

消息队列 MQTT 版

最佳实践



腾讯云

【 版权声明 】

©2013–2025 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【 商标声明 】



及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【 服务声明 】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【 联系我们 】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或 95716。

文档目录

最佳实践

 订阅模式

 共享订阅

MQTT 客户端开发注意事项

最佳实践

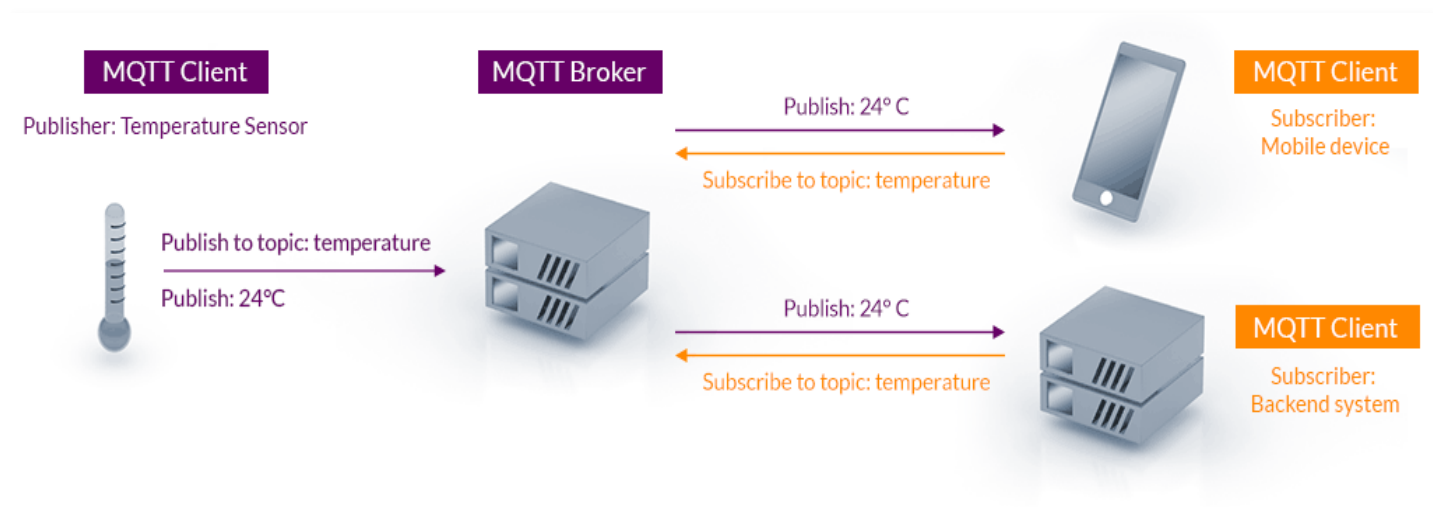
订阅模式

共享订阅

最近更新时间：2025-06-17 14:05:22

发布订阅模型

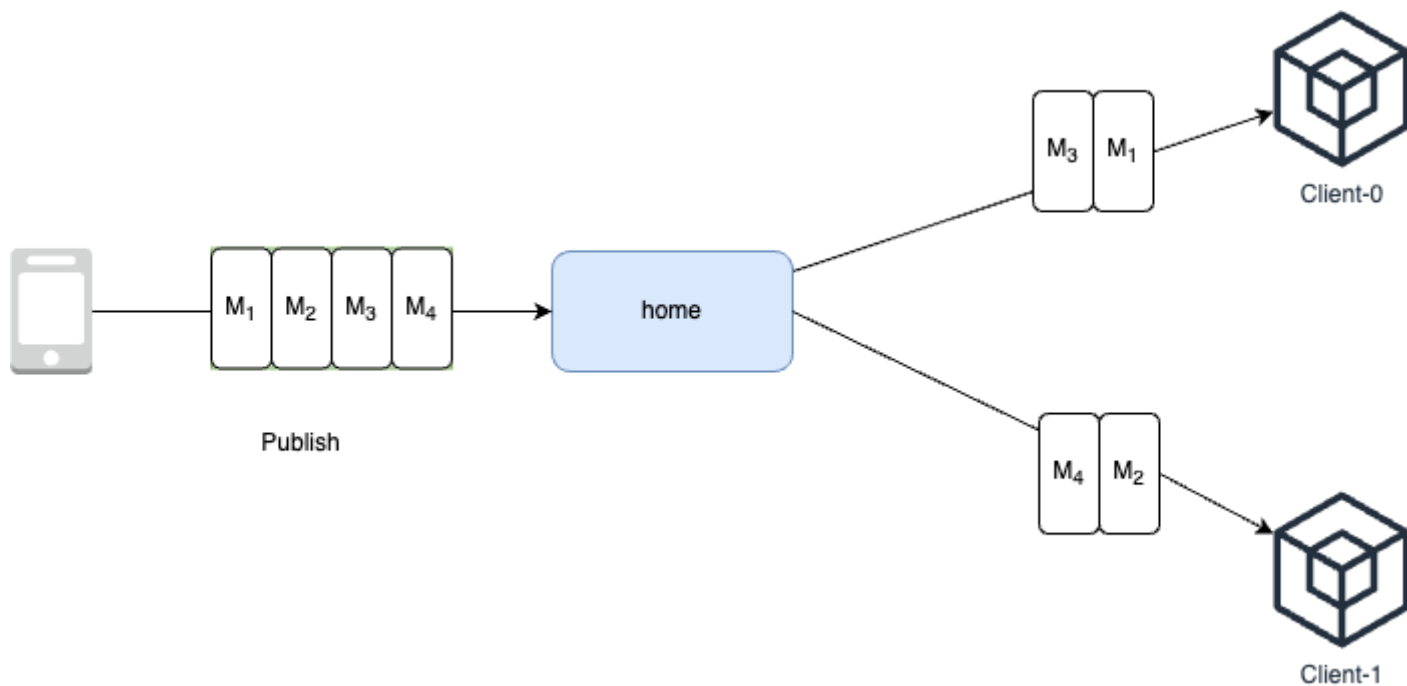
MQTT 协议定义了基于发布订阅（Pub / Sub）的消息模型。例如下图场景中，温度传感器将温度读数发布到 “temperature” 主题，手机和后端业务系统订阅该主题后，都将获得温度传感器发布的全量数据。



什么是共享订阅？

共享订阅是一种负载均衡的消费方式，属于同一个消费组的多个客户端轮转分配订阅的消息。因此，通常也称为消费者负载均衡。

例如下图中，发布者发布了4条消息：M₁、M₂、M₃、M₄。客户端 Client-0 和 Client-1 以共享订阅的方式负载均衡消费订阅的主题。这与默认的 MQTT Pub/Sub 不同，Client-0 仅消费到了 M₁，M₃；Client-1 仅消费到了 M₂，M₄。



MQTT 3.1、3.1.1

MQTT 3.1及3.1.1协议并未定义共享订阅相关内容，腾讯云通过扩展的方式提供了共享订阅的支持。

使用方式

当期望以负载均衡的方式订阅消息时，订阅 topic-filter 以如下方式配置：

```
$share/{ShareName}/{TopicFilter}
```

参数	说明
\$share	字面常量，固定字符串。
{ShareName}	是负载均衡组名称，不能包含 “/” ， “+” ， “#” 。
{TopicFilter}	与 MQTT TopicFilter 要求 和语义相同。

示例代码

```
package org.apache.rocketmq.mqtt.example.quickstart;

import java.util.concurrent.CountDownLatch;
import java.util.concurrent.TimeUnit;
import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
import org.eclipse.paho.client.mqttv3.MqttCallback;
import org.eclipse.paho.client.mqttv3.MqttClient;
```

```
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;
import org.eclipse.paho.client.mqttv3.persist.MemoryPersistence;

public class SharedSubscriptionQuickStart {
    public static void main(String[] args) throws MqttException,
