议，通过在 MQ 核心的基础上增加 MQTT 协议网关的方式对互联网上的客户端提供服务。整个系统架构如下图所示：



其中，Aliware MQ Core Service 负责底层的消息存储和分发，上层支持 **MQ 私有协议服务**以及 **MQTT 协议网关服务**，实现互联互通。

MQTT 网关负责对用户的 MQTT 客户端提供服务，同时负责 MQTT 协议和后端 MQ 私有协议的转换。主要工作如下：

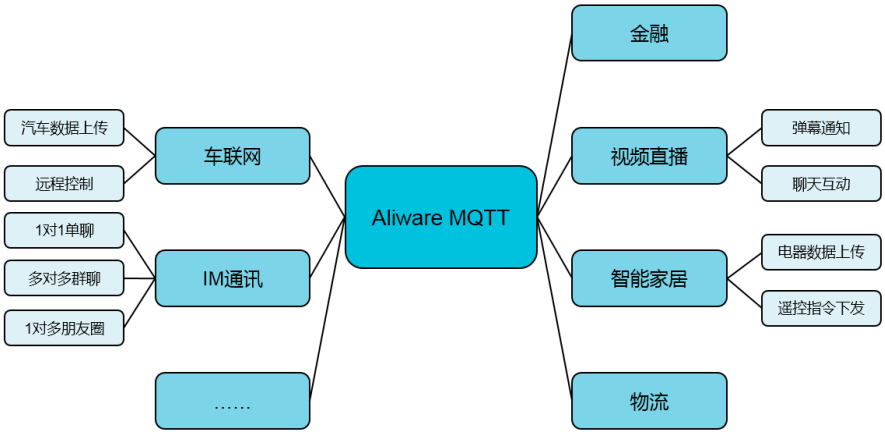
1. 提供 MQTT 服务，将用户的 MQTT 消息转换成后端 MQ 消息后，存储到 MQ 消息系统中，实现数据上行采集；
2. 接收来自 MQ 的消息，并将消息转换成对应的 MQTT 消息，推送给 MQTT 客户端，实现数据下行推送。

客户端分布：

- 使用 MQTT 协议客户端作为移动端接入 MQTT 网关，一般分布在公网环境，比如嵌入式设备、移动手机、平板、浏览器之类的平台上；
- 使用 MQ 协议客户端一般业务上的服务端接入 MQ 消息系统，一般部署在阿里云的 ECS 环境。

应用场景

MQTT 作为移动互联网以及物联网领域的主流协议，应用场景丰富。目前 MQ 提供的 MQTT 服务已经在各个领域有所应用，覆盖了直播互动、车联网、金融支付、即时聊天等多个场景。同时，MQTT 服务端采用分布式理念进行设计，系统无单点瓶颈，各个组件都可以无限水平扩展，保证容量可以随着您的在线使用量进行调整。



相比其他移动端消息服务，基于 MQ 的 MQTT 服务具有以下优势：

1. 支持标准的 MQTT 协议，应用方无技术捆绑，可以无缝迁移。
2. 可以支持移动端 MQTT 消息和服务端 MQ 消息的相互流转，实现服务端和移动端的双向打通。
3. 支持设备级权限控制，并支持 SSL/TLS 加密通信，数据传输更安全可靠。
4. 兼容任何支持 MQTT 3.1.1 协议的 SDK，覆盖绝大多数移动端开发语言和平台。

微消息队列名词解释

本文主要介绍 MQTT 协议的一些基本术语以及在阿里云环境使用 MQTT 涉及到的主要概念和术语。MQTT 接入的基本概念及使用场景请参考 MQTT 微消息队列。

资源类

Instance

用户创建购买 MQTT 服务的实体单元，每个 MQTT 实例都对应一个全局唯一的服务接入点 URL。使用 MQTT 前都需要在对应的 Region 创建一个实例，并使用对应的接入点来访问服务。如何创建 MQTT 实例请参考资源创建。

Parent Topic

MQTT 协议基于 Pub/Sub 模型，因此任何消息都属于一个 Topic。根据 MQTT 协议，Topic 存在多级，定义第一级 Topic 为父 Topic（Parent Topic），使用 MQTT 前，该 Parent Topic 需要先在 MQ 控制台创建。

Subtopic

MQTT 的二级 Topic，甚至三级 Topic 都是父 Topic 下的子类。使用时，直接在代码里设置，无需创建。需要注意的是 MQTT 限制 Parent Topic 和 Subtopic 的总长度为64个字符，如果超出长度限制将会导致客户端异常。

Client ID

MQTT 的 Client ID 是每个客户端的唯一标识，要求全局唯一，使用相同的 Client ID 连接 MQTT 服务会被拒绝。

Client ID 由两部分组成，组织形式为 GroupID@@@DeviceID。Client ID 的长度限制为64个字符，不要使用不可见字符。

- **Group ID**: 用于指定一组逻辑功能完全一致的节点共用的组名，代表一类相同功能的设备。Group ID 需要在 MQ 控制台创建方可使用，创建链接参考创建资源。
- **Device ID**: 每个设备独一无二的标识，由业务方自己指定。需要保证全局唯一，例如每个传感器设备的序列号。

权限类

Username

使用 MQTT 客户端收发消息时，MQ 会根据用户设置的 Username 和 Password 来进行鉴权。鉴权逻辑遵循阿里云统一的权限规范。此处 Username 设置为阿里云的 AccessKey 即可。

Password

MQ 要求用户将 GroupID 作为签名字段，SecretKey 作为秘钥，使用 HmacSHA1 算法计算签名字符串，并将签名字符串设置到 Password 参数中用于鉴权。关于鉴权的签名计算规则请参考签名计算章节文档。具体的代码实现可以参考各个语言版本的 demo 程序。

网络类

ServerUrl

MQ 提供的 MQTT 服务的接入点 URL，都是公网 URL，目前 MQTT 的接入除了支持标准协议的1883端口，同时还支持加密 SSL，WebSocket，Flash 等方式。接入点 URL 是在创建实例之后自动分配，请妥善保管。创建实例请参考资源创建章节。

协议相关

QoS

QoS （Quality of Service）指代消息传输的服务质量。它包括 QoS0（最多分发一次）、QoS1（至少达到一次）和 QoS2（仅分发一次）三种级别。

cleanSession

cleanSession 标志是 MQTT 协议中对一个客户端建立 TCP 连接后是否关心之前状态的定义。具体语义如下：

- cleanSession=true：客户端再次上线时，将不再关心之前所有的订阅关系以及离线消息。
- cleanSession=false：客户端再次上线时，还需要处理之前的离线消息，而之前的订阅关系也会持续生效。

注意：

- MQTT 要求每个客户端每次连接时的 cleanSession 标志必须固定，不允许动态变化，否则会导致离线消息的判断有误。
- MQTT 目前对外 QoS2 消息不支持非 cleanSession ，如果客户端以 QoS2 方式订阅消息，即使设置 cleanSession=false 也不会生效。
- P2P 消息的 cleanSession 判断以发送方客户端的配置为准。
- QoS 和 cleanSession 的不同组合产生的结果如下表所示：

QoS 级别	cleanSession=true	cleanSession=false
QoS0	无离线消息，在线消息只尝试推一次	无离线消息，在线消息只尝试推一次
QoS1	无离线消息，在线消息保证可达	有离线消息，所有消息保证可达
QoS2	无离线消息，在线消息保证只推一次	暂不支持

微消息队列使用限制

微消息队列使用限制

微消息队列对某些具体指标进行了约束和规范，您在使用微消息队列时注意不要超过相应的限制值，以免程序出现异常。具体的限制项和限制值请参见下表。

说明：企业铂金版用户可以根据自身需求定制部分指标（下表中已标出），咨询定制请提工单。

限制项	限制值	说明
Topic 长度	64 个字符	使用 MQTT 收发消息时，Topic 长度不得超过该限制

		，否则会导致无法发送或者订阅。
ClientId 长度	64 个字符	使用 MQTT 收发消息时，ClientId 长度不得超过该限制，否则会导致连接被断开。
消息大小	64 K 字节	消息负载不得超过该限制，否则消息会被丢弃（**企业铂金版可定制**）。
消息保存时间	3 天	目前仅当使用 QoS1 持久化 Session 时才会保留离线消息，且最多保留 3 天，超过时间将自动滚动删除（**企业铂金版可定制**）。
用户消息收发 TPS	参考实例规格	预付费实例会根据购买的规格进行限流，超过规格后将不保证消息可靠到达。后付费实例不会限流，但会默认提供监控报警。请合理调整监控阈值。
用户连接数	参考实例规格	预付费实例会根据购买的规格进行限流，超过规格后将不保证连接稳定性。后付费实例不会限流，但会默认提供监控报警。请合理调整监控阈值。
用户订阅关系数	参考实例规格	预付费实例会根据购买的规格进行限流，超过规格后将不保证订阅关系完整。后付费实例不会限流，但会默认提供监控报警。请合理调整监控阈值。
单个客户端订阅 Topic 数量	30 个	使用 MQTT 收发消息时，每个客户端最多允许同时订阅 30 个 Topic，超过该限制会导致无法新增新的订阅关系（**企业铂金版可定制**）。
QoS 和 CleanSession	目前不支持 QoS2 的持久 Session	使用 MQTT 收发消息时，服务端目前不支持 QoS2 方式的持久化 Session 订阅。如果需要接收离线消息，请使用 QoS1 方式的持久化 Session 订阅。
消息顺序性	上行顺序	使用 MQTT 收发消息时，目前只保证每个客户端发送消息的顺序性，如果消费消息需要顺序，需要使用 MQ TCP 方式接收消息。
离线消息可见时间	60 秒	服务端首次推送消息后必须要等待超时或者失败后才能确认该消息是否转化为离线消息，对应的延迟时间通常在 60 秒以内。
离线消息存储数量	1,000,000条（如未达到，请提工单请求处理）	服务端会限制每个实例存储的离线消息数量。超过该限制后，服务端会从最早的消息开始清理。

		因此，请合理使用持久化订阅模式，以免产生很多无用的离线消息。

Demo 工程

Demo 环境准备

本 Demo 主要目的在于帮助初次接触微消息队列的工程师，一步一步搭建微消息队列测试工程。Demo 程序以 Java 为例，包括使用 MQTT 协议收发 MQTT 消息，使用加密 SSL 协议发 MQTT 消息以及使用 MQ 协议收发 MQTT 消息的示例。

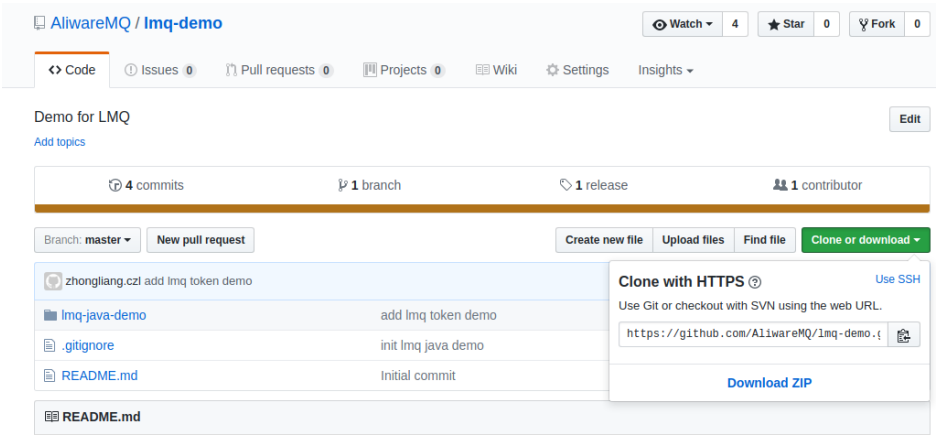
安装 IDE

您可以使用 IDEA 或者 Eclipse。本文以 IDEA 为例。

请在 https://www.jetbrains.com/idea/ 下载 IDEA Ultimate 版本，并参考 IDEA 说明进行安装。

下载微消息队列 Demo 工程

在 https://github.com/AliwareMQ/lmq-demo 下载微消息队列 Demo 工程到本地：



下载完成后解压即可看到本地新增了 lmq-demo 文件夹。

Demo 工程配置

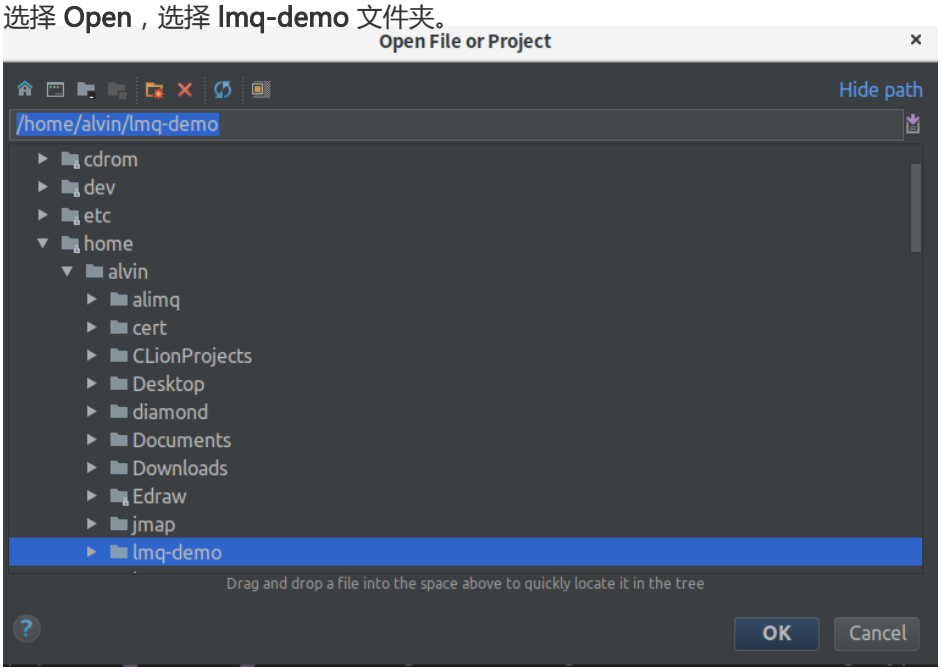
Demo 工程设置包含以下几个步骤。

微消息队列 Demo 工程文件导入 IDEA

注意：如果本地未安装 JDK，请先下载安装。

1. 双击 IDEA 图标打开 IDEA。

   选择 **Open**，选择 **lmq-demo** 文件夹。



   默认单击 Next，直到导入完成。Demo 工程需要加载依赖的 JAR 包，因此导入过程需要等待2-3分钟。

创建微消息队列资源

请在 MQ 控制台购买微消息队列实例，创建 Topic 和设备分组 ID (GID)，具体操作指导请查看微消息队列接入准备章节的资源创建。

配置微消息队列 Demo

需要配置文件 test.properties，具体功能如下所示：

- brokerUrl：连接 MQTT 服务的接入点地址，购买实例后即可从控制台获取到该地址。
- sslBrokerUrl：使用加密方式连接时需要配置此参数，和 brokerUrl 的域名一样，只是协议类型和端

口不一样。
- topic：在 MQ 控制台创建的收发消息的一级主题，MQTT 协议的子级主题不需要创建。
- accessKey：阿里云帐号中心获取。
- secretKey：阿里云帐号中心获取。
- groupId：MQTT 客户端配置的分组 ID，在控制台创建后使用。
- qos：收发消息设置的 QoS 级别。具体参考 MQTT 协议说明。
- cleanSession：设置客户端的会话是否使用持久化模式，具体参考 MQTT 完整协议说明。
- producerId：使用 MQ 客户端发消息时需要配置。
- consumerId：使用 MQ 客户端收消息时需要配置。

运行 Demo

以 MQTT 方式启动收发消息

以 Main 方式启动 MQTT 客户端收发消息：

1. 运行 MqttSimpleRecvDemo 类订阅消息，可以看见消息被接收打印的日志。
2. 运行 MqttSimpleSendDemo 类发送消息，可以看见消息被发送成功打印的日志。
3. 登录 MQ 控制台，在左侧菜单栏选择**消息查询**>**按 Topic 查询**，选择 Topic 名称进行查询，可以看见消息已经发送至 Topic。

以 MQ 方式启动收发消息

1. 运行 MqRecvMqttDemo 类接收来自 MQTT 客户端发送的消息。
2. 运行 MqSendMqttDemo 类向 MQTT 客户端发消息。

查看结果跟上面过程类似。

以 SSL 方式启动收发消息

运行 MqttSSLSendDemo 类使用加密方式连接服务端发送消息。

接入准备

环境准备

本文主要介绍 MQTT 接入需要完成的准备工作，包含 MQTT 客户端适配、服务接入配置等内容。

客户端适配

MQ 提供的 MQTT 服务严格遵循 MQTT3.1.1 协议设计，理论上能够适配所有的 MQTT 客户端，但不排除部分客户端存在细节上的兼容性问题。针对 MQTT 用户常用的平台，推荐对应的三方包如下：

使用平台	推荐的第三方 SDK	相关链接
Java	Eclipse Paho SDK	http://www.eclipse.org/paho/clients/java/
iOS	MQTT-Client-Framework	https://github.com/ckrey/MQTT-Client-Framework
Android	Eclipse Paho SDK	https://github.com/eclipse/paho.mqtt.android
JavaScript	Eclipse Paho JavaScript	http://www.eclipse.org/paho/clients/js/
Python	Eclipse Paho Python SDK	https://pypi.python.org/pypi/paho-mqtt/
C	Eclipse Paho C SDK	https://eclipse.org/paho/clients/c/
C#	Eclipse Paho C# SDK	https://github.com/eclipse/paho.mqtt.m2mqtt
Go	Eclipse Paho Go SDK	https://github.com/eclipse/paho.mqtt.golang

其他语言的客户端 SDK 如 PHP 等暂时没有提供测试。如有需要可以访问 https://github.com/mqtt/mqtt.github.io/wiki/libraries 进行下载。

服务接入配置

MQ 已经在阿里云各个 Region 开放 MQTT 服务，各个 Region 的接入点信息参见下表。同时，MQ 目前开放的 MQTT 服务除了支持标准的 MQTT 协议，还支持 MQTT SSL 、WebSocket、WebScoket TLS、Flash。对应的服务端口如下，请根据实际需求修改。

Region 名称	标准协议端口	SSL 端口	Websocket 端口	Websocket SSL 端口	Flash 端口	使用场景
公网	1883	8883	80	443	843	测试环境使用，Topic 资

						源仅限公网
华北2	1883	8883	80	443	843	华北2(北京)线上环境使用，Topic 资源仅限华北2
华东1	1883	8883	80	443	843	华东1(杭州)线上环境使用，Topic 资源仅限华东1
华东2	1883	8883	80	443	843	华东2(上海)线上环境使用，Topic 资源仅限华东2
华南1	1883	8883	80	443	843	华南1(深圳)线上环境使用，Topic 资源仅限华南1
新加坡	1883	8883	80	443	843	新加坡线上环境使用，Topic 资源仅限新加坡
金融云华东1	1883	8883	80	443	843	金融云华东1(杭州)线上环境使用，Topic 资源仅限金融云华东1
金融云华南1	1883	8883	80	443	843	金融云华南1(深圳)线上环境使用，Topic 资源仅限金融云华南1
金融云华东2	1883	8883	80	443	843	金融云华东2(上海)线上环境使用，Topic 资源仅限金

						融云华东2
政务云华北2	1883	8883	80	443	843	政务云华北2(北京)线上环境使用，Topic 资源仅限政务云华北2

注意：

使用 MQTT 时，一定要在对应的 Region 创建 Topic 资源，如果跨 Region 调用会收不到消息。

接入点获取

根据业务需求确定需要使用哪个 Region，同时根据业务选择对应的协议端口，然后在对应的 Region 创建实例，GroupID 等资源。创建实例后即可获得自己独享的域名接入点 URL。具体流程参考资源创建文档。

资源申请

使用 MQTT 收发消息需要先购买 MQTT 实例，并申请 Topic 和 MQTT Group ID 资源，否则服务端会拒绝非法的 Client ID 的连接。对于非 MQTT 协议的接入，比如新能源，808协议，必须购买企业铂金版才能支持。