        InterruptedException {
        String serverUri = "tcp://127.0.0.1:1883";
        String clientId = "shared-sub-0";
        try (MqttClient client = new MqttClient(serverUri, clientId, new
            MemoryPersistence())) {
            client.setTimeToWait(3000);
            MqttConnectOptions options = new MqttConnectOptions();
            options.setUserName("YOUR-USERNAME");
            options.setPassword("YOUR-PASSWORD".toCharArray());
            options.setCleanSession(true);
            options.setAutomaticReconnect(true);
            client.connect(options);
            int total = 1;
            CountDownLatch latch = new CountDownLatch(total);

            client.setCallback(new MqttCallback() {
                public void messageArrived(String topic, MqttMessage
message) {
                    System.out.printf("Message arrived, topic=%s, QoS=%d
content=[%s]%n", topic, message.getQos(),
                        new String(message.getPayload()));
                    latch.countDown();
                }

                public void connectionLost(Throwable cause) {
                    System.out.println("connectionLost: " +
cause.getMessage());
                }

                public void deliveryComplete(IMqttDeliveryToken token) {
                    System.out.println("deliveryComplete: " +
token.isComplete());
                }
            });

            // 共享订阅表达式
```

```
// {ShareName} 这里为 Group0
// {topic-filter} 这里为 home/#
String topic = "$share/Group0/home/#";

// Subscribe
client.subscribe(topic, 1);
TimeUnit.HOURS.sleep(1);
client.disconnect();
}
}
}
```

MQTT 5 使用说明

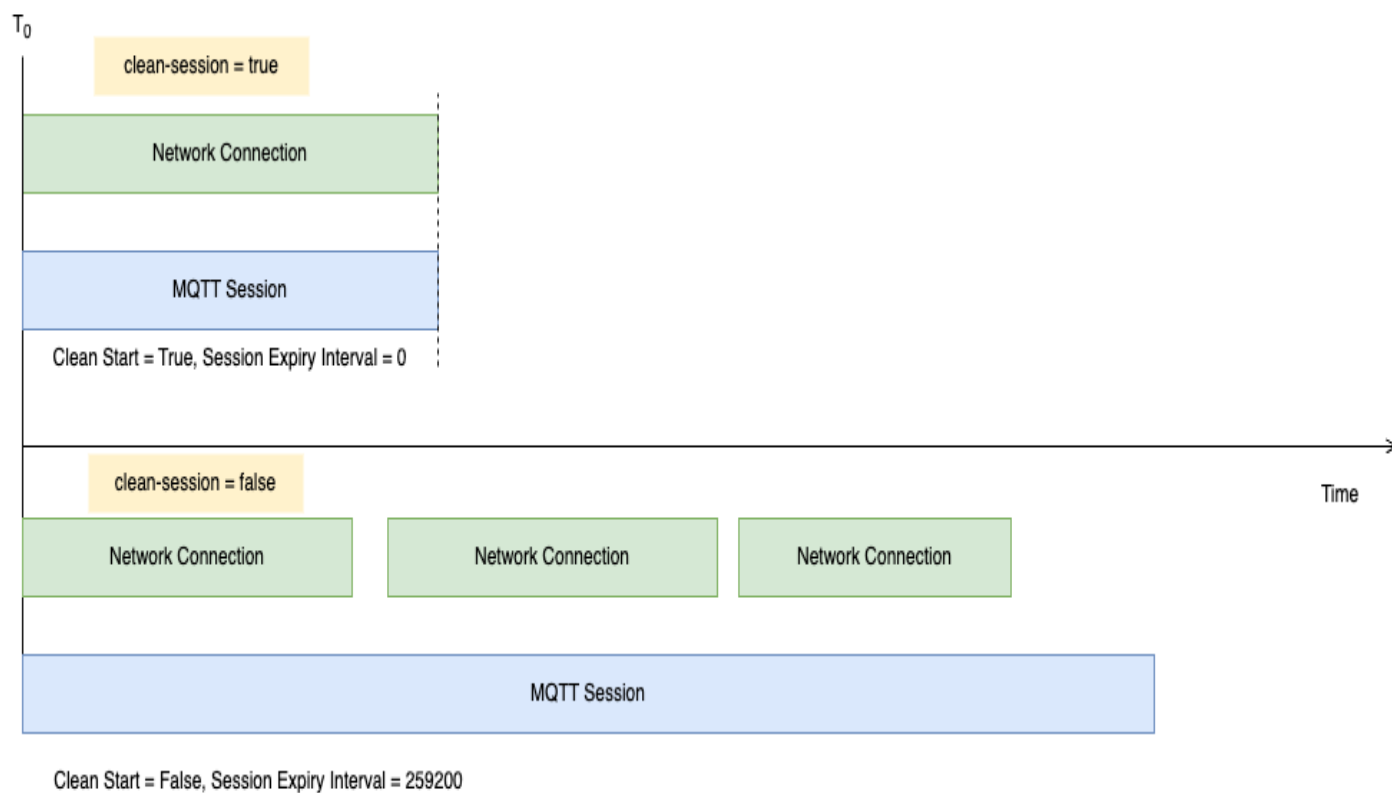
MQTT 5 协议层面明确了 [共享订阅的定义](#)。

共享订阅离线消息保留策略

$\{ShareName\}$ 下如果存在有效的 Session 订阅 $\{TopicFilter\}$ ，符合 $\{TopicFilter\}$ 的离线消息就会保留，订阅者再次上线时，会从上次进度恢复消费。

Session 有效期

MQTT 3.1、3.1.1 通过 clean-session 定义 Session 的生命周期。当 clean-session = true, Session 的生命周期与传输层生命周期一致。当 clean-session = false, Session 与传输层生命周期无关。为避免资源浪费，产品定义传输层断开后，Session 最大存续 3 天。



按照 MQTT 5.0 协议，有以下等价语义：

MQTT 3.1、3.1.1	MQTT 5	
<code>clean-session = true</code>	<code>clean-start = true</code>	<code>session-expiry-interval = 0</code>
<code>clean-session = false</code>	<code>clean-start = false</code>	<code>session-expiry-interval = 259200</code>

MQTT 客户端开发注意事项

最近更新时间：2025-06-26 10:05:42

设置客户端自动重连

无论是通过公网还是通过内网接入 MQTT 集群，传输层连接断开都是常态：移动设备在基站间切换、网络抖动、服务端版本发布等。因此 MQTT 客户端均需要设置连接断开后自动重连，并配置合理的退避策略。

需要在 Connect Options 设置**连接超时时间**、**自动重连**、**最小自动重连间隔**、**最大自动重连间隔**等选项。

Java

```
public class MqttConnectionOptions {  
    ...  
    // Automatic Reconnect  
    private boolean automaticReconnect = false;  
  
    // Time to wait before first automatic reconnection attempt in  
    seconds.  
    private int automaticReconnectMinDelay = 1;  
  
    // Max time to wait for automatic reconnection attempts in seconds.  
    private int automaticReconnectMaxDelay = 120;  
  
    // Connection timeout in seconds  
    private int connectionTimeout = 30;  
  
    private int maxReconnectDelay = 128000;  
    ...  
}
```

C

```
struct MQTTAsync_connectOptions {  
    ...  
  