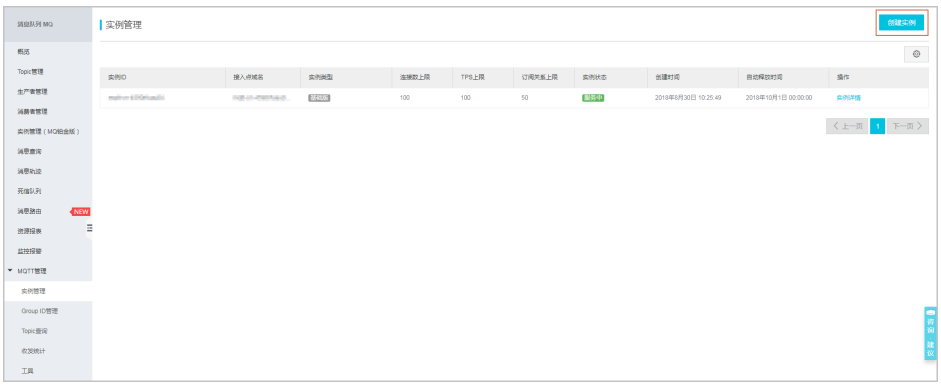
购买 MQTT 实例（2017 年 4 月 24 日商业化后）

您可以在 MQ 产品首页单击**购买套餐**，也可以直接登录 MQ 控制台购买实例。

在 MQ 控制台购买实例的操作如下：

用 RAM 主账号登录 MQ 控制台，在左侧导航栏选择 **MQTT 管理** > **实例管理**，然后单击**实例管理**页面右上角的**创建实例**按钮。

注意：RAM 子账号不能创建 MQTT 实例，在控制台也无法看到 MQTT 实例信息，请用主账号进行资源申请。

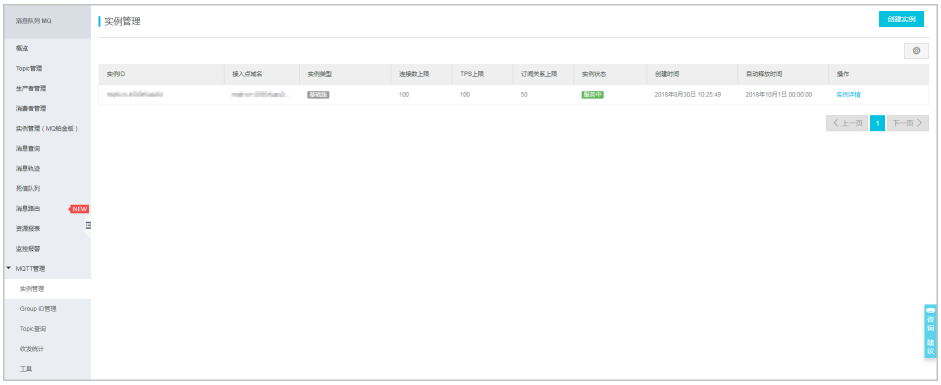


在阿里云产品购买页面，根据自己的业务场景选择 **微消息队列-预付费**或 **微消息队列-按量付费。**

注意：

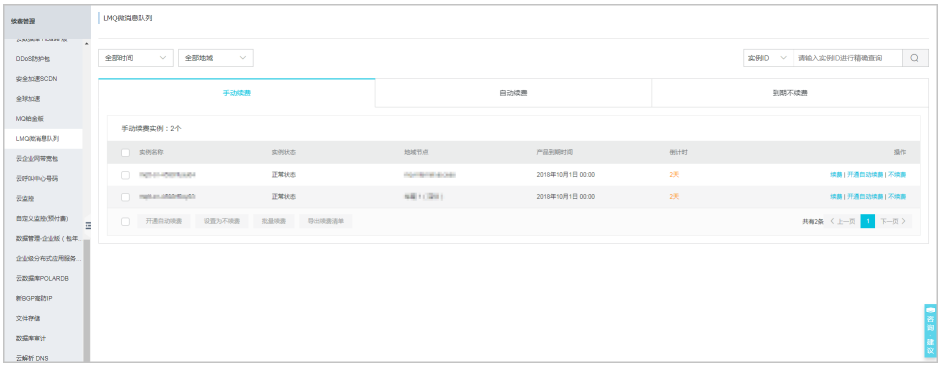
- 每个用户在每个 Region 下，每个类型的实例目前仅限购买1个，即一个用户在一个 Region 下最多拥有一个标准版和一个企业铂金版实例。
- 请根据业务场景估算 TPS、连接数和订阅关系，选择合理的规格。选择过小的规格会触发服务限流影响业务。
- 购买普通版实例实时生效，购买企业铂金版实例需要 MQ 后端人员介入部署，实例可运行时会通知用户验收。

购买完成后，您可在 MQ 控制台**实例管理**页面看到实例概况。



说明：此页面给出的接入点域名可用于业务代码中进行服务访问。

对于预付费实例，您可以在 MQ 控制台设置自动续费，以免由于人为因素造成实例过期，影响服务。具体操作路径是 **MQ 控制台** > **费用** > **续费管理** > **微消息队列**，最终跳转至以下页面。



创建 MQTT Topic

使用 MQTT 协议收发消息，需要创建 MQTT 的 Parent Topic（一级父 Topic）。多级子 Topic 无需创建，直接在代码中使用即可。

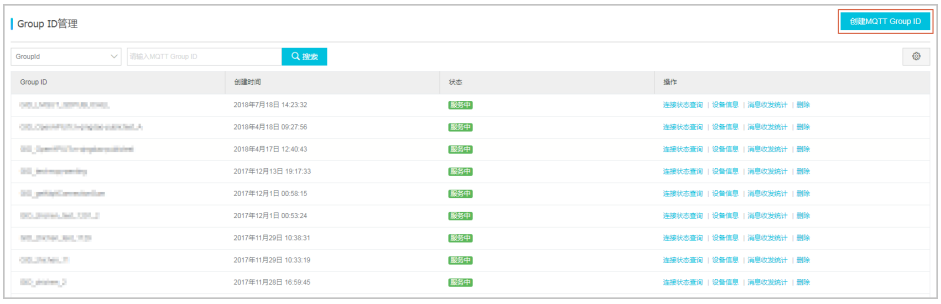
如之前在使用 MQ 时已经创建过 Topic，直接使用即可。如果没有创建过，请参考**快速入门**指导中的**创建消息主题（Topic）**。

注意：如果使用的 Topic 类型为顺序消息，在 MQTT 场景下不保证消息顺序。因此建议使用无序 Topic。

创建 MQTT Group ID

MQTT Group ID 用于指定一组逻辑功能完全一致的节点共用的组名，代表一类相同功能的设备。Group ID 和 Device ID 共同组成用于识别 MQTT 客户端的 Client ID。

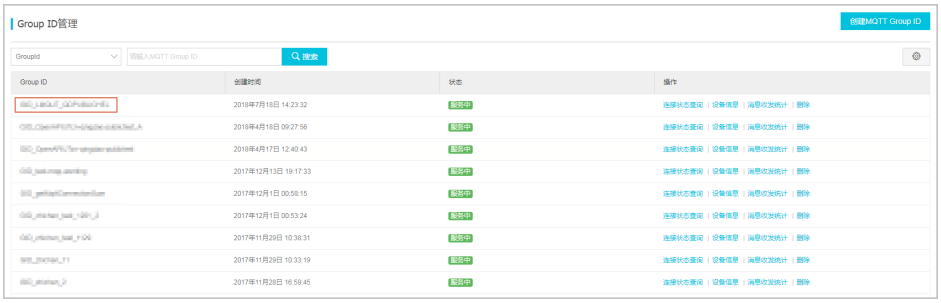
在 MQ 控制台的左侧导航栏，选择 **MQTT管理**> **Group ID管理**，然后在 **GroupID管理**页面，单击右上角的**创建MQTT Group ID**按钮。



在**创建MQTT Group ID** 对话框中，输入需要创建的 Group ID，然后单击**确定**。

注意： MQTT Group ID 长度限制在7-32字符，必须以“GID_”开头，内容仅限数字和大小写字母。如果输入不合法字符或者长度不符合，将无法创建 MQTT Group ID。

创建完成后，在**Group ID管理**页面，**Group ID** 列表中显示创建的 Group ID。该页面显示当前 Region 下当前账号拥有的 MQTT Group ID。



注意：

- 目前华北 1 Region 暂不支持 MQTT，因此控制台上无法创建对应的资源。
- 如果 Group ID 不再使用，请及时删除。
- Group ID 仅限创建账号使用，父账号创建的 Group ID 子账号无法使用，子账号必须单独创建，反之同理。

MQTT 签名计算

本文档介绍使用 MQTT 收发消息中需要用到的签名的计算方式以及使用控制台工具生成签名的方法。

使用 MQTT 收发消息，服务端需要对客户端的身份进行权限校验，因此客户端请求中都需要带上签名以便比对身份。

MQTT SDK 访问消息服务器

MQTT 客户端实际连接 MQTT 消息服务器时，在 connect 报文中需要上传 username 和 password。其中 username 就是 AccessKey，password 则是将 Group ID 作为待签名字符串，用 SecretKey 作为秘钥计算得到的签名。

比如客户端的 Client ID 是 GID_AAA@@@BBB001。

此时待签名字符串就是取 Client ID 的前缀，即 Group ID，“GID_AAA”。

然后用 SecretKey 作为秘钥，使用 HmacSHA1 方法对上面的待签名字符串做签名计算得到一个二进制数组

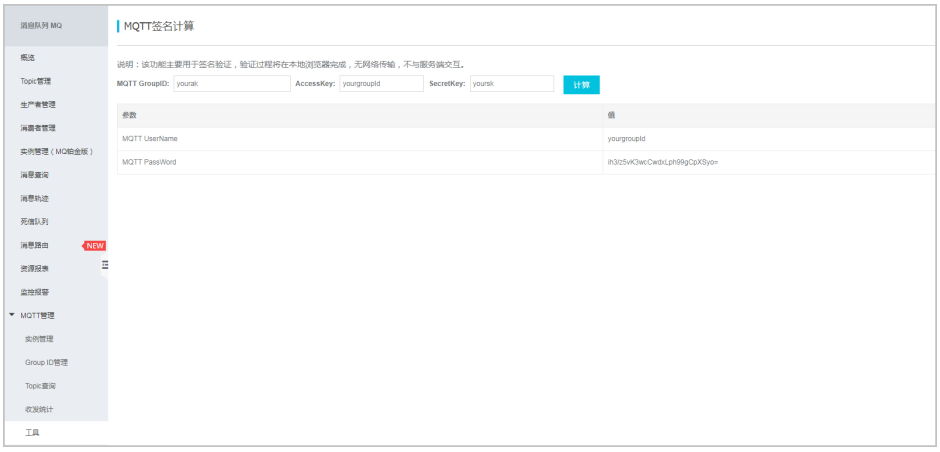
，最后对该二进制数组做 Base64 编码得到最终的 password 签名字符串，即 “eqweq+adwe23fssf” 。

Hmac 的算法实现，各个语言都有现成的函数库，请自行搜索。

使用控制台生成签名

为方便用户比对验证自己的签名计算是否正确，微消息队列控制台提供了签名计算工具供参考比对。

输入程序使用的帐号 AccessKey，SecretKey 以及 GroupId，即可得到程序中需要设置的 UserName 和 PassWord 参数。具体参考如下：



注意：此工具仅仅使用浏览器前端 JavaScript 完成计算，并不会传输 SecretKey 到 MQ 后端，因此不用担心 SecretKey 泄漏的风险。

MQTT 获取离线消息

离线消息使用场景

- 场景一：客户端对于离线消息的优先级比较低，只要求离线消息最终能被处理即可。
- 场景二：客户端对于离线消息需要优先处理，且要求比较实时。

针对场景一，因为 MQTT 默认的工作模式即可支持，客户端上线后，离线消息不会立即推送，是按照固定时间间隔，固定数量的方式推送。这种工作模式的好处是可以降低客户端上线时的压力，优先处理在线消息，离线消息只要保证最终能处理即可。该模式不需要任何设定。

针对场景二，因为客户端需要自己控制离线消息优先处理，所以 MQTT 提供了主动拉的模式来供客户端获取离线消息。即在客户端上线后，客户端自己调用接口来拉取自己所需的指定数量的消息。此模式下，客户端自己控制拉取的时间间隔和条数。

主动拉取离线消息使用说明

具体步骤如下：

1. 客户端启动后，以控制消息的形式发起拉取消息的指令，设置拉取条数和顺序。
2. 客户端等待本地处理成功。
3. 继续拉取下一批消息。

注意事项

- 服务端需要等待某消息首次推送失败或者推送超时后，才能确认是否将该消息转化为离线消息。因此，您在发送消息后立即使用主动拉取的模式，很可能看不到离线消息，必须等待离线消息可见之后才能获取到。关于离线消息的可见时间，参见使用限制。
- 客户端如果需要使用主动模式，请务必在连接建立后的第一个心跳周期内发起请求，否则系统会按照自动模式，即按照固定周期推送离线数据。
- 客户端每次最多拉取消息的数量为30条，客户端发起拉取请求的最大频率限制为5次/秒。
- 客户端需要自己控制拉取时机，因为消息从发起指令到推送到客户端，到客户端消费完成回应 ACK 都是异步过程。如果客户端拉取过快，很有可能拉到前一批还没有删除的消息，造成重复；或者拉取到重复的消息，因为前一次的消息还没有回复 ACK。

拉取离线消息相关 API

拉取离线消息：

发送 Topic：$SYS/getOfflineMsg

内容：JSONString

内容信息：

名称	类型	说明
pushOrder	String	“DESC”或者”ASC”，分别代表从最新消息拉取还是从最早消息拉取
maxPushNum	Integer	一次最多拉取消息的条数，设置范围为1-30，超过会以上限计算

返回值

普通的 PubAck 报文。

示例程序

MQTT 客户端使用 Demo

```
public void testOfflineMsg() throws Exception {
String broker = "tcp://XXXX:1883";
String clientId = "GID_XXX@@@XXXX";
final String topic = "XXXX/11";
final String topicFilter[] = {topic};
final int qos[] = {1};
MemoryPersistence persistence = new MemoryPersistence();
try {
final MqttClient sampleClient = new MqttClient(broker, clientId, persistence);
final MqttConnectOptions connOpts = new MqttConnectOptions();
System.out.println("Connecting to broker: " + broker);
connOpts.setServerURIs(new String[]{broker});
connOpts.setCleanSession(false);
connOpts.setAutomaticReconnect(true);
/**
* 客户端长链接需要设置心跳时间，建议100s 以下，超时，服务端会断开连接
*/
connOpts.setKeepAliveInterval(90);
sampleClient.setCallback(new MqttCallbackExtended() {
public void connectComplete(boolean reconnect, String serverURI) {
System.out.println("connect success");
sampleClient.subscribe(topicFilter, qos);
}
public void messageArrived(String topic, MqttMessage mqttMessage) throws Exception {
System.out.println("recv Msg from " + topic);
}
public void deliveryComplete(IMqttDeliveryToken iMqttDeliveryToken) {
System.out.println("deliveryComplete:" + iMqttDeliveryToken.getMessageId());
}
});
sampleClient.connect(connOpts);
JSONObject object = new JSONObject();
object.put("maxPushNum", 20);
object.put("pushOrder", "DESC");
sampleClient.publish("$SYS/getOfflineMsg", new MqttMessage(object.toJSONString().getBytes()));
Thread.sleep(1000000);
} catch (Exception e) {
e.printStackTrace();
}
}
```

获取客户端在线状态

本文介绍如何获取 MQTT 客户端当前在线情况。

应用场景

- 主业务流程中需要根据客户端是否在线来决定后续运行逻辑；

- 运维过程需要判断特定客户端当前是否在线；
- 服务端需要在客户端上线或者下线时触发一些预定义的动作。

基本原理

MQTT 服务端对上述功能需求提供了2种获取方式，分别是同步查询接口以及异步上下线通知。

同步查询接口的方式相对简单，即用户通过开放的接入点地址调用 HTTP 方式的 API 查询特定客户端的在线情况，该方式只能查询单个客户端当前实时状态，适用于对单个客户端的状态判断。

异步上下线通知则是利用消息通知的方式，在客户端上线和下线事件触发时， MQTT 服务器会向后端 MQ 推送一条上下线消息。用户的服务端一般部署在阿里云的服务器上。用户的服务端通过订阅这条 MQ 消息来获取所有客户端的上下线动作。该方式属于异步感知客户端的状态，且感知到的是上下线事件，而非在线状态，应用需要根据事件发生的时间序列分析出客户端的状态。

注意：同步查询接口是查询当前客户端的实时状态，理论上比异步通知的方式更精确，但只能查询单个客户端状态。上下线通知因为采用消息解耦，状态判断更加复杂，且误判可能性更大，但该方法可以基于事件分析多个客户端的运行状态轨迹。

同步查询接口（公测期间）

MQTT 服务端对外通过 HTTP协议方式暴露查询接口，查询接口的使用文档如下：

注意：同步查询接口功能目前处于公测期间，仅限公网测试 region 使用，后续会在其他 region 开放服务。

查询接口

```
https://mqtt-xxx.mqtt.aliyuncs.com/route/clientId/get
```

其中：

- 接口地址复用购买实例的接入点域名。
- 协议可选 HTTP 或者 HTTPS，调用方式仅限 GET 方法。

参数列表

参数名称	参数类型	说明
accessKey	String	当前请求使用的账号的 AK
resource	String	需要查询的客户端 clientId，只能查询当前账号拥有的 clientId
timestamp	Long	构建当前请求的UNIX 毫秒时间戳，时间戳过期时间为当前真实时间前后5分钟
signature	String	对当前请求参数计算得到的签名

		字符串，服务端用于安全认证

其中：

- timstamp 参数用于阻止非法过期请求，需要设置为当前真实时间，系统容忍前后5分钟误差区间。
- signature 为请求参数的签名，将其他参数作为待签名字符串，使用账户 SK 计算得到，用于权限验证和防止请求篡改。具体计算方式参考文档。

返回结果

MQTT 服务端接收到查询请求后，会进行接口参数验证，对于合法的查询请求返回当前 客户端的在线连接数。具体返回情况如下表所示：

HTTP 状态码	说明
400	非法请求，可能是接口 URL 不正确，或者参数缺失
403	权限验证失败，可能是签名计算错误，或者没有权限查询对应的 clientId 状态
200	请求处理成功

返回结果中，除 HTTP 状态码，HTTP content 中也会包含对应的返回结果，返回结果为 JSON 格式，具体的字段映射如下：

字段名称	类型	说明
code	int	请求处理结果状态码，用于判断当前请求是否成功和失败原因
success	boolean	true/false，代表查询是否成功
online	int	当前客户端维持的 tcp 连接数，可能存在-1，代表查询结果未知，建议保守判断为在线
message	string	请求处理结果描述，用于判断异常原因

注意：查询结果中 online 字段为0代表不在线，大于0代表在线，具体数字为在线的 tcp 连接数，应用方应注意该字段大于1可能存在重复连接情况。因服务端数据同步延迟，会有很小概率出现返回-1的情况，代表查询结果未知，建议应用重试，或者基于实际场景做保守判断。

其中 code 错误码说明如下：

code	说明
0	请求处理成功
1	参数缺失，建议检查参数对是否完整
2	签名计算错误，建议检查签名计算过程
3	权限验证错误，建议检查当前账号是否创建过该设

	备所属的 GroupId
4	请求时间戳过期，建议检查 timestamp 字段是否为最近5分钟内

异步上下线通知

如上文所述，使用异步上下线通知的方式，上下线动作的这个事件会映射到 MQ 的消息中，该消息对应的 Topic 命名规范是：**“GroupID”拼接后缀“_MQTT”**

例如您需要关注 GID_XXX@@@YYYYY 类型的 MQTT 客户端，那么对应的事件 Topic 就是 GID_XXX_MQTT。具体接入步骤参考如下：

1. 创建事件 Topic

根据上文原理介绍，用户关注哪些 Group ID 分组的设备，就创建对应的事件 Topic，创建 Topic 方法请参考**快速入门指导中的步骤二：创建资源。**

2. 服务端订阅消息

使用上一步骤中创建的 Topic，即可收到关注的 MQTT 客户端上下线事件。MQ 的接收程序请参考**订阅消息。**

其中，数据格式如下：

事件类型放在 MQ 的 Tag 中，代表是上线还是下线。

MQTag：connect/disconnect/tcpclean

其中：

- connect 事件代表客户端上线动作。
- disconnect 事件代表客户端主动断开连接。按照 MQTT 协议，客户端主动断开 TCP 连接之前应该发送 disconnect 报文，服务端在收到 disconnect 报文后触发该类型消息。如果某些客户端 SDK 没有按照协议发送 disconnect 报文，服务端相应无法收到该消息。
- tcpclean 事件代表实际的 TCP 连接断开。无论客户端是否显示发送过 disconnect 报文，只要当前 TCP 连接断开就会触发 tcpclean 事件。

注意： tcpclean 消息代表客户端网络层连接的真实断开。对应的，disconnect 消息仅仅代表客户端是主动发送了下线报文。受限于客户端的实现，有时候客户端异常退出会导致 disconnect 消息并没有正常发送。因此判断客户端下线请使用 tcpclean 事件。

数据内容为 JSON 类型，相关的 Key 如下：

- ClientId 代表具体设备；
- time 代表本次事件的时间；
- eventType 代表事件类型，供 MQTT 客户端区分事件类型；
- channelId 代表每个 TCP 连接的唯一标识；

示例：

```
clientId：GID_XXX@@@XXXXX
time:1212121212
eventType:connect/disconnect/tcpclean
channelId:2b9b1281046046faafe5e0b458e4f553
```

注意：判断客户端当前是否在线不能仅仅根据收到的最后一条消息的状态，而需要结合上下线消息的前后关联来判断。

具体判断规则如下：

- 同一个 clientId 的客户端，产生上下线事件的先后顺序以时间为准。越大则越新。
- 同一个 clientId 的客户端，可能存在多次闪断，因此，当收到下线消息时，一定要根据 channelId 字段判断是否是当前的 TCP 连接。简而言之，下线消息只能覆盖 channelId 相同的下线消息，如果下线消息的 channelId 不一样，尽管 eventIndex 较新，也不能覆盖。

获取状态的 Demo 如下：

```
public static void main(String[] args) throws InterruptedException {
/**
* 设置阿里云的 AccessKey，用于鉴权
*/
final String acessKey ="XXXXXX";
/**
* 设置阿里云的 SecretKey，用于鉴权
*/
final String secretKey ="XXXXXXX";
/**
* 上述步骤中涉及的事件 Topic，需要先在 MQ 控制台里创建
*/
final String topic ="GID_XXX_MQTT";
/**
* ConsumerID,需要先在 MQ 控制台里创建
*/
final String consumerID ="CID_XXXX";
Properties properties =new Properties();
//PropertyKeyConst.ONSAddr 地址请根据实际情况对应以下几类进行输入：
//公共云生产环境：http://onsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal
//公共云公测环境：http://onsaddr-internet.aliyun.com/rocketmq/nsaddr4client-internet
//杭州金融云环境：http://jbponsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal
//杭州深圳云环境：http://mq4finance-sz.addr.aliyun.com:8080/rocketmq/nsaddr4client-internal
//亚太东南1公共云环境（只适用于新加坡 ECS）：http://ap-southeastaddr-
internal.aliyun.com:8080/rocketmq/nsaddr4broker-internal
properties.put(PropertyKeyConst.ONSAddr,
"http://onsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal");//此处以公共云生产环境为例
properties.put(PropertyKeyConst.ConsumerId, consumerID);
properties.put(PropertyKeyConst.AccessKey, acessKey);
properties.put(PropertyKeyConst.SecretKey, secretKey);
Consumer consumer =ONSFactory.createConsumer(properties);
/**
* 处理收到的事件，根据 Tag 区分是上线还是下线，body 是个 JSON 字符串
*/
```

```
consumer.subscribe(topic, "*", new MessageListener() {
public Action consume(Message message, ConsumeContext consumeContext) {
String event = message.getTag();
if(event.equals("connect")){
// this is connect event
}else if(event.equals("disconnect")){
// this is client disconnect event
}else if(event.equals("tcpclean")){
// this is tcp disconnect
}
String body = new String(message.getBody());
JSONObject object = JSON.parseObject(body);
String clientId = object.getString("clientId");
long time =object.getLong("time");
return Action.CommitMessage;
}
});
consumer.start();
System.out.println("[Case Normal Consumer Init] Ok");
Thread.sleep(Integer.MAX_VALUE);
consumer.shutdown();
System.exit(0);
}
```