    /**  
     * The time interval in seconds to allow a connect to complete.  
     */  
    int connectTimeout;
```

```
/**
 * Reconnect automatically in the case of a connection being lost.
0=false, 1=true
 */
int automaticReconnect;

/**
 * The minimum automatic reconnect retry interval in seconds. Doubled
on each failed retry.
 */
int minRetryInterval;

/**
 * The maximum automatic reconnect retry interval in seconds. The
doubling stops here on failed retries.
 */
int maxRetryInterval;
};
```

Paho SDK CleanSession=True 或 CleanStart = True 订阅

Paho SDK Subscriber Client, 如果 Session 配置 CleanSession = True 或者 CleanStart = True , 当发生自动重连后, SDK 不会自动重新订阅, 需要在回调中重新订阅。参考 [Issue 221](#)。

以 Java 回调为例:

```
try (MqttClient client = new MqttClient(serverUri, clientId, new
MemoryPersistence())) {
    MqttConnectOptions options = new MqttConnectOptions();
    options.setCleanSession(true);
    ...
    client.setCallback(new MqttCallbackExtended() {
        @Override
        public void connectComplete(boolean reconnect, String serverURI)
        {
            ...
            try {
                // 必须重新订阅
                client.subscribe(topicFilter, qos);
            } catch (MqttException e) {
                e.printStackTrace();
            }
        }
    })
}
```

```
...  
});  
client.connect(options);  
}
```

QoS 降级

MQTT 服务端向订阅者投递消息的时候，并不总是按照订阅表达式指定的 QoS 投递，而是取 { 发布消息 QoS，服务端支持的最大 QoS，订阅 QoS } 三者最小值。

假设发布消息 M_i 使用的 QoS 为 1、服务端最大支持的 QoS 为 2、订阅使用的 QoS 为 2，投递该消息使用的 QoS 为 1。