Token 服务

MQTT Token 服务简介

MQTT 通过 Token 鉴权服务向 MQTT 客户端提供短期访问权限。通过 MQTT Token 服务可以给本地账号系统管理的用户颁发临时的访问凭证，并限制访问权限，以实现对单一客户端，单一资源的细粒度权限控制。

访问 MQTT 资源的权限认证一般分为以下两个场景。

- 场景一：使用 MQTT 的业务方单一，所有 MQTT 客户端运行的环境可控，逻辑意义上都属于一个账号。
- 场景二：用户自己有本地账号系统，需要对阿里云账号的身份做二次拆分，所有 MQTT 客户端可能隶属于不同的本地账号范围。

针对场景一，因为在业务层面上，所有使用 MQTT 的场景都归属于一个账号，所以没有必要再做身份的识别和划分。使用 MQTT 默认的权限控制系统即可，即所有客户端直接配置阿里云账号的 AccessKey 和签名即可识别身份。

针对场景二，因为运行的 MQTT 客户端程序虽然隶属于同一个阿里云账号，但是还需要有本地账号（业务部门或者单一客户端）的角色区分。此时场景一划定的阿里云账号维度就无法满足。特别是在移动端场景，客户端面临破解劫持的风险，因此权限控制需要细化到单一客户端，且权限粒度需要细化到单一资源，而非账号维度。

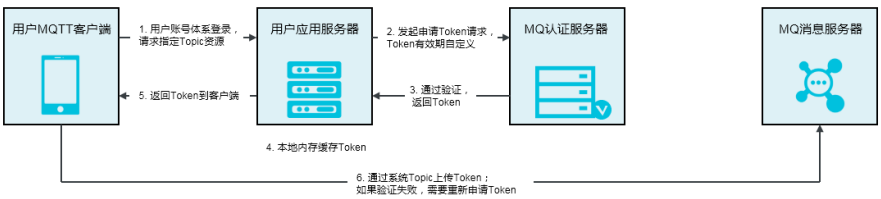
MQTT Token 服务即用来解决上述场景二的需求。用户作为本地账号管理者，为实际客户端申请临时访问凭证（Token），客户端使用临时凭证来做实际的业务访问。Token 代表单一客户端的临时身份，由业务管理者指定该 Token 所拥有的权限、资源列表和访问类型。具体名词解释如下：

术语	中文名称	说明
Token	临时凭证	MQ 颁发的临时访问凭证，代表单一客户端对特定资源的访问权限
AppServer	应用服务器	用户管理本地账号的服务器，用来替客户端申请和管理 Token 服务的应用
AuthServer	认证服务器	MQ 权限认证服务器，用来处理 AppServer 发起的 Token 相关的请求

MQTT Token 鉴权流程

鉴权流程

MQTT Token 鉴权流程如下图所示。



具体步骤如下：

1. 客户端启动时，需要先连接自己的应用服务器验证身份。
2. 客户端向应用服务器申请自己所需的所有 Topic 的权限。
3. 应用服务器验证客户端是否有必要操作所需的 Topic 的权限，如果验证通过则向 MQ 认证服务器申请对应资源的 Token。
4. MQ 认证服务器验证申请 Token 的请求，判断合法之后返回对应的 Token。
5. MQTT 客户端将应用服务器返回的 Token 持久化到本地，对每个客户端需要的权限以及 Token 信息做一个映射。

- 客户端重新启动进行访问时，应用服务器可以返回缓存的 Token，避免重复申请；
- 客户端重新申请 Token，认证服务器如果无响应，应用服务器可以尝试返回客户端之前申请的 Token。

6. MQTT 客户端连接 MQTT 消息服务器。
7. 客户端在收发消息之前必须通过 QoS1 方式上传自己的 Token。上传 Token 通过发送一条特殊的消息来实现，具体消息格式参考下文。
8. 客户端正常收发消息。如果服务端判断 Token 失效，即会触发鉴权失败，客户端应该重新发起申请 Token 的请求。
9. 在 Token 失效前 5 分钟，服务端也会推送一次即将失效的通知，客户端可以监听该消息以便提前更换 Token。

客户端行为约束

1. 发送 Pub/Sub 报文前，必须先采用 MQTT 控制报文来上传 Token。
2. 必须以同步发送模式上传 Token，且消息 QoS 必须设置为 1，以保证服务端未返回之前不会收发消息，否则会报错甚至断开链接。
3. 必须监听 Proxy 可能下推的 Token 失效的通知消息。
4. 收到 Token 失效的消息后，客户端必须重新申请 Token。
5. 客户端应该对应用服务器返回的 Token 做持久化，避免每次重连都申请一样的 Token。
6. 客户端发现连接被断开时应该先判断自己的 Token 是否失效，如果失效需要先重新申请 Token 再重连。判断标准可以参考服务端下推的消息，以及 Token 的有效周期。

应用服务器行为约束

1. 应用服务器必须验证自己的客户端是否合法，避免客户端冒用身份申请 Token。
2. 应用服务器应该对 Token 和客户端的关系进行管理，避免同一个客户端重复调用。
3. 应用服务器需要做本地容灾，避免因访问认证服务器短暂不可用导致的业务阻塞。

相关 API

MQTT 的 Token 鉴权流程通过相关的 API 来完成。

Token 的申请和吊销管理交由应用服务器负责，和 MQ 认证服务器之间通过 HTTPS 的 Open API 进行交互。每个接口都要求通过 AccessKey 和请求签名做来做身份验证。目前开放申请 Token、吊销 Token 以及校验 Token 接口。

MQTT 客户端则包括上传 Token、监听 Token 失效信息，以及监听 Token 非法信息三个接口。

Token 服务地址

目前 Token 服务支持国内公共云地域和新加坡地域，暂不支持金融云。具体的服务地址如下：

服务地域	Token 服务器地址
华北2	mqauth.aliyuncs.com
公网	mqauth.aliyuncs.com

华东1	mqauth.aliyuncs.com
华东2	mqauth.aliyuncs.com
华南1	mqauth.aliyuncs.com
新加坡	mqauth.ap-southeast-1.aliyuncs.com

签名机制说明

您的应用服务器访问认证服务器申请和管理 Token 时，需要将请求参数按照指定方式排序拼接，组织成待签名字符串，然后使用秘钥 secretKey 加密得到签名。

具体约束规则如下：

- 参数对使用 key=value 格式；
- 参数对之间使用“&”分隔；
- 参数对内部，即同名参数，使用逗号“,”分隔；
- 计算签名时，需要将参数对按照 key 的字典序排序，参数对内部，也需要使用字典序排序。

示例

比如原始请求的 HTTP 方法是：

```
https://mqauth.aliyuncs.com/token/apply
```

参数列表有：

```
parama=a
paramc=c2,c1
paramb=b2,b1,b3
```

那么先将参数按照 key 排序，得到：

```
parama=a
paramb=b2,b1,b3
paramc=c2,c1
```

再将同一参数内的值也按照字典序排序，参数内部用逗号分隔，得到：

```
parama=a
paramb=b1,b2,b3
paramc=c1,c2
```

最终拼接得到待签名字符串：

```
parama=a&paramb=b1,b2,b3&paramc=c1,c2
```

然后使用 secretKey 作为密钥，HmacSHA1 算法计算得到签名。

注意：

- HTTP 管理 Token 时，可以选用 HTTP 或者 HTTPS，应用自行选择。
- HTTP 方法可以选择 GET 或者 POST，建议生产环境使用 POST，测试环境可以使用 GET。

公共返回值

HTTP 请求到达认证服务器后，正常处理后会返回给调用方响应，类型是 JSON 字符串，调用方可根据需求取出 JSON 中特定的值。

JSON 字符串的 Key-Value 映射如下：

Key	类型	说明
success	boolean	是否成功。
message	String	处理结果信息，或者异常描述。
code	int	返回码，可根据 code 判断当前请求处理情况或者错误类型。
tokenData	String	Token 字符串，申请 Token 的接口会返回该值。

其中，错误码列表如下：

Code	说明
200	成功
400	参数校验错误
407	签名计算错误
409	Token 生成处理错误
410	Token 吊销处理失败
1	伪造 Token，不可解析
2	Token 已经过期
3	Token 已经被吊销

申请 Token 接口

接口

https://{tokenServerUrl}/token/apply

请求参数

名称	类型	说明
actions	String	Token 的权限类型，有“R”（读），“W”（写），“R,W”（读和写）三种类型。**注意：**如果同时需要读和写的权限，“R”和“W”之间需要用逗号隔开。
resources	String	资源名称，即 MQTT Topic，多个 Topic 以逗号“,”分隔，每个 token 最多运行操作100个资源。
accessKey	String	当前请求使用的账号的 AK。
expireTime	long	Token 有效期，最小间隔是60秒。
proxyType	String	Token 类型，填 MQTT，HTTP 或 Kafka，MQ，根据实际产品选择。
serviceName	String	填 mq，其他参数无效 。
signature	String	签名字符串，本请求需要计算签名的字段是 actions，resources，expireTime，serviceName。

示例

```
public void applyToken() throws InvalidKeyException, NoSuchAlgorithmException, IOException {
String apiUrl="https://XXXXXX/token/apply";
Map<String,String >paramMap=new HashMap<String, String>();
paramMap.put("resources","topic1/1,topic2/2");
paramMap.put("actions","R,W");
paramMap.put("serviceName","mq");
paramMap.put("expireTime", String.valueOf(System.currentTimeMillis()+1000000));
String signature= Tools.doHttpSignature(paramMap,"XXXX");
paramMap.put("proxyType","MQTT");
paramMap.put("accessKey","XXXXX");
paramMap.put("signature",signature);
```

```
JSONObject object = Tools.httpsPost(apiUrl,paramMap);
System.out.println(object);
}
```

接口所需的工具方法及示例代码请参考文档**工具方法类**。

吊销 Token 接口

接口

https://{tokenServerUrl}/token/revoke

请求参数

名称	类型	说明
token	String	需要做吊销处理的 Token
accessKey	String	当前请求使用的账号的 AK
signature	String	签名字符串，本请求需要计算签名的字段是 Token

示例

```
public void revokeToken() throws InvalidKeyException, NoSuchAlgorithmException, IOException {
String apiUrl="https://XXXXX/token/revoke";
Map<String,String >paramMap=new HashMap<String, String>();
paramMap.put("token","XXXXXXXXXXXXXXX");
String signature=Tools.doHttpSignature(paramMap,"XXXX");
paramMap.put("signature",signature);
paramMap.put("accessKey","XXXX");
JSONObject object = Tools.httpsPost(apiUrl,paramMap);
System.out.println(object);
}
```

接口所需的工具方法及示例代码请参考文档**工具方法类**。

查询 Token 接口

接口

https://{tokenServerUrl}/token/query

请求参数

名称	类型	说明
token	String	需要查询的 Token
accessKey	String	当前请求使用的账号的 AK
signature	String	签名字符串，本请求需要计算签名的字段是 Token

接口所需的工具方法及示例代码请参考文档工具方法类。

工具方法类

本文给出各个接口所需的工具方法类，包含如何计算签名，如何发起 HTTPS 请求等接口。

使用该测试类需要添加 HTTP 客户端的依赖，参考如下：

```
<dependency>
 <groupId>org.apache.httpcomponents</groupId>
 <artifactId>httpclient</artifactId>
 <version>4.5.2</version>
</dependency>
```

测试类仅仅给出一个 Demo 程序，实际线上使用时，请根据业务情况自行改造 HTTP 请求的代码。

```
public class Tools {
    /**
     * 计算签名，参数分别是参数对以及密钥
     *
     * @param requestParams 参数对，即参与计算签名的参数
     * @param secretKey     密钥
     * @return 签名字符串
     * @throws NoSuchAlgorithmException
     * @throws InvalidKeyException
     */
    public static String doHttpSignature(Map<String, String> requestParams, String secretKey) throws
NoSuchAlgorithmException, InvalidKeyException {
        List<String> paramList = new ArrayList<String>();
        for (Map.Entry<String, String> entry : requestParams.entrySet()) {
            paramList.add(entry.getKey() + "=" + entry.getValue());
        }
        Collections.sort(paramList);
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < paramList.size(); i++) {
```

```
            if (i > 0) {
                sb.append('&');
            }
            sb.append(paramList.get(i));
        }
        return macSignature(sb.toString(), secretKey);
    }

    /**
     * @param text      要签名的文本
     * @param secretKey 阿里云 MQ secretKey
     * @return 加密后的字符串
     * @throws InvalidKeyException
     * @throws NoSuchAlgorithmException
     */
    public static String macSignature(String text, String secretKey) throws InvalidKeyException,
NoSuchAlgorithmException {
        Charset charset = Charset.forName("UTF-8");
        String algorithm = "HmacSHA1";
        Mac mac = Mac.getInstance(algorithm);
        mac.init(new SecretKeySpec(secretKey.getBytes(charset), algorithm));
        byte[] bytes = mac.doFinal(text.getBytes(charset));
        return new String(Base64.encodeBase64(bytes), charset);
    }

    /**
     * 创建 HTTPS 客户端
     *
     * @return 单例模式的客户端
     * @throws KeyStoreException
     * @throws UnrecoverableKeyException
     * @throws NoSuchAlgorithmException
     * @throws KeyManagementException
     */
    private static HttpClient httpClient = null;

    public static HttpClient getHttpsClient() throws KeyStoreException, UnrecoverableKeyException,
NoSuchAlgorithmException, KeyManagementException {
        if (httpClient != null) {
            return httpClient;
        }
        X509TrustManager xtm = new X509TrustManager() {
            @Override
            public void checkClientTrusted(X509Certificate[] arg0, String arg1)
                    throws CertificateException {
            }

            @Override
            public void checkServerTrusted(X509Certificate[] arg0, String arg1)
                    throws CertificateException {
            }

            @Override
            public X509Certificate[] getAcceptedIssuers() {
                return new X509Certificate[]{};
            }
```

```
        };
        SSLContext context = SSLContext.getInstance("TLS");
        context.init(null, new TrustManager[]{xtm}, null);
        SSLConnectionSocketFactory scsf = new SSLConnectionSocketFactory(context,
NoopHostnameVerifier.INSTANCE);
        Registry<ConnectionSocketFactory> sfr = RegistryBuilder.<ConnectionSocketFactory>create()
                .register("http", PlainConnectionSocketFactory.INSTANCE)
                .register("https", scsf).build();
        PoolingHttpClientConnectionManager pcm = new PoolingHttpClientConnectionManager(sfr);
        httpClient = HttpClientBuilder.create().setConnectionManager(pcm).build();
        return httpClient;
    }

    /**
     * 发起 HTTPS Get 请求，并得到返回的 JSON 响应
     *
     * @param url    接口 URL
     * @param params 参数对
     * @return
     * @throws IOException
     * @throws KeyStoreException
     * @throws UnrecoverableKeyException
     * @throws NoSuchAlgorithmException
     * @throws KeyManagementException
     */
    public static JSONObject httpsGet(String url, Map<String, String> params) throws IOException,
KeyStoreException, UnrecoverableKeyException, NoSuchAlgorithmException, KeyManagementException {
        HttpClient client = Tools.getHttpsClient();
        JSONObject jsonResult = null;
        //发送 get 请求
        List<NameValuePair> urlParameters = new ArrayList<NameValuePair>();
        for (Map.Entry<String, String> entry : params.entrySet()) {
            urlParameters.add(new BasicNameValuePair(entry.getKey(), entry.getValue()));
        }
        String paramUrl = URLEncodedUtils.format(urlParameters, Charset.forName("UTF-8"));
        HttpGet request = new HttpGet(url + "?" + paramUrl);
        HttpResponse response = client.execute(request);
        if (response.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {
            String strResult = EntityUtils.toString(response.getEntity());
            jsonResult = JSON.parseObject(strResult);
        }
        return jsonResult;
    }
  /**
     * 工具方法，发送一个 HTTP POST 请求，并尝试将响应转换为 JSON
     *
     * @param url    请求的方法名 URL
     * @param params 参数表
     * @return 如果请求成功则返回 JSON，否则抛异常或者返回空
     * @throws IOException
     */
    public static JSONObject httpsPost(String url, Map<String, String> params) throws IOException,
UnrecoverableKeyException, NoSuchAlgorithmException, KeyStoreException, KeyManagementException {
        JSONObject jsonResult = null;
        HttpClient client = getHttpsClient();
        //发送 get 请求
```

```
        HttpPost request = new HttpPost(url);
        List<NameValuePair> urlParameters = new ArrayList<NameValuePair>();
        for (Map.Entry<String, String> entry : params.entrySet()) {
            urlParameters.add(new BasicNameValuePair(entry.getKey(), entry.getValue()));
        }
        HttpEntity postParams = new UrlEncodedFormEntity(urlParameters);
        request.setEntity(postParams);
        HttpResponse response = client.execute(request);
        if (response.getStatusLine().getStatusCode() == HttpStatus.SC_OK) {
            String strResult = EntityUtils.toString(response.getEntity());
            jsonResult = JSON.parseObject(strResult);
        }
        return jsonResult;
    }
}
```

MQTT Token 客户端接口

本文介绍 MQTT 客户端上传 Token、监听 Token 失效信息、监听 Token 非法信息三个接口的使用，并给出相关示例代码以供参考。

客户端上传 Token 凭证

发送 Topic：$SYS/uploadToken

内容：JSONString

内容信息：

名称	类型	说明
Token	String	如果客户端选用 Token 模式，则需要上传 Token 字符串。
type	String	Token 类型，分为 W，R，RW 共三种，对应三种权限类型的 Token。一个客户端最多拥有这 3个 Token，设置错误的类型会导致权限校验错误。

返回值

普通的 PubAck 报文。客户端必须等到该响应才能进行下一步 Pub 或者 Sub 操作，否则服务端可能会鉴权失败导致连接断开。

客户端监听即将失效的 Token 信息

接收 Topic：$SYS/tokenExpireNotice

内容：JSONString

内容信息：

名称	类型	说明
expireTime	int	该 Token 即将于什么时候失效，一般提前5分钟通知，只通知一次。
type	String	Token 类型，分为 W，R，RW 共三种，对应客户端上传的三种权限类型的 Token。

响应：

客户端收到 Token 即将失效的消息后，需要尽快处理重新申请 Token 的动作，以免造成收发消息失败。

客户端监听 Token 非法的通知

接收 Topic：$SYS/tokenInvalidNotice

内容：JSONString

内容信息：

名称	类型	说明
code	int	Token 校验失败的类型。
type	String	Token 类型，分为 W，R，RW 共三种，对应客户端上传的三种权限类型的 Token。

响应：

服务端校验 Token 失效，会导致鉴权失败，服务端会主动断开链接。断开链接之前，服务端会给客户端推送失败的 code，客户端根据 code 即可判断原因。

type code	错误类型
1	伪造 Token，不可解析
2	Token 已经过期
3	Token 已经被吊销
4	资源和 Token 不匹配
5	权限类型和 Token 不匹配
8	签名不合法
-1	帐号权限不合法

注意：

- 服务端下推 Token 非法的通知后，会接着断开连接。正常情况下客户端应该能根据该通知判断是否需要申请 Token。
- 异常情况下，可能尚未收到通知，即已经断开连接。此时客户端需要判断原先的 Token 是否到期，如果确实接近到期时间，则先申请 Token 再重连。

示例程序

```
public static void main(String[] args) throws MqttException, InterruptedException {
        String broker = "tcp://XXXX:1883";
        String clientId = "GID_XXX@@@XXXX";
        final String topic = "XXXX/XXXXX";
        MemoryPersistence persistence = new MemoryPersistence();
        final MqttClient sampleClient = new MqttClient(broker, clientId, persistence);
        final MqttConnectOptions connOpts = new MqttConnectOptions();
        System.out.println("Connecting to broker: " + broker);
        connOpts.setServerURIs(new String[] {broker});
        connOpts.setCleanSession(true);
        connOpts.setKeepAliveInterval(90);
        sampleClient.setCallback(new MqttCallbackExtended() {
            @Override
            public void connectComplete(boolean reconnect, String serverURI) {
                //连接成功，需重新上传 Token
                JSONObject object = new JSONObject();
                object.put("token", "XXXX");//设置 Token 内容
                object.put("type", "RW");//设置 Token 类型，共有 RW，R，W 三种类型
                MqttMessage message = new MqttMessage(object.toJSONString().getBytes());
                message.setQos(1);
                try {
                    sampleClient.publish("$SYS/uploadToken", message);
                    sampleClient.subscribe(topic, 0);
                } catch (MqttException e) {
                    e.printStackTrace();
                }
            }

            @Override
            public void connectionLost(Throwable throwable) {
                System.out.println("mqtt connection lost");
                throwable.printStackTrace();
            }

            @Override
            public void messageArrived(String topic, MqttMessage mqttMessage) throws Exception {
                if (topic.equals("$SYS/tokenInvalidNotice")) {
                    //Token 非法的通知
                } else if (topic.equals("$SYS/tokenExpireNotice")) {
                    //Token 即将失效的通知
                }
            }

            @Override
            public void deliveryComplete(IMqttDeliveryToken iMqttDeliveryToken) {
```

```
            System.out.println("deliveryComplete:" + iMqttDeliveryToken.getMessageId());
        }
    });
    sampleClient.connect(connOpts);
    JSONObject object = new JSONObject();
    object.put("token", "XXXX");//设置 Token 内容
    object.put("type", "RW");//设置 Token 类型,共有 RW，R，W 三种类型
    MqttMessage message = new MqttMessage(object.toJSONString().getBytes());
    message.setQos(1);
    MqttTopic pubTopic = sampleClient.getTopic("$SYS/uploadToken");
    MqttDeliveryToken token = pubTopic.publish(message);
    token.waitForCompletion();//同步等待上传结果
    //Token 上传成功，在开始正常的业务消息收发
    sampleClient.subscribe(topic, 0);
    while (true) {
        sampleClient.publish(topic, message);
        Thread.sleep(5000);
    }
}
```

接入示例

Java 接入示例

MQTT 客户端收发 MQTT 消息

本文主要介绍如何使用 MQTT 客户端收发 MQTT 消息，并给出示例代码供前期开发测试参考，包括资源创建、环境准备、示例代码、注意事项等。

注意：

本文给出的实例均基于 Eclipse Paho Java SDK 实现，SDK 下载请参见 MQTT 接入准备。如使用其他第三方的客户端，请适当修改。

1. 资源创建

使用 MQ 提供的 MQTT 服务，首先需要核实应用中使用的 Topic 资源是否已经创建，如果没有，请先去控制台创建 Topic，Group ID 等资源。

创建资源时需要根据需求选择对应的 Region，例如 MQTT 需要使用华北2的接入点，那么 Topic 等资源就在华北2 创建，资源创建具体请参见**快速入门指导**中的**步骤二：创建资源**。

注意：MQTT 使用的多级子 Topic 不需要创建，代码里直接使用即可，没有限制。

2. 环境准备

使用 MQTT 协议来收发消息，需要根据应用平台选择合适的客户端。本示例运行在 Java 平台，使用 Eclipse Paho Java SDK 构建。首先引入 Maven 依赖，POM 文件配置如下：

```
<dependencies>
<dependency>
<groupId>org.eclipse.paho</groupId>
<artifactId>org.eclipse.paho.client.mqttv3</artifactId>
<version>1.2.0</version>
</dependency>
<dependency>
<groupId>commons-codec</groupId>
<artifactId>commons-codec</artifactId>
<version>1.10</version>
</dependency>
</dependencies>
<repositories>
<repository>
<id>Eclipse Paho Repo</id>
<url>https://repo.eclipse.org/content/repositories/paho-releases/</url>
</repository>
<repository>
<id>snapshots-repo</id>
<url>https://oss.sonatype.org/content/repositories/snapshots</url>
<releases>
<enabled>false</enabled>
</releases>
<snapshots>
<enabled>true</enabled>
</snapshots>
</repository>
</repositories>
```

3. MQTT 发送消息

本段示例代码演示如何使用 MQTT 客户端发送普通消息和 P2P 的点对点消息，其中用到的工具 MacSignature 参考下文。

```
public class MQTTSendMsg {
public static void main(String[] args) throws IOException {
/**
* 设置当前用户私有的 MQTT 的接入点。例如此处示意使用 XXX，实际使用请替换用户自己的接入点。接入点的获取方法是
，在控制台创建 MQTT 实例，每个实例都会分配一个接入点域名。
*/
final String broker ="tcp://XXXX.mqtt.aliyuncs.com:1883";
/**
```

```
* 设置阿里云的 AccessKey，用于鉴权
*/
final String acessKey ="XXXXXX";
/**
* 设置阿里云的 SecretKey，用于鉴权
*/
final String secretKey ="XXXXXXX";
/**
* 发消息使用的一级 Topic，需要先在 MQ 控制台里创建
*/
final String topic ="XXXX";

/**
* MQTT 的 ClientID，一般由两部分组成，GroupID@@@DeviceID
* 其中 GroupID 在 MQ 控制台里创建
* DeviceID 由应用方设置，可能是设备编号等，需要唯一，否则服务端拒绝重复的 ClientID 连接
*/
final String clientId ="GID_XXX@@@ClientID_XXXX";
String sign;
MemoryPersistence persistence = new MemoryPersistence();
try {
final MqttClient sampleClient = new MqttClient(broker, clientId, persistence);
final MqttConnectOptions connOpts = new MqttConnectOptions();
System.out.println("Connecting to broker: " + broker);
/**
* 计算签名，将签名作为 MQTT 的 password。
* 签名的计算方法，参考工具类 MacSignature，第一个参数是 ClientID 的前半部分，即 GroupID
* 第二个参数阿里云的 SecretKey
*/
sign = MacSignature.macSignature(clientId.split("@@@")[0], secretKey);
connOpts.setUserName(acessKey);
connOpts.setServerURIs(new String[] { broker });
connOpts.setPassword(sign.toCharArray());
connOpts.setCleanSession(true);
connOpts.setKeepAliveInterval(90);
connOpts.setAutomaticReconnect(true);
connOpts.setMqttVersion(MQTT_VERSION_3_1_1);
sampleClient.setCallback(new MqttCallbackExtended() {
public void connectComplete(boolean reconnect, String serverURI) {
System.out.println("connect success");
//连接成功，需要上传客户端所有的订阅关系
}
public void connectionLost(Throwable throwable) {
System.out.println("mqtt connection lost");
}
public void messageArrived(String topic, MqttMessage mqttMessage) throws Exception {
System.out.println("messageArrived:" + topic + "------" + new String(mqttMessage.getPayload()));
}
public void deliveryComplete(IMqttDeliveryToken iMqttDeliveryToken) {
System.out.println("deliveryComplete:" + iMqttDeliveryToken.getMessageId());
}
});
sampleClient.connect(connOpts);
for (int i = 0; i < 10; i++) {
try {
String scontent = new Date()+"MQTT Test body" + i;
```

```
//此处消息体只需要传入 byte 数组即可，对于其他类型的消息，请自行完成二进制数据的转换
final MqttMessage message = new MqttMessage(scontent.getBytes());
message.setQos(0);
System.out.println(i+" pushed at "+new Date()+" "+ scontent);
/**
*消息发送到某个主题 Topic，所有订阅这个 Topic 的设备都能收到这个消息。
* 遵循 MQTT 的发布订阅规范，Topic 也可以是多级 Topic。此处设置了发送到二级 Topic
*/
sampleClient.publish(topic+"/notice/", message);
/**
* 如果发送 P2P 消息，二级 Topic 必须是“p2p” ，三级 Topic 是目标的 ClientID
* 此处设置的三级 Topic 需要是接收方的 ClientID
*/
String p2pTopic =topic+"/p2p/GID_mqttdelay3@@@DEVICEID_001";
sampleClient.publish(p2pTopic,message);
} catch (Exception e) {
e.printStackTrace();
}
}
} catch (Exception me) {
me.printStackTrace();
}
}
}
```

4. MQTT 接收消息

本段代码演示如何使用 MQTT 客户端订阅消息，接收普通的消息以及点对点消息。

```
public class MQTTRecvMsg {
public static void main(String[] args) throws IOException {
/**
* 设置当前用户私有的 MQTT 的接入点。例如此处示意使用 XXX，实际使用请替换用户自己的接入点。接入点的获取方法是
，在控制台创建 MQTT 实例，每个实例都会分配一个接入点域名。
*/
final String broker ="tcp://XXXX.mqtt.aliyuncs.com:1883";
/**
* 设置阿里云的 AccessKey，用于鉴权
*/
final String acessKey ="XXXXXX";
/**
* 设置阿里云的 SecretKey，用于鉴权
*/
final String secretKey ="XXXXXXX";
/**
* 发消息使用的一级 Topic，需要先在 MQ 控制台里创建
*/
final String topic ="XXXX";

/**
* MQTT 的 ClientID，一般由两部分组成，GroupID@@@DeviceID
* 其中 GroupID 在 MQ 控制台里创建
* DeviceID 由应用方设置，可能是设备编号等，需要唯一，否则服务端拒绝重复的 ClientID 连接
*/
```

```
final String clientId ="GID_XXXX@@@ClientID_XXXXXX";
String sign;
MemoryPersistence persistence = new MemoryPersistence();
try {
final MqttClient sampleClient = new MqttClient(broker, clientId, persistence);
final MqttConnectOptions connOpts = new MqttConnectOptions();
System.out.println("Connecting to broker: " + broker);
/**
* 计算签名，将签名作为 MQTT 的 password
* 签名的计算方法，参考工具类 MacSignature，第一个参数是 ClientID 的前半部分，即 GroupID
* 第二个参数阿里云的 SecretKey
*/
sign = MacSignature.macSignature(clientId.split("@@@")[0], secretKey);
/**
* 设置订阅方订阅的 Topic 集合，此处遵循 MQTT 的订阅规则，可以是一级 Topic，二级 Topic，P2P 消息请订阅/p2p
*/
final String[] topicFilters=new String[]{topic+"/notice/",topic+"/p2p"};
final int[]qos={0,0};
connOpts.setUserName(acessKey);
connOpts.setServerURIs(new String[] { broker });
connOpts.setPassword(sign.toCharArray());
connOpts.setCleanSession(true);
connOpts.setKeepAliveInterval(90);
connOpts.setAutomaticReconnect(true);
sampleClient.setCallback(new MqttCallbackExtended() {
public void connectComplete(boolean reconnect, String serverURI) {
System.out.println("connect success");
//连接成功，需要上传客户端所有的订阅关系
sampleClient.subscribe(topicFilters,qos);
}
public void connectionLost(Throwable throwable) {
System.out.println("mqtt connection lost");
}
public void messageArrived(String topic, MqttMessage mqttMessage) throws Exception {
System.out.println("messageArrived:" + topic + "------" + new String(mqttMessage.getPayload()));
}
public void deliveryComplete(IMqttDeliveryToken iMqttDeliveryToken) {
System.out.println("deliveryComplete:" + iMqttDeliveryToken.getMessageId());
}
});
//客户端每次上线都必须上传自己所有涉及的订阅关系，否则可能会导致消息接收延迟
sampleClient.connect(connOpts);
//每个客户端最多允许存在30个订阅关系，超出限制可能会丢弃导致收不到部分消息
sampleClient.subscribe(topicFilters,qos);
Thread.sleep(Integer.MAX_VALUE);
} catch (Exception me) {
me.printStackTrace();
}
}
}
```

上文代码用到的工具类 MacSignature.java 如下：

```
public class MacSignature {
/**
```

```
* @param text 要签名的文本
* @param secretKey 阿里云 MQ SecretKey
* @return 加密后的字符串
* @throws InvalidKeyException
* @throws NoSuchAlgorithmException
*/
public static String macSignature(String text, String secretKey) throws InvalidKeyException,
NoSuchAlgorithmException {
Charset charset = Charset.forName("UTF-8");
String algorithm = "HmacSHA1";
Mac mac = Mac.getInstance(algorithm);
mac.init(new SecretKeySpec(secretKey.getBytes(charset), algorithm));
byte[] bytes = mac.doFinal(text.getBytes(charset));
return new String(Base64.encodeBase64(bytes), charset);
}
/**
* 发送方签名方法
*
* @param clientId MQTT ClientID
* @param secretKey 阿里云 MQ SecretKey
* @return 加密后的字符串
* @throws NoSuchAlgorithmException
* @throws InvalidKeyException
*/
public static String publishSignature(String clientId, String secretKey) throws NoSuchAlgorithmException,
InvalidKeyException {
return macSignature(clientId, secretKey);
}
/**
* 订阅方签名方法
*
* @param topics 要订阅的 Topic 集合
* @param clientId MQTT ClientID
* @param secretKey 阿里云 MQ SecretKey
* @return 加密后的字符串
* @throws NoSuchAlgorithmException
* @throws InvalidKeyException
*/
public static String subSignature(List<String> topics, String clientId, String secretKey) throws
NoSuchAlgorithmException, InvalidKeyException {
Collections.sort(topics); //以字典顺序排序
String topicText = "";
for (String topic : topics) {
topicText += topic + "\n";
}
String text = topicText + clientId;
return macSignature(text, secretKey);
}
/**
* 订阅方签名方法
*
* @param topic 要订阅的 Topic
* @param clientId MQTT ClientID
* @param secretKey 阿里云 MQ SecretKey
* @return 加密后的字符串
* @throws NoSuchAlgorithmException
```

```
* @throws InvalidKeyException
*/
public static String subSignature(String topic, String clientId, String secretKey) throws NoSuchAlgorithmException,
InvalidKeyException {
List<String> topics = new ArrayList<String>();
topics.add(topic);
return subSignature(topics, clientId, secretKey);
}
}
```

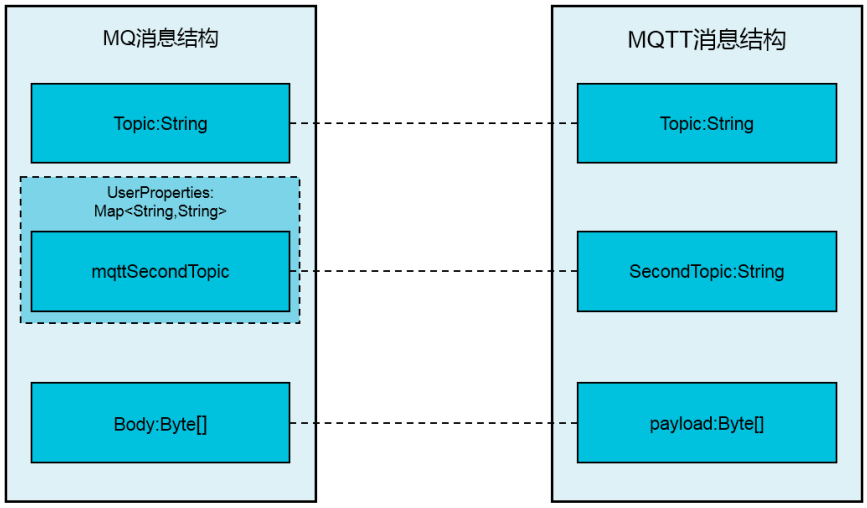
MQ 客户端收发 MQTT 消息

除了使用 MQTT 客户端收发 MQTT 消息，MQ 也支持使用 MQ 客户端来收发 MQTT 消息，实现 MQ 私有协议和 MQTT 协议之间的数据互通。

MQTT 是面向移动端的消息协议，一般单个客户端的处理能力都比较弱。因此 MQTT 协议适用于大量在线客户端，但是每个客户端消息很少的场景。而 MQ 是面向服务端的消息引擎，单个客户端处理能力强，TPS 高，适用于服务端进行大批量处理分析的场景。因此，一般推荐在移动端设备上使用 MQTT，而在服务端应用则使用 MQ 。

例如，业务上有千万数量级的传感器在使用 MQTT 客户端上传数据，此时服务端就可以用 MQ 客户端搭建，用少量的机器来完成这些传感器数据的处理。

MQ 消息和 MQTT 消息的对应关系如下图所示：



使用 MQ 客户端的相关说明，请参见 MQ TCP Java SDK 接入。

1. MQ 客户端发送 MQTT 消息

本小节介绍如何使用 MQ 的 SDK 向 MQTT 设备发送消息。

使用 MQ 客户端向 MQTT 设备发消息的方式和使用 MQ 客户端向 MQ 设备发消息的方式没有区别，只是要根据需求，将 MQTT 的二级 Topic 设置到 MQ 的消息属性中，同时指定消息的 Tag 为 MQ2MQTT 即可。

其中，涉及到 MQTT 的相关特性需要设置到消息的属性中，具体列表如下：

属性	key	value	默认值	说明
QoS	qoslevel	0,1,2	1	该消息的 QoS 级别
CleanSession	cleansessionflag	true/false	true	该消息的 cleanSession 标签，仅 P2P 消息支持设置
subTopic	mqttSecondTopic	字符串	空	该消息的子 topic，如果不设默认为空
mqttRealTopic	mqttRealTopic	字符串	无	客户端收到消息时显示的topic名称，该属性一般用于P2P消息，即客户端收到P2P消息时可以显示成预期的业务topic

示例代码：

```
public class ONSSendMsg {
public static void main(String[] args) throws InterruptedException {
/**
* 设置阿里云的 AccessKey，用于鉴权
*/
final String acessKey ="XXXXXX";
/**
* 设置阿里云的 SecretKey，用于鉴权
*/
final String secretKey ="XXXXXXX";
/**
* 发消息使用的一级 Topic，需要先在 MQ 控制台里创建
*/
final String topic ="MQTTTestTopic";
/**
* ProducerID，需要先在 MQ 控制台里创建
*/
final String producerId ="PID_MQTTTestTopic";
Properties properties =new Properties();
properties.put(PropertyKeyConst.ProducerId, producerId);
properties.put(PropertyKeyConst.AccessKey, acessKey);
properties.put(PropertyKeyConst.SecretKey, secretKey);
Producer producer = ONSFactory.createProducer(properties);
producer.start();
byte[] body=new byte[1024];
final Message msg = new Message(
```

```
topic,//MQ 消息的 Topic，需要事先创建
"MQ2MQTT",//MQ Tag，通过 MQ 向 MQTT 客户端发消息时，必须指定 MQ2MQTT 作为 Tag，其他 Tag 或者不设都将导致 MQTT 客户端收不到消息
body);//消息体，和 MQTT 的 body 对应
/**
* 使用 MQ 客户端给 MQTT 设备发送 P2P 消息时，需要在 MQ 消息中设置 mqttSecondTopic 属性
* 设置的值是“/p2p/”+目标 ClientID
*/
String targetClientID="GID_MQTTTestTopic@@@DeviceID_0001";
msg.putUserProperties("mqttSecondTopic", "/p2p/"+targetClientID);
//发送消息，只要不抛异常就是成功。
SendResult sendResult = producer.send(msg);
System.out.println(sendResult);
/**
* 如果仅仅发送 Pub/Sub 消息，则只需要设置实际 MQTT 订阅的 Topic 即可，支持设置二级 Topic
*/
msg.putUserProperties("mqttSecondTopic", "/notice/");
SendResult result =producer.send(msg);
producer.shutdown();
System.exit(0);
}
}
```

2. MQ 客户端接收 MQTT 消息

本小节介绍如何使用 MQ 的 SDK 接收来自 MQTT 设备发送的消息。使用 MQ 客户端接收 MQTT 设备的消息和普通 MQ 客户端收消息没有区别，MQ 客户端只需要订阅 MQTT 的一级 Topic 即可。

```
public class ONSRecvMsg {
public static void main(String[] args) throws InterruptedException {
/**
* 设置阿里云的 AccessKey，用于鉴权
*/
final String acessKey ="XXXXXX";
/**
* 设置阿里云的 SecretKey，用于鉴权
*/
final String secretKey ="XXXXXXX";
/**
* 收消息使用的一级 Topic，需要先在 MQ 控制台里创建
*/
final String topic ="MQTTTestTopic";
/**
* ConsumerID ,需要先在 MQ 控制台里创建
*/
final String consumerID ="CID_MQTTTestTopic";
Properties properties =new Properties();
properties.put(PropertyKeyConst.ConsumerId, consumerID);
properties.put(PropertyKeyConst.AccessKey, acessKey);
properties.put(PropertyKeyConst.SecretKey, secretKey);
Consumer consumer =ONSFactory.createConsumer(properties);
/**
* 此处 MQ 客户端只需要订阅 MQTT 的一级 Topic 即可
*/
```

```
consumer.subscribe(topic, "*", new MessageListener() {
public Action consume(Message message, ConsumeContext consumeContext) {
System.out.println("recv msg:"+message);
return Action.CommitMessage;
}
});
consumer.start();
System.out.println("[Case Normal Consumer Init] Ok");
Thread.sleep(Integer.MAX_VALUE);
consumer.shutdown();
System.exit(0);
}
```

MQTT客户端发送顺序消息

本文主要介绍如何使用 MQTT 客户端发送顺序消息。

注意：

本文给出的实例均基于 Eclipse Paho Java SDK 实现，其他语言和平台的实现方法类似，请自行对照修改。

1. Topic 创建

MQTT 协议的消息目前支持仅支持顺序发送，即可以保证每个使用 MQTT 协议的客户端的消息发送是顺序的，但并不保证消息接收是顺序的。举例而言，物联网领域里只需要保证每个传感器采集的数据是顺序上传即可。消息的处理方，即服务端应用可以使用 MQ 协议的 SDK 来做顺序消费。

使用 MQTT 发顺序消息，首先需要保证 topic 本身是支持顺序的，即 topic 的创建必须选择分区顺序或者全局顺序类型。更多信息请参考《接入准备》一章。

注意：使用普通 topic 发送，即使程序用法正确，也无法保证顺序性。其次，MQTT 服务仅保证消息的发送是顺序的，如果需要顺序消费，请使用 MQ 协议的顺序客户端消费。

2. 顺序发送示例

使用 MQTT 协议来发送顺序消息，需要在建立连接之后，首先发送控制报文标志客户端即将使用顺序模式发消息，然后再进行正常的业务发送，控制报文每次建立连接时都必须且仅仅需要设置一次。具体参考以下 Demo 程序。

```
public class MQTTSendMsg {
public static void main(String[] args) throws IOException {
/**
* 设置当前用户私有的 MQTT 的接入点。例如此处示意使用 XXX，实际使用请替换用户自己的接入点。接入点的获取方法是，在控制台创建 MQTT 实例，每个实例都会分配一个接入点域名。
*/
final String broker ="tcp://XXXX.mqtt.aliyuncs.com:1883";
```

```
/**
* 设置阿里云的 AccessKey，用于鉴权
*/
final String acessKey ="XXXXXX";
/**
* 设置阿里云的 SecretKey，用于鉴权
*/
final String secretKey ="XXXXXXX";
/**
* 发消息使用的一级 Topic，需要先在 MQ 控制台里创建
*/
final String topic ="XXXX";

/**
* MQTT 的 ClientID，一般由两部分组成，GroupID@@@DeviceID
* 其中 GroupID 在 MQ 控制台里创建
* DeviceID 由应用方设置，可能是设备编号等，需要唯一，否则服务端拒绝重复的 ClientID 连接
*/
final String clientId ="GID_XXX@@@ClientID_XXXX";
String sign;
MemoryPersistence persistence = new MemoryPersistence();
try {
final MqttClient sampleClient = new MqttClient(broker, clientId, persistence);
final MqttConnectOptions connOpts = new MqttConnectOptions();
System.out.println("Connecting to broker: " + broker);
/**
* 计算签名，将签名作为 MQTT 的 password。
* 签名的计算方法，参考工具类 MacSignature，第一个参数是 ClientID 的前半部分，即 GroupID
* 第二个参数阿里云的 SecretKey
*/
sign = MacSignature.macSignature(clientId.split("@@@")[0], secretKey);
connOpts.setUserName(acessKey);
connOpts.setServerURIs(new String[] { broker });
connOpts.setPassword(sign.toCharArray());
connOpts.setCleanSession(true);
connOpts.setKeepAliveInterval(90);
connOpts.setAutomaticReconnect(true);
connOpts.setMqttVersion(MQTT_VERSION_3_1_1);
sampleClient.setCallback(new MqttCallbackExtended() {
public void connectComplete(boolean reconnect, String serverURI) {
System.out.println("connect success");
//连接成功，需要上传客户端的顺序控制报文
JSONObject object = new JSONObject();
object.put("order", "true");//设置顺序发送的标记
MqttMessage message = new MqttMessage(object.toJSONString().getBytes());
message.setQos(1);
sampleClient.publish("$SYS/enableOrderMsg", message);
}
public void connectionLost(Throwable throwable) {
System.out.println("mqtt connection lost");
}
public void messageArrived(String topic, MqttMessage mqttMessage) throws Exception {
System.out.println("messageArrived:" + topic + "------" + new String(mqttMessage.getPayload()));
}
public void deliveryComplete(IMqttDeliveryToken iMqttDeliveryToken) {
System.out.println("deliveryComplete:" + iMqttDeliveryToken.getMessageId());
```

```
}
});
sampleClient.connect(connOpts);
for (int i = 0; i < 10; i++) {
try {
String scontent = new Date()+"MQTT Test body" + i;
//此处消息体只需要传入 byte 数组即可，对于其他类型的消息，请自行完成二进制数据的转换。
final MqttMessage message = new MqttMessage(scontent.getBytes());
message.setQos(0);
System.out.println(i+" pushed at "+new Date()+" "+ scontent);
/**
*消息发送到某个主题 Topic，所有订阅这个 Topic 的设备都能收到这个消息。
* 遵循 MQTT 的发布订阅规范，Topic 也可以是多级 Topic。此处设置了发送到二级 Topic。
*/
sampleClient.publish(topic+"/notice/", message);
/**
* 如果发送 P2P 消息，二级 Topic 必须是“p2p” ，三级 Topic 是目标的 ClientID。
* 此处设置的三级 Topic 需要是接收方的 ClientID。
*/
String p2pTopic =topic+"/p2p/GID_mqttdelay3@@@DEVICEID_001";
sampleClient.publish(p2pTopic,message);
} catch (Exception e) {
e.printStackTrace();
}
}
} catch (Exception me) {
me.printStackTrace();
}
}
}
```

SSL 方式接入示例

本文主要介绍如何使用 MQTT 的 SSL 加密通道来收发消息。

注意：

SSL 方式接入需要服务端支持，目前各个 Region 对 SSL 通道的支持情况请参考环境准备章节的文档。

本文给出的示例均基于 Eclipse Paho Java SDK 实现，SDK 下载请参见 MQTT 接入准备。如使用其他第三方的客户端，请适当修改。

客户端设置 SSL 属性

使用加密端口连接服务，需要注意应用的代码中服务端的端口和协议类型和默认的 MQTT 协议不一样。具体示例如下：

```
public static void main(String[] args) throws IOException {
/**
* 设置当前用户私有的 MQTT 的接入点。例如此处示意使用 XXX，实际使用请替换用户自己的接入点。接入点的获取方法是
，在控制台创建 MQTT 实例，每个实例都会分配一个接入点域名。
*/
final String broker ="ssl://XXXX.mqtt.aliyuncs.com:8883";
/**
* 设置阿里云的 AccessKey，用于鉴权
*/
final String acessKey ="XXXXXX";
/**
* 设置阿里云的 SecretKey，用于鉴权
*/
final String secretKey ="XXXXXXX";
/**
* 发消息使用的一级 Topic，需要先在 MQ 控制台里创建
*/
final String topic ="XXXX";

/**
* MQTT 的 ClientID，一般由两部分组成，GroupID@@@DeviceID
* 其中 GroupID 在 MQ 控制台里创建
* DeviceID 由应用方设置，可能是设备编号等，需要唯一，否则服务端拒绝重复的 ClientID 连接
*/
final String clientId ="GID_XXX@@@ClientID_XXXX";
String sign;
MemoryPersistence persistence = new MemoryPersistence();
try {
final MqttClient sampleClient = new MqttClient(broker, clientId, persistence);
final MqttConnectOptions connOpts = new MqttConnectOptions();
System.out.println("Connecting to broker: " + broker);
/**
* 计算签名，将签名作为 MQTT 的 password。
* 签名的计算方法，参考工具类 MacSignature，第一个参数是 ClientID 的前半部分，即 GroupID
* 第二个参数阿里云的 SecretKey
*/
sign = MacSignature.macSignature(clientId.split("@@@")[0], secretKey);
connOpts.setUserName(acessKey);
connOpts.setServerURIs(new String[] { broker });
connOpts.setPassword(sign.toCharArray());
connOpts.setCleanSession(false);
connOpts.setKeepAliveInterval(100);
connOpts.setAutomaticReconnect(true);
connOpts.setMqttVersion(MQTT_VERSION_3_1_1);
sampleClient.setCallback(new MqttCallback() {
public void connectComplete(boolean reconnect, String serverURI) {
System.out.println("connect success");
}
public void connectionLost(Throwable throwable) {
System.out.println("mqtt connection lost");
throwable.printStackTrace();
}
public void messageArrived(String topic, MqttMessage mqttMessage) throws Exception {
System.out.println("messageArrived:" + topic + "------" + new String(mqttMessage.getPayload()));
}
public void deliveryComplete(IMqttDeliveryToken iMqttDeliveryToken) {
```

```
System.out.println("deliveryComplete:" + iMqttDeliveryToken.getMessageId());
}
});
sampleClient.connect(connOpts);
for (int i = 0; i < 10; i++) {
try {
String scontent = new Date()+"MQTT Test body" + i;
final MqttMessage message = new MqttMessage(scontent.getBytes());
message.setQos(0);
System.out.println(i+" pushed at "+new Date()+" "+ scontent);
/**
*消息发送到某个主题 Topic，所有订阅这个 Topic 的设备都能收到这个消息
* 遵循 MQTT 的发布订阅规范，Topic 也可以是多级 Topic。此处设置了发送到二级 Topic
*/
sampleClient.publish(topic+"/notice/", message);
/**
* 如果发送 P2P 消息，二级 Topic 必须是“p2p” ，三级 Topic 是目标的 ClientID
* 此处设置的三级 Topic 需要是接收方的 ClientID
*/
String p2pTopic =topic+"/p2p/GID_mqttdelay3@@@DEVICEID_001";
sampleClient.publish(p2pTopic,message);
} catch (Exception e) {
e.printStackTrace();
}
}
} catch (Exception me) {
me.printStackTrace();
}
}
```

注意：使用 SSL 接入时，服务端的地址协议请设置为 ssl：//XXX，不要使用 tcp：//XXX，其他部分的设置和非加密通道一致即可。

Android 接入示例

本文主要介绍如何使用 Android 客户端收发 MQTT 消息。因为 Android 和 Java 语法上互通，所以本文仅给出 Android 开发环境的配置，收发程序代码请参考 Java 语言示例。

1. Android 依赖添加

本文给出的 Android 示例工程使用 Gradle 包管理器管理依赖。在 Android 工程中找到 app 下的配置文件 build.gradle。添加依赖如下。

```
dependencies {
compile 'org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.1.0'
compile group: 'commons-codec', name: 'commons-codec', version: '1.5'
}
```

2. 收发消息 Demo 工程

由于 Android 和 Java 语法互通，所以收发消息的代码可以复用 Java 的代码，具体请参考Android Demo 工程。

C 接入示例

本文主要介绍如何使用 C 客户端收发 MQTT 消息，并给出示例代码供前期开发测试参考。

注意：

本文给出的实例均基于 Eclipse Paho C SDK 实现，该 SDK 下载请参见 MQTT 接入准备。如使用其他第三方的客户端，请适当修改示例。

1. 资源创建

使用 MQ 提供的 MQTT 服务，首先需要核实应用中使用的 Topic 资源是否已经创建，如果没有请先去控制台创建 Topic，Group ID 等资源。

创建资源时需要根据需求选择对应的 Region，例如，MQTT 需要使用华北 2 的接入点，那么 Topic 等资源就在华北 2 创建，资源创建具体请参见快速入门指导中的**步骤二：创建资源**。

注意：MQTT 使用的多级子 Topic 不需要创建，代码里直接使用即可，没有限制。

2. C 客户端依赖

Paho MQTT C 客户端 SDK 从官网下载源码后，参考文档进行编译安装。可能需要安装以下开发库和工具。

```
openssl：用于签名计算。
cmake：用于编译示例程序，实际也可以根据习惯选用其他编译工具。
```

3. MQTT 收发消息

本段示例代码演示如何使用 C 客户端收发 MQTT 消息，其中编译工具使用 CMake。

CMakeLists.txt 文件

```
cmake_minimum_required(VERSION 3.7)
project(mqttdemo)

INCLUDE_DIRECTORIES(
.
${CMAKE_SOURCE_DIR}/src
${CMAKE_BINARY_DIR}
```

```
)
SET(CMAKE_BUILD_TYPE "Debug")
SET(CMAKE_CXX_FLAGS_DEBUG "$ENV{CXXFLAGS} -O0 -Wall -g2 -ggdb")
SET(CMAKE_CXX_FLAGS_RELEASE "$ENV{CXXFLAGS} -O3 -Wall")

set(CMAKE_C_STANDARD 99)
add_executable(mqttDemo mqttDemo.c)
TARGET_LINK_LIBRARIES(mqttDemo crypto)
TARGET_LINK_LIBRARIES(mqttDemo paho-mqtt3as)
```

收发消息程序：

```
#include "MQTTAsync.h"

#include <signal.h>
#include <memory.h>
#include <stdlib.h>
#include <unistd.h>
#include <openssl/hmac.h>
#include <openssl/bio.h>

volatile int connected = 0;
char *topic;
char *userName;
char *passWord;

int messageDeliveryComplete(void *context, MQTTAsync_token token) {
/* not expecting any messages */
printf("send message %d success\n", token);
return 1;
}

int messageArrived(void *context, char *topicName, int topicLen, MQTTAsync_message *m) {
/* not expecting any messages */
printf("recv message from %s ,body is %s\n", topicName, (char *) m->payload);
MQTTAsync_freeMessage(&m);
MQTTAsync_free(topicName);
return 1;
}

void onConnectFailure(void *context, MQTTAsync_failureData *response) {
connected = 0;
printf("connect failed, rc %d\n", response ? response->code : -1);
MQTTAsync client = (MQTTAsync) context;
}

void onSubcribe(void *context, MQTTAsync_successData *response) {
printf("subscribe success \n");
}

void onConnect(void *context, MQTTAsync_successData *response) {
connected = 1;
printf("connect success \n");
MQTTAsync client = (MQTTAsync) context;
//do sub when connect success
```

```
MQTTAsync_responseOptions sub_opts = MQTTAsync_responseOptions_initializer;
sub_opts.onSuccess = onSubcribe;
int rc = 0;
if ((rc = MQTTAsync_subscribe(client, topic, 1, &sub_opts)) != MQTTASYNC_SUCCESS) {
printf("Failed to subscribe, return code %d\n", rc);
}
}

void onDisconnect(void *context, MQTTAsync_successData *response) {
connected = 0;
printf("connect lost \n");
}

void onPublishFailure(void *context, MQTTAsync_failureData *response) {
printf("Publish failed, rc %d\n", response ? -1 : response->code);
}

int success = 0;

void onPublish(void *context, MQTTAsync_successData *response) {
printf("send success %d\n", ++success);
}


void connectionLost(void *context, char *cause) {
connected = 0;
MQTTAsync client = (MQTTAsync) context;
MQTTAsync_connectOptions conn_opts = MQTTAsync_connectOptions_initializer;
int rc = 0;

printf("Connecting\n");
conn_opts.MQTTVersion = MQTTVERSION_3_1_1;
conn_opts.keepAliveInterval = 60;
conn_opts.cleansession = 1;
conn_opts.username = userName;
conn_opts.password = passWord;
conn_opts.onSuccess = onConnect;
conn_opts.onFailure = onConnectFailure;
conn_opts.context = client;
//如果使用加密ssl的方式，此处需要初始化，否则设置为NULL
conn_opts.ssl = NULL;
//MQTTAsync_SSLOptions ssl =MQTTAsync_SSLOptions_initializer;
//conn_opts.ssl = &ssl;
if ((rc = MQTTAsync_connect(client, &conn_opts)) != MQTTASYNC_SUCCESS) {
printf("Failed to start connect, return code %d\n", rc);
exit(EXIT_FAILURE);
}
}


int main(int argc, char **argv) {
MQTTAsync_disconnectOptions disc_opts = MQTTAsync_disconnectOptions_initializer;
MQTTAsync client;
//MQTT 使用的 Topic，其中第一级父 Topic 需要在 MQ 控制台先创建
topic = "XXXXX";
//MQTT 的接入域名，在 MQ 控制台购买付费实例后即分配该域名,如果是使用加密ssl的方式，域名格式是
```

```
ssl://XXX.mqtt.aliyuncs.com
char *host = "XXX.mqtt.aliyuncs.com";
//MQTT 客户端分组 Id，需要先在 MQ 控制台创建
char *groupId = "GID_XXXXXX";
//MQTT 客户端设备 Id，用户自行生成，需要保证对于所有 TCP 连接全局唯一
char *deviceId = "XXXXXXX";
//帐号 AK
char *accessKey = "XXXXXXX";
//帐号 SK
char *secretKey = "XXXXXXX";
//服务端口，使用 MQTT 协议时设置1883,其他协议参考文档选择合适端口,如果是加密ssl的话端口是8883
int port = 1883;
//QoS，消息传输级别，参考文档选择合适的值
int qos = 1;
//cleanSession，是否设置持久会话，如果需要离线消息则 cleanSession 必须是 false，QoS 必须是1
int cleanSession = 0;
int rc = 0;
char tempData[100];
int len = 0;
HMAC(EVP_sha1(), secretKey, strlen(secretKey), groupId, strlen(groupId), tempData, &len);
char resultData[100];
int passWordLen = EVP_EncodeBlock((unsigned char *) resultData, tempData, len);
resultData[passWordLen] = '\0';
printf("passWord is %s", resultData);
userName = accessKey;
passWord = resultData;
//1.create client
MQTTAsync_createOptions create_opts = MQTTAsync_createOptions_initializer;
create_opts.sendWhileDisconnected = 0;
create_opts.maxBufferedMessages = 10;
char url[100];
sprintf(url, "%s:%d", host, port);
char clientIdUrl[64];
sprintf(clientIdUrl, "%s@@@%s", groupId, deviceId);
rc = MQTTAsync_createWithOptions(&client, url, clientIdUrl, MQTTCLIENT_PERSISTENCE_NONE, NULL,
&create_opts);
rc = MQTTAsync_setCallbacks(client, client, connectionLost, messageArrived, NULL);
//2.connect to server
MQTTAsync_connectOptions conn_opts = MQTTAsync_connectOptions_initializer;
conn_opts.MQTTVersion = MQTTVERSION_3_1_1;
conn_opts.keepAliveInterval = 60;
conn_opts.cleansession = cleanSession;
conn_opts.username = userName;
conn_opts.password = passWord;
conn_opts.onSuccess = onConnect;
conn_opts.onFailure = onConnectFailure;
conn_opts.context = client;
//如果使用加密ssl的方式，此处需要初始化，否则设置为NULL
conn_opts.ssl = NULL;
//MQTTAsync_SSLOptions ssl =MQTTAsync_SSLOptions_initializer;
//conn_opts.ssl = &ssl;
conn_opts.automaticReconnect = 1;
conn_opts.connectTimeout = 3;
if ((rc = MQTTAsync_connect(client, &conn_opts)) != MQTTASYNC_SUCCESS) {
printf("Failed to start connect, return code %d\n", rc);
exit(EXIT_FAILURE);
```

```
}
//3.publish msg
MQTTAsync_responseOptions pub_opts = MQTTAsync_responseOptions_initializer;
pub_opts.onSuccess = onPublish;
pub_opts.onFailure = onPublishFailure;
for (int i = 0; i < 1000; i++) {
do {
char data[100];
sprintf(data, "hello mqtt demo");
rc = MQTTAsync_send(client, topic, strlen(data), data, qos, 0, &pub_opts);
sleep(1);
} while (rc != MQTTASYNC_SUCCESS);
}
sleep(1000);
disc_opts.onSuccess = onDisconnect;
if ((rc = MQTTAsync_disconnect(client, &disc_opts)) != MQTTASYNC_SUCCESS) {
printf("Failed to start disconnect, return code %d\n", rc);
exit(EXIT_FAILURE);
}
while (connected)
sleep(1);
MQTTAsync_destroy(&client);
return EXIT_SUCCESS;
}
```

.NET 接入示例

本文主要介绍如何使用 .NET SDK 收发 MQTT 消息，并给出示例代码供前期开发测试参考。

注意：

本文给出的示例均基于 Eclipse Paho .NET SDK 实现，SDK 下载请参见 MQTT 接入准备。如使用其他第三方的客户端，请适当修改示例。

1. 资源创建

使用 MQ 提供的 MQTT 服务，首先需要核实应用中使用的 Topic 资源是否已经创建。如果没有，请先去控制台创建 Topic，Group ID 等资源。

创建资源时需要根据需求选择对应的 Region，例如 MQTT 需要使用华北2的接入点，那么 Topic 等资源就在华北2 创建。资源创建具体请参见快速入门指导中的**步骤二：创建资源**。

注意：MQTT 使用的多级子 Topic 不需要创建，代码里直接使用即可，没有限制。

2. 环境准备

.NET SDK 依赖的添加可以自行搜索。可以直接使用 NuGet 包管理器安装，或者自行根据源码编译，具体源码

编译请参考 paho 官方文档。本文档介绍直接使用包管理器安装项目依赖。

```
Install-Package M2Mqtt -Version 4.3.0
```

3. MQTT 收发消息

本段示例代码演示如何使用 MQTT .NET SDK 发送和接收消息。需要注意此文档仅仅给出建立连接，收发消息并关闭连接的一个简单过程。实际应用集成应处理客户端断线重连等情况，尽可能保证客户端连接稳定，不要频繁创建客户端重新连接。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;
using uPLibrary.Networking.M2Mqtt;
using uPLibrary.Networking.M2Mqtt.Messages;

namespace MQTTDemo
{
class MQTTDoNetDemo
{
static void Main(string[] args)
{
//此处填写购买得到的 MQTT 接入点域名
String brokerUrl = "XXX.mqtt.aliyuncs.com";
//此处填写阿里云帐号 AccessKey
String accessKey = "XXXXXX";
//此处填写阿里云帐号 SecretKey
String secretKey = "XXXXXX";
//此处填写在 MQ 控制台创建的 Topic，作为 MQTT 的一级 Topic
String parentTopic = "XXXXX";
//此处填写客户端 ClientId，需要保证全局唯一，其中前缀部分即 GroupId 需要先在 MQ 控制台创建
String clientId = "GID_XXXX@@@XXXX";
MqttClient client = new MqttClient(brokerUrl);
client.MqttMsgPublishReceived += client_recvMsg;
client.MqttMsgPublished += client_publishSuccess;
client.ConnectionClosed += client_connectLose;
String userName = accessKey;
//计算签名
String passWord = HMACSHA1(secretKey, clientId.Split('@')[0]);
client.Connect(clientId, userName,passWord,true,60);
//订阅 Topic，支持多个 Topic，以及多级 Topic
string[] subTopicArray = { parentTopic+"/subDemo1", parentTopic+"/subDemo2/level3" };
byte[] qosLevels = { MqttMsgBase.QOS_LEVEL_AT_MOST_ONCE, MqttMsgBase.QOS_LEVEL_AT_MOST_ONCE };
client.Subscribe(subTopicArray, qosLevels);
client.Publish(parentTopic + "/subDemo1", Encoding.UTF8.GetBytes("hello
mqtt"),MqttMsgBase.QOS_LEVEL_AT_LEAST_ONCE,false);
//发送 P2P 消息，二级 topic 必须是 p2p,三级 topic 是接收客户端的 clientId
client.Publish(parentTopic + "/p2p/"+clientId, Encoding.UTF8.GetBytes("hello mqtt"),
MqttMsgBase.QOS_LEVEL_AT_LEAST_ONCE, false);
```

```
System.Threading.Thread.Sleep(10000);
client.Disconnect();
}
static void client_recvMsg(object sender, MqttMsgPublishEventArgs e)
{
// access data bytes throug e.Message
Console.WriteLine("Recv Msg : Topic is "+e.Topic+" ,Body is "+Encoding.UTF8.GetString(e.Message));
}
static void client_publishSuccess(object sender, MqttMsgPublishedEventArgs e)
{
// access data bytes throug e.Message
Console.WriteLine("Publish Msg Success");
}
static void client_connectLose(object sender, EventArgs e)
{
// access data bytes throug e.Message
Console.WriteLine("Connect Lost,Try Reconnect");
}
public static string HMACSHA1(string key, string dataToSign)
{
Byte[] secretBytes = UTF8Encoding.UTF8.GetBytes(key);
HMACSHA1 hmac = new HMACSHA1(secretBytes);
Byte[] dataBytes = UTF8Encoding.UTF8.GetBytes(dataToSign);
Byte[] calcHash = hmac.ComputeHash(dataBytes);
String calcHashString = Convert.ToBase64String(calcHash);
return calcHashString;
}
}
}
```

MQTT iOS 接入示例

本文主要介绍如何使用 iOS 客户端收发 MQTT 消息，并给出示例代码供前期开发测试参考，包括资源创建、环境准备、示例代码、注意事项等。

注意：

本文给出的实例均基于第三方框架 MQTT-Client-Framework 实现，SDK 下载请参见 MQTT 接入准备。如使用其他第三方的客户端，请适当修改示例。

资源创建

使用 MQ 提供的 MQTT 服务，首先需要核实应用中使用的 Topic 资源是否已经创建。如果没有，请先去控制台创建 Topic，Group ID 等资源。

创建资源时需要根据需求选择对应的 Region，例如 MQTT 需要使用华北2的接入点，那么 Topic 等资源就在华北2 创建，资源创建具体请参见快速入门指导中的**步骤二：创建资源**。

注意：MQTT 使用的多级子 Topic 不需要创建，代码里直接使用即可，没有限制。

环境准备

iOS 环境下开发 MQTT 客户端程序，一般依赖稳定的第三方 FrameWork，由于涉及网络数据传输，建议选择 Object-c 原生的框架，比如 MQTT-Client-Framework 。

第三方 FrameWork 一般可以用 CocoaPods 来管理资源包依赖。具体配置流程如下：

CocoaPods 安装配置

CocoaPods 依赖 Ruby 等基础环境。请确保安装过 Ruby，然后将国内的镜像源更新为淘宝源，提高资源包下载速度。

```
//查看软件源
gem sources -l
//清理掉默认的软件源
gem sources --remove https://rubygems.org/
//加入淘宝的源
gem sources -a https://ruby.taobao.org/
```

更新完地址后，运行以下命令，等待一段时间后即可运行 pod 工具：

sudo gem install cocoapods

更新项目依赖

MQTT-Client-FrameWork 第三方框架支持 pod 管理方式，因此添加依赖只需要在项目根目录的 Podfile 中加入依赖即可，内容如下：

```
pod 'MQTTClient'
target '${yourprojectname}' do
end
```

其中 target 填入项目的名称，然后在 Podfile 目录下运行以下命令，更新依赖即可。

pod install

主要代码示例

初始化客户端

MQTT-Client-FrameWork 包提供的客户端类有 MQTTSession 和 MQTTSessionManager，建议使用后者维持静态资源，而且已经封装好自动重连等逻辑。初始化时需要传入相关的网络参数。具体如下：

```
        self.manager = [[MQTTSessionManager alloc] init];
self.manager.delegate = self;
self.manager.subscriptions = [NSDictionary dictionaryWithObject:[NSNumber numberWithInt:self.qos]
forKey:[NSString stringWithFormat:@"%@/#", self.rootTopic]];
//password的计算方式是，使用secretkey对groupId做hmac签名算法，具体实现参考macSignWithText方法
NSString *passWord = [[self class] macSignWithText:self.groupId secretKey:self.secretKey];
[self.manager connectTo:self.mqttSettings[@"host"]
port:[self.mqttSettings[@"port"] intValue]
tls:[self.mqttSettings[@"tls"] boolValue]
keepalive:60 //心跳间隔不得大于120s
clean:true
auth:true
user:self.accessKey
pass:passWord
will:false
willTopic:nil
willMsg:nil
willQos:0
willRetainFlag:FALSE
withClientId:self.clientId];
```

添加回调接口

针对连接当前状态，添加对应的回调接口，可以进行相关的业务逻辑处理。

```
[self.manager addObserver:self
forKeyPath:@"state"
options:NSKeyValueObservingOptionInitial | NSKeyValueObservingOptionNew
context:nil];


- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object change:(NSDictionary *)change
context:(void *)context {
switch (self.manager.state) {
case MQTTSessionManagerStateClosed:
break;
case MQTTSessionManagerStateClosing:
break;
case MQTTSessionManagerStateConnected:
break;
case MQTTSessionManagerStateConnecting:
break;
case MQTTSessionManagerStateError:
break;
case MQTTSessionManagerStateStarting:
default:
break;
}
}
```

userName 和 passWord 的设置

由于服务端需要对客户端进行鉴权，因此需要传入合法的 userName 和 passWord。userName 设置为当前用户的 AccessKey，password 则设置为 MQTT 客户端 GroupID 的签名字符串，签名计算方式是使用

SecretKey 对 GroupID 做 HmacSHA1 散列加密。具体方法请参考 Demo 中的 macSignWithText 函数。

```
+ (NSString *)macSignWithText:(NSString *)text secretKey:(NSString *)secretKey
{
NSData *saltData = [secretKey dataUsingEncoding:NSUTF8StringEncoding];
NSData *paramData = [text dataUsingEncoding:NSUTF8StringEncoding];
NSMutableData* hash = [NSMutableData dataWithLength:CC_SHA1_DIGEST_LENGTH ];
CCHmac(kCCHmacAlgSHA1, saltData.bytes, saltData.length, paramData.bytes, paramData.length,
hash.mutableBytes);
NSString *base64Hash = [hash base64EncodedStringWithOptions:0];

return base64Hash;
}
```

Demo 工程下载

上文描述的客户端代码具体实现请参考Demo 工程。请根据业务需求适当修改后再用于生产环境。

JavaScript 接入示例

本文主要介绍如何使用 JavaScript 客户端收发 MQTT 消息，并给出示例代码供前期开发测试参考。

注意：

本文给出的实例均基于 Eclipse Paho JavaScript SDK 实现，该 SDK 下载请参见MQTT 接入准备。如使用其他第三方的客户端，请适当修改示例。

1. 资源创建

使用 MQ 提供的 MQTT 服务，首先需要核实应用中使用的 Topic 资源是否已经创建，如果没有请先去控制台创建 Topic，Group ID 等资源。

创建资源时需要根据需求选择对应的 Region，例如，MQTT 需要使用华北2的接入点，那么 Topic 等资源就在华北2 创建，资源创建具体请参见快速入门指导中的**步骤二：创建资源**。

注意：MQTT 使用的多级子 Topic 不需要创建，代码里直接使用即可，没有限制。

2. MQTT 收发消息

本段示例代码演示如何使用 JavaScript 客户端收发 MQTT 消息。

config.js 文件

```
host = 'XXX.mqtt.aliyuncs.com';// 设置当前用户的接入点域名，接入点获取方法请参考接入准备章节文档，先在控制台创建
```

```
实例
port = 80;//WebSocket 协议服务端口，如果是走 HTTPS，设置443端口
topic = 'XXXXXXX';//需要操作的 Topic
useTLS = false;//是否走加密 HTTPS，如果走 HTTPS，设置为 true
accessKey = 'XXXXXX';//账号的 AccessKey，在阿里云控制台查看
secretKey = 'XXXXXX=';//账号的的 SecretKey，在阿里云控制台查看
cleansession = true;
groupId='GID_XXXX';
clientId=groupId+'@@@XXXX';//GroupId@@@DeviceId，由控制台创建的 Group ID 和自己指定的 Device ID 组合构成
```

收发消息程序：

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Aliyun Mqtt Websockets</title>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<script src="mqttws31.js" type="text/javascript"></script>
<script src="config.js" type="text/javascript"></script>
<script src="crypto-js.js" type="text/javascript"></script>
<script type="text/javascript">
var mqtt;
var reconnectTimeout = 2000;
var username =accessKey;
var password=CryptoJS.HmacSHA1(groupId,secretKey).toString(CryptoJS.enc.Base64);
function MQTTconnect() {
mqtt = new Paho.MQTT.Client(
host,//MQTT 域名
port,//WebSocket 端口，如果使用 HTTPS 加密则配置为443,否则配置80
clientId//客户端 ClientId
);
var options = {
timeout: 3,
onSuccess: onConnect,
mqttVersion:4,
onFailure: function (message) {
setTimeout(MQTTconnect, reconnectTimeout);
}
};

mqtt.onConnectionLost = onConnectionLost;
mqtt.onMessageArrived = onMessageArrived;

if (username != null) {
options.userName = username;
options.password = password;
options.useSSL=useTLS;//如果使用 HTTPS 加密则配置为 true
}
mqtt.connect(options);
}

function onConnect() {
// Connection succeeded; subscribe to our topic
```

```
mqtt.subscribe(topic, {qos: 0});
message = new Paho.MQTT.Message("Hello mqtt!!");//set body
message.destinationName =topic;// set topic
mqtt.send(message);
}

function onConnectionLost(response) {
setTimeout(MQTTconnect, reconnectTimeout);
};

function onMessageArrived(message) {

var topic = message.destinationName;
var payload = message.payloadString;
console.log("recv msg : "+topic+" "+payload);
};
MQTTconnect();
</script>
</head>
</html>
```

Python 接入示例

本文主要介绍如何使用 Python 客户端收发 MQTT 消息，并给出示例代码供前期开发测试参考。包括资源创建、环境准备、示例代码、注意事项等。

注意：

本文给出的示例均基于 Eclipse Paho Python SDK 实现，SDK 下载请参见 MQTT 接入准备。如使用其他第三方的客户端，请适当修改示例。

1. 资源创建

使用 MQ 提供的 MQTT 服务，首先需要核实应用中使用的 Topic 资源是否已经创建。如果没有，请先去控制台创建 Topic，Group ID 等资源。

创建资源时需要根据需求选择对应的 Region，例如 MQTT 需要使用华北2的接入点，那么 Topic 等资源就在华北2 创建。资源创建具体请参见**快速入门指导**中的**步骤二：创建资源**。

注意：MQTT 使用的多级子 Topic 不需要创建，代码里直接使用即可，没有限制。

2. 环境准备

本示例使用 Python 语言编写， Python 客户端依赖的添加可以自行搜索。一般采用 pip 包管理器来处理，具体安装步骤请参考 paho 官方文档。

官方版本目前不支持 SNI。如果需要使用 SSL 加密的连接方式，请参考以下步骤安装 patch 以支持 SNI，后续等官方更新 SDK 的版本：

```
pip install git+https://github.com/exosite/paho.mqtt.python.git@openssl_sni_support --upgrade
```

3. MQTT 收发消息

本段示例代码演示如何使用 MQTT Python 客户端发送和接收消息。

获取示例代码中的接入点域名，请参见**环境准备**中的**接入点获取**。

```
#!/usr/bin/env python

import hmac
import base64
from hashlib import sha1
import time
from paho.mqtt.client import MQTT_LOG_INFO, MQTT_LOG_NOTICE, MQTT_LOG_WARNING, MQTT_LOG_ERR,
MQTT_LOG_DEBUG

from paho.mqtt import client as mqtt

#accessKey get from aliyun account console
accessKey = 'XXXXXX'
#secretKey get from aliyun account console
secretKey = 'XXXXX'
#MQTT GroupID,get from mq console
groupId = 'GID_XXX'
client_id=groupId+'@@@'+'TEST00001'
# Topic
topic = 'XXXX/python/test'
#MQTT endPoint get from mq console
brokerUrl='XXXXXXXXX'

def on_log(client, userdata, level, buf):
if level == MQTT_LOG_INFO:
head = 'INFO'
elif level == MQTT_LOG_NOTICE:
head = 'NOTICE'
elif level == MQTT_LOG_WARNING:
head = 'WARN'
elif level == MQTT_LOG_ERR:
head = 'ERR'
elif level == MQTT_LOG_DEBUG:
head = 'DEBUG'
else:
head = level
print('%s: %s' % (head, buf))

def on_connect(client, userdata, flags, rc):
print('Connected with result code ' + str(rc))

def on_message(client, userdata, msg):
```

```
print(msg.topic + ' ' + str(msg.payload))

def on_disconnect(client, userdata, rc):
if rc != 0:
print('Unexpected disconnection %s' % rc)


client = mqtt.Client(client_id, protocol=mqtt.MQTTv311, clean_session=True)
client.on_log = on_log
client.on_connect = on_connect
client.on_message = on_message
client.on_disconnect = on_disconnect
password = base64.b64encode(hmac.new(secretKey.encode(), groupId.encode(), sha1).digest()).decode()
client.username_pw_set(accessKey, password)
client.connect(brokerUrl, 1883, 60)
client.subscribe(topic, 0)
for i in range(1, 11):
print(i)
rc = client.publish(topic, str(i), qos=0)
print ('rc: %s' % rc)
time.sleep(0.1)

client.loop_forever()
```

4. SSL 方式连接服务

使用 Python 客户端通过 SSL 方式来连接服务，需要客户端支持 SNI。

如果是 Paho 提供的客户端，请根据 Release Note 确认是否支持 SNI。如果没有，请按照本文**第2步-环境准备**的说明安装 patch。

相比标准方式，使用 SSL 方式时需要注意以下两点：

- 加密端口应设置为8883。
- 需要导入服务端公钥证书，并配置 TLS 参数。证书下载链接如下：ca.crt

同理，代码上需要在连接之前，导入证书文件并设置连接的 broker 地址。

```
client.tls_set('XXX.crt', server_hostname='XXXXXXX')
client.connect(brokerUrl, 8883, 60)
```

微消息队列管理接口

创建 GroupID

本接口限企业铂金版客户专用，请前往企业铂金版购买页查看详情。

描述

OnsMqttGroupIdCreate 接口用于在服务器上创建 MQTT 分组（groupId）。

使用场景

新应用上线时，在创建 Topic 资源后，如果需要使用 MQTT 客户端收发消息，需要首先在 MQ 控制台或者调用该 API 创建 groupId 。

请求参数

名称	类型	是否必须	描述
OnsRegionId	String	是	当前查询 MQ 所在区域，可以通过 OnsRegionList 方法获取
OnsPlatform	String	否	该请求来源，默认是从 POP 平台
PreventCache	Long	是	用于 CSRF 校验，设置为系统当前时间即可
GroupId	String	是	创建的 MQTT 分组 groupId
Topic	String	是	使用的 Topic

返回参数

名称	类型	描述
RequestId	String	为公共参数，每个请求的 RequestId 独一无二
HelpUrl	String	帮助链接

相关 API

- OnsMqttGroupIdList： 获取所有的 GroupId 列表。

使用示例

本示例在 daily 区域创建名为 GID_MingduanTest 。

```
public static void main(String []args) {
```

```
String regionId = "cn-hangzhou";
String accessKey = "XXXXXXXXXXXXXXXX";
String secretKey = "XXXXXXXXXXXXXXXX";
String endPointName ="cn-hangzhou";
String productName ="Ons";
String domain ="ons.cn-hangzhou.aliyuncs.com";

/**
*根据自己需要访问的区域选择 Region，并设置对应的接入点
*/
try {
DefaultProfile.addEndpoint(endPointName，regionId，productName，domain);
} catch (ClientException e) {
e.printStackTrace();
}
IClientProfile profile= DefaultProfile.getProfile(regionId，accessKey，secretKey);
IAcsClient iAcsClient= new DefaultAcsClient(profile);
OnsMqttGroupIdCreateRequest request = new OnsMqttGroupIdCreateRequest();
/**
*ONSRegionId 是指你需要 API 访问 MQ 哪个区域的资源
*该值必须要根据 OnsRegionList 方法获取的列表来选择和配置，因为 OnsRegionId 是变动的，不能够写固定值
*/
request.setOnsRegionId("daily");
request.setPreventCache(System.currentTimeMillis());
request.setAcceptFormat(FormatType.JSON);
request.setTopic("Mingduan_67dd");
request.setGroupId("GID_MingduanTest");
try {
OnsMqttGroupIdCreateResponse response=iAcsClient.getAcsResponse(request);
System.out.println(response.getRequestId());
} catch (ServerException e) {
e.printStackTrace();
} catch (ClientException e) {
e.printStackTrace();
}
}
```

查询分组列表

OnsMqttGroupIdList 接口用于查询目标 Region 里当前用户所拥有的所有 GroupId。

使用场景

获取 GroupId 列表的接口一般用于管理用户所有 GroupId 的场景，首先获取列表，然后可以针对单个 GroupId 进行相关的数据查询。

请求参数列表

名称	类型	是否必须	描述

OnsRegionId	String	是	当前操作的 MQ 所在区域，详情参见公共术语页面
OnsPlatform	String	否	请求来源，默认是从 POP 平台
PreventCache	Long	是	用于 CSRF 校验，设置为系统当前时间即可

返回参数列表

名称	类型	描述
RequestId	String	为公共参数，每个请求独一无二，用于排查定位问题
HelpUrl	String	帮助链接
Data	MqttGroupIdDo	GroupId 信息数据结构

MqttGroupIdDo 数据结构

名称	类型	描述
Id	Long	数据编号
ChannelId	Integer	访问的途径编号，阿里云环境为 0
OnsRegionId	String	MQ 的 RegionId
RegionName	String	MQ 的 Region 名称
Owner	String	GroupId 所属账号
GroupId	String	GroupId
Topic	String	该 GroupId 关联的 Parent Topic
Status	Integer	当前状态（0 服务中 1 冻结 2 暂停）
CreateTime	Long	创建时间，单位是毫秒时间戳
UpdateTime	Long	最后更新时间，单位是毫秒时间戳

错误码列表

无

相关 API

- OnsMqttQueryClientByGroupId：根据 GroupId 查询当前在线客户端数量

- OnsMqttQueryHistoryOnline：根据 GroupId 查询历史在线数统计曲线

使用示例

本示例仅仅提供一个参考，从杭州接入点接入，获取杭州 Region 的所有 GroupId。

```
public static void main(String[] args) {
    String regionId = "cn-hangzhou";
    String accessKey = "XXXXXXXXXXXXXXXXX";
    String secretKey = "XXXXXXXXXXXXXXXXX";
    String endPointName ="cn-hangzhou";
    String productName ="Ons";
    String domain ="ons.cn-hangzhou.aliyuncs.com";

    /**
    *根据自己所在的区域选择 Region 后,设置对应的接入点
    */
    try {
        DefaultProfile.addEndpoint(endPointName,regionId,productName,domain);
    } catch (ClientException e) {
        e.printStackTrace();
    }
    IClientProfile profile= DefaultProfile.getProfile(regionId,accessKey,secretKey);
    IAcsClient iAcsClient= new DefaultAcsClient(profile);
    OnsMqttGroupIdListRequest request = new OnsMqttGroupIdListRequest();
    /**
    *ONSRegionId 是指你需要 API 访问 MQ 哪个区域的资源。
    *该值必须要根据 OnsRegionList 方法获取的列表来选择和配置，因为 OnsRegionId 是变动的，不能够写固定值
    */
    request.setOnsRegionId("XXXX");
    request.setPreventCache(System.currentTimeMillis());
    request.setAcceptFormat(FormatType.JSON);
    try {
      OnsMqttGroupIdListResponse response=iAcsClient.getAcsResponse(request);
      List<OnsMqttGroupIdListResponse.MqttGroupIdDo> groupIdList=response.getData();
      for(OnsMqttGroupIdListResponse.MqttGroupIdDo groupDo:groupIdList){
        System.out.println(groupDo.getId()+" "+
                           groupDo.getChannelId()+" "+
                           groupDo.getOnsRegionId()+" "+
                           groupDo.getRegionName()+" "+
                           groupDo.getOwner()+" "+
                           groupDo.getGroupId()+" "+
                           groupDo.getTopic()+" "+
                           groupDo.getStatus()+" "+
                           groupDo.getCreateTime()+" "+
                           groupDo.getUpdateTime());
      }
    } catch (ServerException e) {
      e.printStackTrace();
    } catch (ClientException e) {
      e.printStackTrace();
    }
}
```

查询设备在线信息

OnsMqttQueryClientByClientId 接口用于查询指定设备的在线信息以及订阅关系等数据，查询条件是 clientId。

使用场景

查询设备信息的接口一般用于线上追踪单个设备的运行状态及排查问题。输入 clientId 即可查到对应设备是否在线，设备地址，以及当前的订阅关系等信息。

使用限制

由于OpenAPI面向的场景是用户自定义管控开发，服务端会对过快的调用进行限流（每分钟30次），因此不要在业务的主流程中使用本接口。如需判断设备是否在线，请使用设备上下线通知功能。

请求参数列表

名称	类型	是否必须	描述
OnsRegionId	String	是	当前操作的 MQ 所在区域，详情参见公共术语。
OnsPlatform	String	否	请求来源，默认是从 POP 平台。
PreventCache	Long	是	用于 CSRF 校验，设置为系统当前时间即可。
ClientId	String	是	需要查询的目标 clientId。

返回参数列表

名称	类型	描述
RequestId	String	为公共参数，每个请求独一无二，用于排查定位问题。
HelpUrl	String	帮助链接
Data	MqttClientInfoDo	设备在线信息数据结构

MqttClientInfoDo 数据结构

名称	类型	描述
Online	Boolean	设备是否在线
ClientId	String	设备的 clientId 名称

SocketChannel	String	设备连接的 IP 地址
LastTouch	Long	最后更新时间
SubScriptonData	List(SubscriptionDo)	该设备当前的订阅关系集合

SubscriptionDo 数据结构

名称	类型	描述
ParentTopic	String	MQTT 的一级父 Topic
SubTopic	String	MQTT 的多级子 Topic，如果没有则为 NULL。
Qos	Integer	订阅关系的 QoS 级别

错误码列表

无

相关 API

OnsMqttQueryClientByTopic：根据 Topic 查询当前订阅该 Topic 的在线客户端数量。

使用示例

本示例仅仅提供一个参考，从杭州接入点接入，查询指定 clientId 的在线数据。

```
public static void main(String[] args) {
    String regionId = "cn-hangzhou";
    String accessKey = "XXXXXXXXXXXXXXXXX";
    String secretKey = "XXXXXXXXXXXXXXXXX";
    String endPointName ="cn-hangzhou";
    String productName ="Ons";
    String domain ="ons.cn-hangzhou.aliyuncs.com";

    /**
    *根据自己所在的区域选择 Region 后,设置对应的接入点。
    */
    try {
        DefaultProfile.addEndpoint(endPointName,regionId,productName,domain);
    } catch (ClientException e) {
        e.printStackTrace();
    }
    IClientProfile profile= DefaultProfile.getProfile(regionId,accessKey,secretKey);
    IAcsClient iAcsClient= new DefaultAcsClient(profile);
    OnsMqttQueryClientByClientIdRequest request = new OnsMqttQueryClientByClientIdRequest();
    /**
    *ONSRegionId 是指你需要 API 访问 MQ 哪个区域的资源。
    *该值必须要根据 OnsRegionList 方法获取的列表来选择和配置，因为 OnsRegionId 是变动的，不能够写固定值。
    */
    request.setOnsRegionId("XXXX");
```

```
        request.setPreventCache(System.currentTimeMillis());
        request.setAcceptFormat(FormatType.JSON);
        request.setClientId("GID_XXX@@@XXXXX");
        try {
                OnsMqttQueryClientByClientIdResponse response = iAcsClient.getAcsResponse(request);
                OnsMqttQueryClientByClientIdResponse.MqttClientInfoDo clientInfoDo =
response.getMqttClientInfoDo();
                System.out.println(clientInfoDo.getOnline() + "  " +
                        clientInfoDo.getClientId() + "  " +
                        clientInfoDo.getLastTouch() + "  " +
                        clientInfoDo.getSocketChannel());
                for (OnsMqttQueryClientByClientIdResponse.MqttClientInfoDo.SubscriptionDo subscriptionDo :
clientInfoDo.getSubScriptonData()) {
                    System.out.println(subscriptionDo.getParentTopic() + "   " + subscriptionDo.getSubTopic() + "  " +
subscriptionDo.getQos());
                }
        } catch (ServerException e) {
          e.printStackTrace();
        } catch (ClientException e) {
          e.printStackTrace();
        }
    }
```

查询分组在线数量

OnsMqttQueryClientByGroupId 接口根据 GroupId 统计属于该分组下的在线设备的数量。

使用场景

查询在线设备数量的接口一般用于业务分析，统计一个 GroupId 分组下终端设备的活跃程度。

使用限制

由于OpenAPI面向的场景是用户自定义管控开发，服务端会对过快的调用进行限流（每分钟30次），因此不要在业务的主流程中使用本接口。

请求参数列表

名称	类型	是否必须	描述
OnsRegionId	String	是	当前操作的 MQ 所在区域，详情参见公共术语页面
OnsPlatform	String	否	请求来源，默认是从 POP 平台
PreventCache	Long	是	用于 CSRF 校验，设置

			为系统当前时间即可
GroupId	String	是	需要查询的目标分组 GroupId

返回参数列表

名称	类型	描述
RequestId	String	为公共参数，每个请求独一无二，用于排查定位问题
HelpUrl	String	帮助链接
MqttClientSetDo	MqttClientSetDo	分组在线信息数据结构

MqttClientSetDo 数据结构

名称	类型	描述
OnlineCount	Long	分组所有在线设备数量

错误码列表

无

相关 API

- OnsMqttQueryClientByTopic：根据 Topic 查询当前订阅该 Topic 的在线客户端数量

使用示例

本示例仅仅提供一个参考，从杭州接入点接入，查询指定 groupId 的在线数量信息。

```
public static void main(String[] args) {
    String regionId = "cn-hangzhou";
    String accessKey = "XXXXXXXXXXXXXXXX";
    String secretKey = "XXXXXXXXXXXXXXXX";
    String endPointName ="cn-hangzhou";
    String productName ="Ons";
    String domain ="ons.cn-hangzhou.aliyuncs.com";

    /**
    *根据自己所在的区域选择 Region 后,设置对应的接入点
    */
    try {
        DefaultProfile.addEndpoint(endPointName,regionId,productName,domain);
    } catch (ClientException e) {
        e.printStackTrace();
    }
    IClientProfile profile= DefaultProfile.getProfile(regionId,accessKey,secretKey);
    IAcsClient iAcsClient= new DefaultAcsClient(profile);
```

```
    OnsMqttQueryClientByGroupIdRequest request = new OnsMqttQueryClientByGroupIdRequest();
    /**
    *ONSRegionId 是指你需要 API 访问 MQ 哪个区域的资源。
    *该值必须要根据 OnsRegionList 方法获取的列表来选择和配置，因为 OnsRegionId 是变动的，不能够写固定值
    */
    request.setOnsRegionId("XXXX");
    request.setPreventCache(System.currentTimeMillis());
    request.setAcceptFormat(FormatType.JSON);
    request.setGroupId("GID_XXXXX");
    try {
            OnsMqttQueryClientByGroupIdResponse response = iAcsClient.getAcsResponse(request);
           OnsMqttQueryClientByGroupIdResponse.MqttClientSetDo clientSetDo = response.getMqttClientSetDo();
           System.out.println(clientSetDo.getOnlineCount() + "  " +
                   clientSetDo.getPersistCount() + "  ");
    } catch (ServerException e) {
      e.printStackTrace();
    } catch (ClientException e) {
      e.printStackTrace();
    }
}
```

根据 Topic 查询在线数量

OnsMqttQueryClientByTopic 接口根据 MQTT 的 Topic 统计订阅该 Topic 的在线设备的数量。

使用场景

根据 Topic 查询在线设备数量的接口一般用于业务分析。业务上订阅某个特定 Topic 的设备一般是参与某个特殊活动的设备，可以根据 Topic 查询参与该活动的设备数量。

使用限制

由于OpenAPI面向的场景是用户自定义管控开发，服务端会对过快的调用进行限流（每分钟30次），因此不要在业务的主流程中使用本接口。如需判断设备是否在线，请使用设备上下线通知功能。

请求参数列表

名称	类型	是否必须	描述
OnsRegionId	String	是	当前操作的 MQ 所在区域，详情参见公共术语页面
OnsPlatform	String	否	请求来源，默认是从 POP 平台
PreventCache	Long	是	用于 CSRF 校验，设置为系统当前时间即可

ParentTopic	String	是	需要查询的目标 MQTT 一级父 Topic
SubTopic	String	否	需要查询的目标子 Topic，如果没有则不填

返回参数列表

名称	类型	描述
RequestId	String	为公共参数，每个请求独一无二，用于排查定位问题
HelpUrl	String	帮助链接
MqttClientSetDo	MqttClientSetDo	分组在线信息数据结构

MqttClientSetDo 数据结构

名称	类型	描述
OnlineCount	Long	分组所有在线设备数量

错误码列表

无

相关 API

- OnsMqttQueryClientByTopic：根据 Topic 查询当前订阅该 Topic 的在线客户端数量

使用示例

本示例仅仅提供一个参考，从杭州接入点接入，查询指定 groupId 的在线数量信息。

```
public static void main(String[] args) {
    String regionId = "cn-hangzhou";
    String accessKey = "XXXXXXXXXXXXXXXXX";
    String secretKey = "XXXXXXXXXXXXXXXXX";
    String endPointName ="cn-hangzhou";
    String productName ="Ons";
    String domain ="ons.cn-hangzhou.aliyuncs.com";

    /**
    *根据自己所在的区域选择 Region 后,设置对应的接入点
    */
    try {
        DefaultProfile.addEndpoint(endPointName,regionId,productName,domain);
    } catch (ClientException e) {
        e.printStackTrace();
    }
```

```
    IClientProfile profile= DefaultProfile.getProfile(regionId,accessKey,secretKey);
    IAcsClient iAcsClient= new DefaultAcsClient(profile);
    OnsMqttQueryClientByTopicRequest request = new OnsMqttQueryClientByTopicRequest();
    /**
    *ONSRegionId 是指你需要 API 访问 MQ 哪个区域的资源。
    *该值必须要根据 OnsRegionList 方法获取的列表来选择和配置，因为 OnsRegionId 是变动的，不能够写固定值
    */
    request.setOnsRegionId("XXXX");
    request.setPreventCache(System.currentTimeMillis());
    request.setAcceptFormat(FormatType.JSON);
    request.setParentTopic("XXXX");
    request.setSubTopic("/secTopic/0/1");//此处如果没有子 topic 则不填，如果有，一定是完整的信息
    try {
            OnsMqttQueryClientByTopicResponse response = iAcsClient.getAcsResponse(request);
            OnsMqttQueryClientByTopicResponse.MqttClientSetDo clientSetDo = response.getMqttClientSetDo();
            System.out.println(clientSetDo.getOnlineCount() + "  " +
                    clientSetDo.getPersistCount() + "  ");
    } catch (ServerException e) {
      e.printStackTrace();
    } catch (ClientException e) {
      e.printStackTrace();
    }
}
```

查询历史在线统计

OnsMqttQueryHistoryOnline 接口根据 GroupId 分组和给定的时间段查询历史在线设备的数量曲线。

使用场景

查询历史在线设备数曲线的功能一般用于生成业务报表。

使用限制

由于OpenAPI面向的场景是用户自定义管控开发，服务端会对过快的调用进行限流（每分钟30次），因此不要在业务的主流程中使用本接口。如需判断设备是否在线，请使用设备上下线通知功能。

请求参数列表

名称	类型	是否必须	描述
OnsRegionId	String	是	当前操作的 MQ 所在区域，详情参见公共术语。
OnsPlatform	String	否	请求来源，默认是从 POP 平台。

PreventCache	Long	是	用于 CSRF 校验，设置为系统当前时间即可。
GroupId	String	是	需要查询的目标分组 GroupId。
BeginTime	Long	是	查询的起始时间
EndTime	Long	是	查询的终止时间，起止时间范围建议尽可能在当天，否则后端会自动截断。

返回参数列表

名称	类型	描述
RequestId	String	为公共参数，每个请求独一无二。
HelpUrl	String	帮助链接
data	Data	数据集合

Data 数据集定义

名称	类型	描述
Title	String	table 的名称
Records	List(StatsDataDo)	采集点信息

StatsDataDo 数据集定义

名称	类型	描述
X	Long	横轴，毫秒时间戳。
Y	Float	纵轴，数量。

错误码列表

无

相关 API

OnsMqttQueryClientByGroupId：根据 GroupId 查询当前在线客户端数量。

使用示例

本示例仅仅提供一个参考，从杭州接入点接入，查询指定 GroupId 过去一小时的在线数量信息。

```
public static void main(String[] args) {
```

```
    String regionId = "cn-hangzhou";
    String accessKey = "XXXXXXXXXXXXXXXX";
    String secretKey = "XXXXXXXXXXXXXXXX";
    String endPointName ="cn-hangzhou";
    String productName ="Ons";
    String domain ="ons.cn-hangzhou.aliyuncs.com";

    /**
    *根据自己所在的区域选择 Region 后,设置对应的接入点。
    */
    try {
        DefaultProfile.addEndpoint(endPointName,regionId,productName,domain);
    } catch (ClientException e) {
        e.printStackTrace();
    }
    IClientProfile profile= DefaultProfile.getProfile(regionId,accessKey,secretKey);
    IAcsClient iAcsClient= new DefaultAcsClient(profile);
    OnsMqttQueryHistoryOnlineRequest request = new OnsMqttQueryHistoryOnlineRequest();
    /**
    *ONSRegionId 是指你需要 API 访问 MQ 哪个区域的资源。
    *该值必须要根据 OnsRegionList 方法获取的列表来选择和配置，因为 OnsRegionId 是变动的，不能够写固定值。
    */
    request.setOnsRegionId("XXXX");
    request.setPreventCache(System.currentTimeMillis());
    request.setAcceptFormat(FormatType.JSON);
    request.setGroupId("XXX");
    request.setBeginTime(System.currentTimeMillis()-1000*3600);
    request.setEndTime(System.currentTimeMillis());
    try {
            OnsMqttQueryHistoryOnlineResponse response = iAcsClient.getAcsResponse(request);
              OnsMqttQueryHistoryOnlineResponse.Data data =response.getData();
            System.out.println(data.getTitle()+"\n"+
            data.getRecords());
    } catch (ServerException e) {
      e.printStackTrace();
    } catch (ClientException e) {
      e.printStackTrace();
    }
}
```

消息查询

OnsMqttQueryMsgByPubInfo 接口根据发送端信息查询消息。

使用场景

根据发送方信息查询消息的接口一般用于线上排查问题，或者查询某条消息的推送到达率等场景使用。需要输入发送方的详细信息来定位特定的消息。条件越精确，匹配出的消息越精确。获取到消息的 TraceId 之后即可

根据 TraceId 调用 OnsMqttQueryTraceByTraceId 接口查询该条消息的实际推送量。

使用限制

由于OpenAPI面向的场景是用户自定义管控开发，服务端会对过快的调用进行限流（每分钟30次），因此不要在业务的主流程中使用本接口。如需判断设备是否在线，请使用设备上下线通知功能。

请求参数列表

名称	类型	是否必须	描述
OnsRegionId	String	是	当前操作的 MQ 所在区域，详情参见公共术语。
OnsPlatform	String	否	请求来源，默认是从 POP 平台。
PreventCache	Long	是	用于 CSRF 校验，设置为系统当前时间即可。
Topic	String	是	需要查询的消息的完整 Topic。
ClientId	String	是	该消息的发送方 ClientId。
BeginTime	Long	是	消息发送时间的起始范围。
EndTime	Long	是	消息发送时间的终止范围，时间范围越小查询越精确，默认只允许查询30分钟区间。

返回参数列表

名称	类型	描述
RequestId	String	为公共参数，每个请求独一无二,用于排查定位问题。
HelpUrl	String	帮助链接
MqttMessageDos	List(MqttMessageDo)	匹配到的所有消息

MqttMessageDo 数据结构

名称	类型	描述
PubTime	Long	消息的精确发送时间
TraceId	String	消息唯一的 TraceId，根据该 ID 可以查询到推送量。

错误码列表

无

相关 API

OnsMqttQueryTraceByTraceId：根据 TraceId 查询指定消息的推送量。

使用示例

本示例仅仅提供一个参考，从杭州接入点接入，查询指定信息查询符合条件的所有消息。

```
public static void main(String[] args) {
    String regionId = "cn-hangzhou";
    String accessKey = "XXXXXXXXXXXXXXXXX";
    String secretKey = "XXXXXXXXXXXXXXXXX";
    String endPointName ="cn-hangzhou";
    String productName ="Ons";
    String domain ="ons.cn-hangzhou.aliyuncs.com";

    /**
    *根据自己所在的区域选择 Region 后,设置对应的接入点。
    */
    try {
        DefaultProfile.addEndpoint(endPointName,regionId,productName,domain);
    } catch (ClientException e) {
        e.printStackTrace();
    }
    IClientProfile profile= DefaultProfile.getProfile(regionId,accessKey,secretKey);
    IAcsClient iAcsClient= new DefaultAcsClient(profile);
    OnsMqttQueryMsgByPubInfoRequest request = new OnsMqttQueryMsgByPubInfoRequest();
    /**
    *ONSRegionId 是指你需要 API 访问 MQ 哪个区域的资源。
    *该值必须要根据 OnsRegionList 方法获取的列表来选择和配置，因为 OnsRegionId 是变动的，不能够写固定值。
    */
    request.setOnsRegionId("XXXX");
    request.setPreventCache(System.currentTimeMillis());
    request.setAcceptFormat(FormatType.JSON);
    request.setBeginTime(System.currentTimeMillis()-1000*60);
    request.setEndTime(System.currentTimeMillis());
    request.setClientId("GID_XXX@@@XXXX");//发送方的 clientId
    request.setTopic("XXX/XX/XXX");
    try {
            OnsMqttQueryMsgByPubInfoResponse response = iAcsClient.getAcsResponse(request);
            List<OnsMqttQueryMsgByPubInfoResponse.MqttMessageDo> msgList =
response.getMqttMessageDos();
            for(OnsMqttQueryMsgByPubInfoResponse.MqttMessageDo msg:msgList){
                System.out.println(msg.getpubTime() + "  " +msg.getTraceId());
            }
    } catch (ServerException e) {
      e.printStackTrace();
    } catch (ClientException e) {
      e.printStackTrace();
```

```
    }
}
```

消息收发统计

OnsMqttQueryMsgTransTrend 接口根据 Topic 等信息查询历史消息收发量。

使用场景

查询历史消息收发曲线功能一般用于生成数据报表，统计业务规模等场景。

使用限制

由于OpenAPI面向的场景是用户自定义管控开发，服务端会对过快的调用进行限流（每分钟30次），因此不要在业务的主流程中使用本接口。如需判断设备是否在线，请使用设备上下线通知功能。

请求参数列表

名称	类型	是否必须	描述
OnsRegionId	String	是	当前操作的 MQ 所在区域，详情参见公共术语。
OnsPlatform	String	否	请求来源，默认是从 POP 平台。
PreventCache	Long	是	用于 CSRF 校验，设置为系统当前时间即可。
ParentTopic	String	是	需要查询的一级父 Topic。
SubTopic	String	否	查询的子 Topic，如果没有或者查询所有的 Topic 信息，可以不填。
MsgType	String	否	消息类别，p2p 或者是 sub，如果不填则默认查所有消息，填其他值无效。
TpsType	String	是	查询类别，TPS 或者 SUM，分别代表是查询总量还是 TPS 信息，其他值无效。
TransType	String	是	收发类别，PUB 或者是 SUB，代表查询发

			送还是接收，其他值无效。
Qos	Integer	是	查询的 QoS 级别，0,1,2，填其他值或者不填都默认查所有消息。
BeginTime	Long	是	查询的起始时间
EndTime	Long	是	查询的终止时间，起止时间范围建议尽可能在当天，否则后端会自动截断。

返回参数列表

名称	类型	描述
RequestId	String	为公共参数，每个请求独一无二。
HelpUrl	String	帮助链接
data	Data	数据集合

Data 数据集定义

名称	类型	描述
Title	String	table 的名称
Records	List(StatsDataDo)	采集点信息

StatsDataDo 数据集定义

名称	类型	描述
X	Long	横轴，毫秒时间戳。
Y	Float	纵轴，数据（TPS 或者总量）。

错误码列表

无

相关 API

无

使用示例

本示例仅仅提供一个参考，从杭州接入点接入，查询指定 Topic 过去一小时的消息发送数量信息。

```
public static void main(String[] args) {
    String regionId = "cn-hangzhou";
    String accessKey = "XXXXXXXXXXXXXXXXX";
    String secretKey = "XXXXXXXXXXXXXXXXX";
    String endPointName ="cn-hangzhou";
    String productName ="Ons";
    String domain ="ons.cn-hangzhou.aliyuncs.com";

    /**
    *根据自己所在的区域选择 Region 后,设置对应的接入点。
    */
    try {
        DefaultProfile.addEndpoint(endPointName,regionId,productName,domain);
    } catch (ClientException e) {
        e.printStackTrace();
    }
    IClientProfile profile= DefaultProfile.getProfile(regionId,accessKey,secretKey);
    IAcsClient iAcsClient= new DefaultAcsClient(profile);
    OnsMqttQueryMsgTransTrendRequest request = new OnsMqttQueryMsgTransTrendRequest();
    /**
    *ONSRegionId 是指你需要 API 访问 MQ 哪个区域的资源。
    *该值必须要根据 OnsRegionList 方法获取的列表来选择和配置，因为 OnsRegionId 是变动的，不能够写固定值。
    */
    request.setOnsRegionId("XXXX");
    request.setPreventCache(System.currentTimeMillis());
    request.setAcceptFormat(FormatType.JSON);
    request.setParentTopic("MQTT_TOPIC_ONSMONITOR_BJ");
    //request.setMsgType("sub");
    request.setTpsType("SUM");
    request.setTransType("PUB");
    //request.setQos(1);
    request.setBeginTime(System.currentTimeMillis()-1000*3600);
    request.setEndTime(System.currentTimeMillis());
    try {
            OnsMqttQueryMsgTransTrendResponse response = iAcsClient.getAcsResponse(request);
              OnsMqttQueryMsgTransTrendResponse.Data data =response.getData();
            System.out.println(data.getTitle()+"\n"+
            data.getRecords());
    } catch (ServerException e) {
      e.printStackTrace();
    } catch (ClientException e) {
      e.printStackTrace();
    }
}
```

消息推送量查询

OnsMqttQueryTraceByTraceId 接口根据消息 TraceId 查询指定消息的推送量。

使用场景

根据消息 TraceId 查询消息推送量的接口一般用于线上排查问题，或者统计特定消息的推送到达率等运营数据。

使用限制

由于OpenAPI面向的场景是用户自定义管控开发，服务端会对过快的调用进行限流（每分钟30次），因此不要在业务的主流程中使用本接口。如需判断设备是否在线，请使用设备上下线通知功能。

请求参数列表

名称	类型	是否必须	描述
OnsRegionId	String	是	当前操作的 MQ 所在区域，详情参见公共术语。
OnsPlatform	String	否	请求来源，默认是从 POP 平台。
PreventCache	Long	是	用于 CSRF 校验，设置为系统当前时间即可。
Topic	String	是	需要查询的消息的完整 Topic。
TraceId	String	是	该消息的 TraceId，由 OnsMqttQueryMsgByPubInfo 接口查询。

返回参数列表

名称	类型	描述
RequestId	String	为公共参数，每个请求独一无二，用于排查定位问题。
HelpUrl	String	帮助链接
PushCount	Long	消息的实际推送量

错误码列表

无

相关 API

OnsMqttQueryMsgByPubInfo：根据发送方信息查询消息，获取消息的 TraceId。

使用示例

本示例仅仅提供一个参考，从杭州接入点接入，根据给定的 TraceId 查询推送量。

```
public static void main(String[] args) {
    String regionId = "cn-hangzhou";
    String accessKey = "XXXXXXXXXXXXXXXXX";
    String secretKey = "XXXXXXXXXXXXXXXXX";
    String endPointName ="cn-hangzhou";
    String productName ="Ons";
    String domain ="ons.cn-hangzhou.aliyuncs.com";

    /**
    *根据自己所在的区域选择 Region 后,设置对应的接入点。
    */
    try {
        DefaultProfile.addEndpoint(endPointName,regionId,productName,domain);
    } catch (ClientException e) {
        e.printStackTrace();
    }
    IClientProfile profile= DefaultProfile.getProfile(regionId,accessKey,secretKey);
    IAcsClient iAcsClient= new DefaultAcsClient(profile);
    OnsMqttQueryTraceByTraceIdRequest request = new OnsMqttQueryTraceByTraceIdRequest();
    /**
    *ONSRegionId 是指你需要 API 访问 MQ 哪个区域的资源。
    *该值必须要根据 OnsRegionList 方法获取的列表来选择和配置，因为 OnsRegionId 是变动的，不能够写固定值。
    */
    request.setOnsRegionId("XXXX");
    request.setPreventCache(System.currentTimeMillis());
    request.setAcceptFormat(FormatType.JSON);
    request.setTraceId("XXXXXXXXXxXXX");
    request.setTopic("XXX/XX/XXX");
    try {
            OnsMqttQueryTraceByTraceIdResponse response = iAcsClient.getAcsResponse(request);
           System.out.println("pushCount:"+response.getPushCount());
    } catch (ServerException e) {
      e.printStackTrace();
    } catch (ClientException e) {
      e.printStackTrace();
    }
}
```

控制台使用指南

MQTT 实例管理

本文介绍如何对 MQTT 实例进行续费、变配及运维管理。

MQTT 实例续费

MQTT 实例到期之前7天会按照帐号给对应的手机发送通知，请及时续费。如果到期未续费，将保留实例7天，7天后将释放该实例。

在 MQ 控制台的左侧导航栏，选择 **MQTT 管理>实例管理。**

在需要续费的实例的右侧操作列，单击**实例详情。**

在**实例详情**页面右上角，单击**续费。**

MQTT 实例变配

服务运行期间，如果发现由于业务规模增长，需要对原有规格的实例进行扩容，则需要进行变配。

在 MQ 控制台的左侧导航栏，选择 **MQTT 管理>实例管理。**

在需要变配的实例的右侧操作列，单击**实例详情。**

在**实例详情**页面右上角，单击**升级。**

注意：

- 普通版实例升级配置一般实时生效；企业铂金版一般需要 MQ 技术人员介入，会需要一定时间，升级期间服务运行不会受到影响。
- 实例降配为实时生效，但不退款，所以建议在当前实例付费周期即将结束时再进行降配操作。
- 实例变配不能跨版本操作，即普通版无法变更为企业铂金版，企业铂金版也不能变更为普通版。

MQTT 实例运维

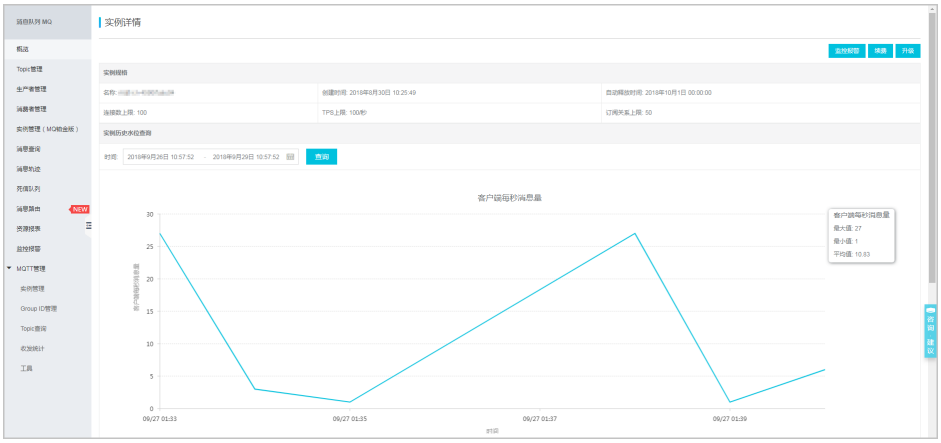
实例运行期间可以查看当前实例的历史水位曲线，包括消息收发 TPS、同时在线连接数、订阅关系数等信息，也可以设置实例报警，实时监控实例使用情况。

查看实例信息

在 MQ 控制台的左侧导航栏，选择 **MQTT 管理>实例管理。**

在需要查看信息的实例的右侧操作列，单击**实例详情。**

在**实例详情**页面的**共享历史水位查询**区域，设置时间范围，然后单击**查询。**



设置实例报警

除了人工查询历史水位，MQ 还提供了监控报警功能。在监测到实际使用量超过设置的报警阈值后，MQ 会向联系人手机发送报警短信，提示尽快升级规格。

报警阈值默认为规格的70%，报警阈值和报警开关都可以自定义设置。

在 MQ 控制台的左侧导航栏，选择 **MQTT 管理**>**实例管理**。

在需要设置报警的实例的右侧操作列，单击**实例详情**。

在**实例详情**页面右上角，单击**监控报警**。

在**编辑实例报警**对话框，输入报警阈值，然后单击**确定**。

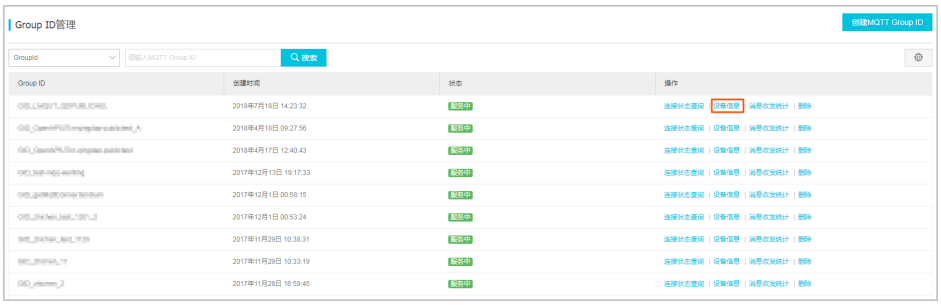
MQTT 设备查询

查询设备信息

根据单个设备独一无二的 Client ID，可以查询该设备的在线状态以及订阅关系等信息。

在 MQ 控制台左侧菜单栏，选择 **MQTT 管理** > **Group ID 管理**。

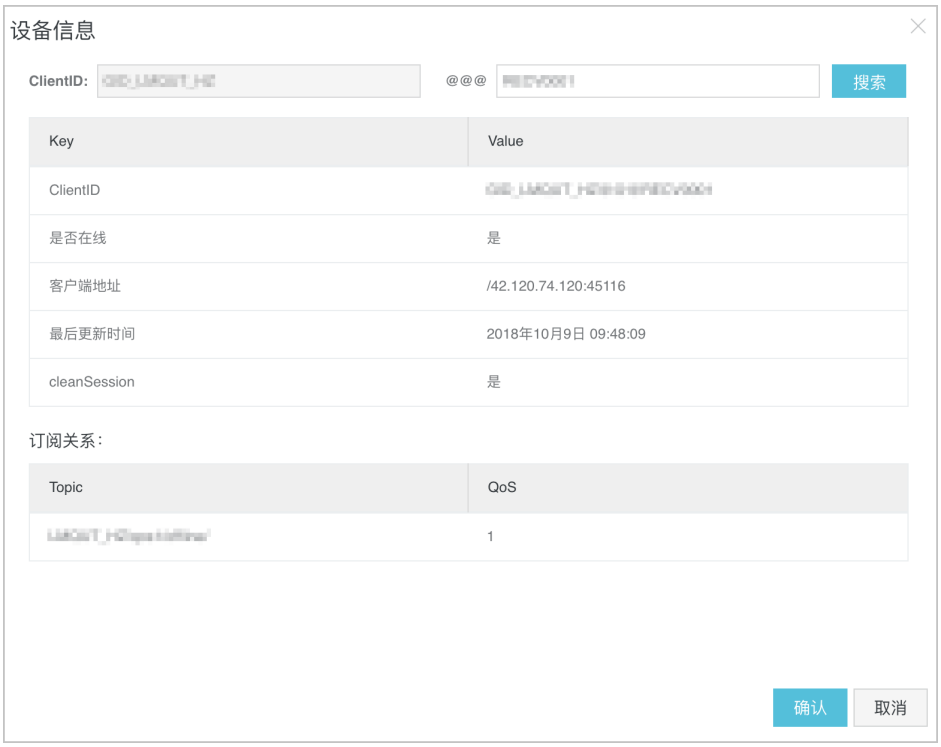
在 **Group ID 管理**页面，在需要查看的 Group ID 右侧，单击操作列的**设备信息**。



在**设备信息**对话框，输入具体的 Device ID 信息，然后单击**搜索**按钮。

查询结果包含两部分：

- 在线情况：包含设备的客户端地址以及最后更新时间。
- 订阅关系：显示当前 Client ID 管理的所有 Topic 以及 QoS 级别信息。

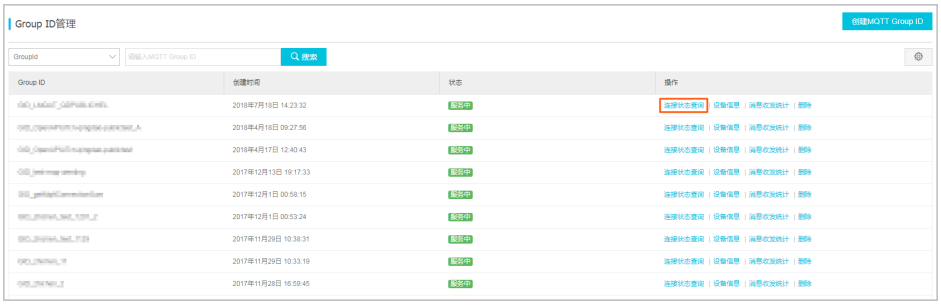


查询连接状态

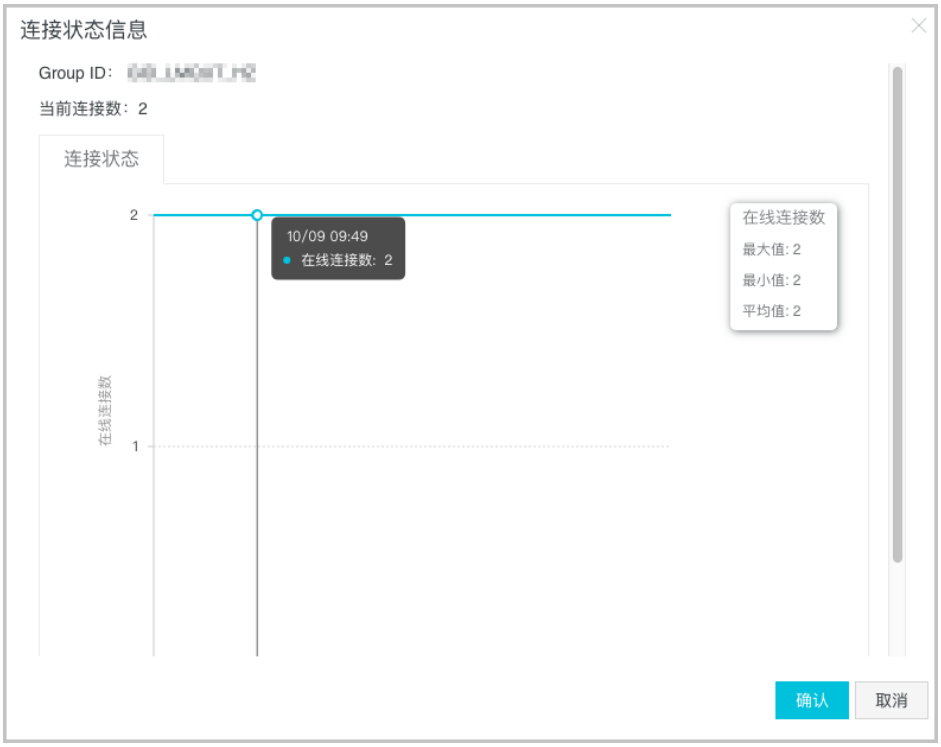
根据特定的 Group ID，还可以查询当前 Group ID 下实际的设备在线统计信息，一般用于宏观统计。状态查询的入口位于 Group ID 管理页面。

在 MQ 控制台左侧菜单栏，选择 **MQTT 管理** > **Group ID 管理**。

在 **Group ID 管理**页面，在需要查看的 Group ID 右侧，单击操作列的**连接状态查询**。



在**连接状态信息**提示框，连接状态曲线显示最近15分钟，以该 Group ID 为前缀的设备在线统计数据，该统计值基于分布式数据采集组件收集，可能存在延迟和少量的误差。

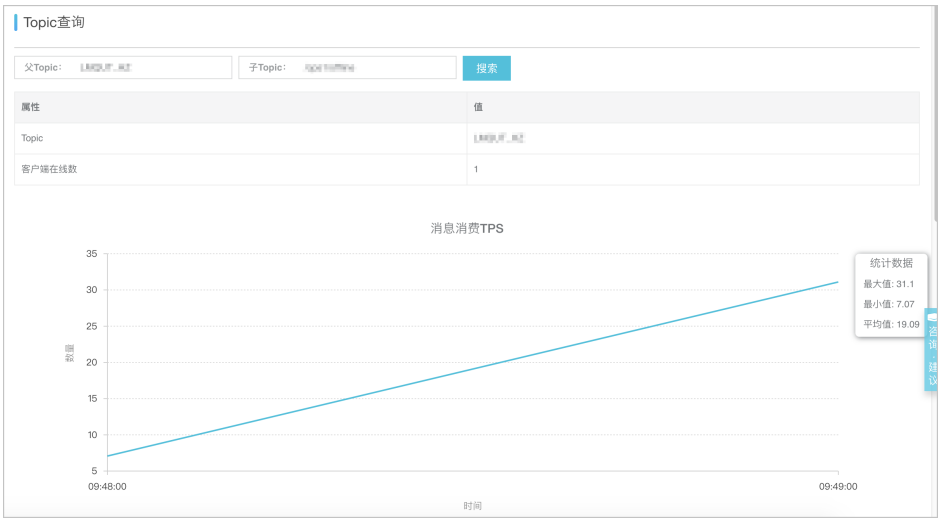


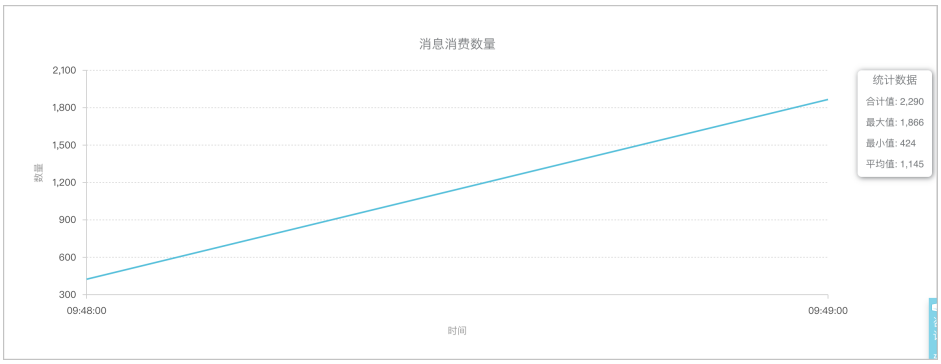
MQTT Topic 查询

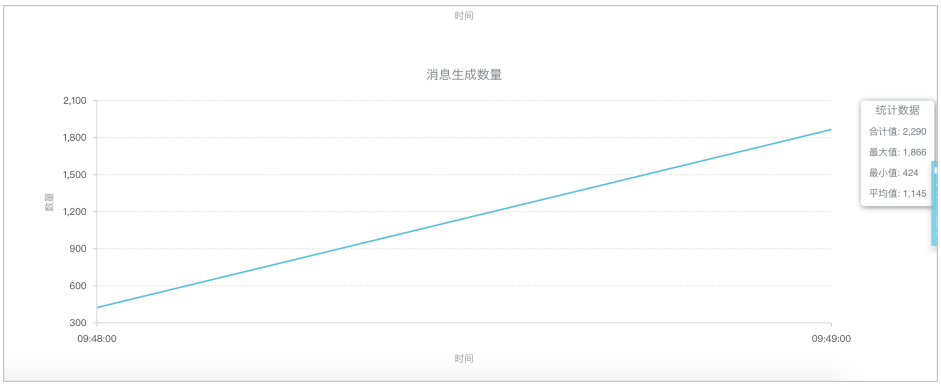
根据 MQTT 的 Topic 信息，可以查询订阅该 Topic 的所有设备信息，以及该 Topic 最近 15 分钟内的消息收发情况。

在 MQ 控制台左侧导航栏，选择 **MQTT 管理** > **Topic 查询**.

输入完整的 Topic 信息，然后单击**搜索**按钮。





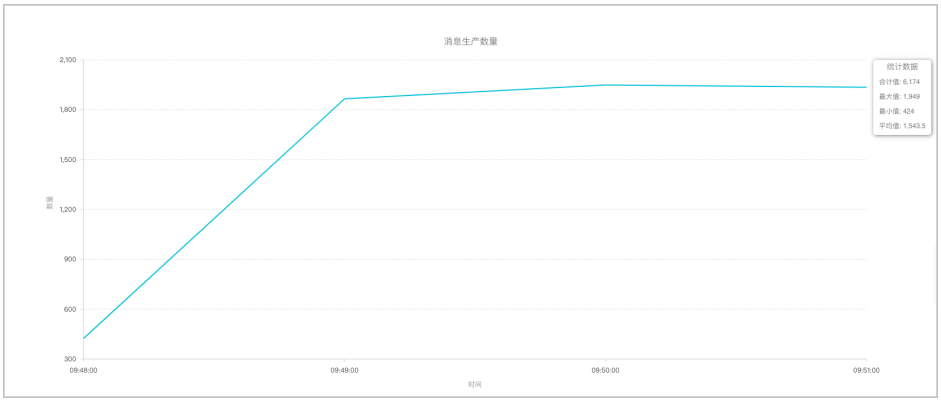


MQTT 消息收发统计

您可以在 MQ 控制台查看 MQTT 所有的 Topic 的收发消息数。此查询支持多维度筛选，以及自定义时间范围查询。

在 MQ 控制台的左侧导航栏，选择 **MQTT 管理** > **收发统计**。

输入查询筛选的条件，然后单击**搜索**按钮。



常见问题

问题1：收到使用非法 Group ID 的短信报警

原因：

MQTT 接入流程中，客户端使用的 Group ID 前缀是需要到控制台先申请再使用的。如果没有申请就使用，可能会导致连接断开，无法正常收发消息。

建议：

根据报警短信提示的非法 Group ID，登录 MQ 控制台，进入 **MQTT 管理** > **Group ID 管理**，确认是否已经创建。如果没有创建，您需要手动申请创建。Group ID 的命名必须符合规范才可以申请成功。

问题2：收到离线消息存储数量超过系统限制的短信报警

原因：

MQTT 对于每个实例中存储的离线消息数量是有限制的。关于具体的限制值，请参见微消息队列使用限制中的相关说明。如果客户端订阅关系设置不当，产生了大量离线消息，超过了规格限制，则系统会从最早的消息清理，直到满足规格限制。

建议：

- 根据报警短信提示的实例 ID，确认接入对应实例的客户端使用的是否为持久化 session，以及是否使用了 QoS1 级别的订阅关系。
- 确认上述客户端是否存在大量不在线的情况。
- 确认是否还有消息发送给这些离线的客户端。
- 确认如果属于错误使用，可以修改为 CleanSession 模式，并提工单请求清理错误的数据。

注意：持久化 session 的订阅关系仅适用于客户端每次上线时 clientId 固定且上线后还需要获取离线期间没有收到的消息的场景。如果客户端 clientId 每次上线时都不固定，或者业务上不需要关注离线状态，请使用 CleanSession 模式，具体说明请见微消息队列名词解释中的相关解释。

问题3：连接被异常断开

原因：

- MQTT 服务端在客户端发送 Publish 和 Subscribe 报文的时候进行权限验证，如果权限验证失败则会断开连接。
- 不同的客户端使用相同的 Client ID 连接 MQTT 服务，会被强制断开。

建议：

客户端确保自己的 Client ID 全局唯一，不要重复连接，同时做好断线重连的逻辑。

问题4：之前订阅过的 Topic 消息还在继续推送

原因：

MQTT 协议中订阅关系是持久化的，因此如果不需要订阅某些 Topic，需要调用 unsubscribe 方法取消订阅关系。

问题5：为什么有的 Topic 的消息能收到，有的收不到

原因：

每个 MQTT 客户端允许持有的订阅关系数量有限制，具体值参见微消息队列使用限制。如果一个客户端试图订阅超过该限制数量的 Topic ，则会被丢弃，导致收不到这部分消息。