



Development Manual

Kingdee Apusic Distributed Message Queue
V2.0.6_for_mqtt

Copyright Notice

All intellectual property rights including software copyrights involved in this document have been legally registered and duly owned by Kingdee Apusic Cloud Computing Co., Ltd. Such rights are protected by Copyright Law of the People's Republic of China, Regulations on Protection of Computer Software, Regulations on Intellectual Property Protection, relevant international copyright conventions, laws, regulations and other intellectual property legislations. Unauthorized use is strictly prohibited.

Disclaimer

All copyright information contained herein is legally owned by Kingdee Apusic Cloud Computing Co., Ltd. and protected by applicable laws. The Company shall bear no liability for any information unrelated to Kingdee Apusic contained in this document. You may view, copy and print this document only within the scope permitted by Copyright Law of the People's Republic of China. No entity or individual may use, modify or republish any part or content of this document without written authorization from Kingdee Apusic Cloud Computing Co., Ltd. Any violation shall constitute infringement, and the Company reserves the right to pursue legal liabilities accordingly.

Updates to this document shall not be notified separately. You may consult the Company regarding issues related to this document. All rights not expressly granted by the Company are reserved.

Trademark Notice

 is a registered trademark filed with the National Intellectual Property Administration of PRC by Kingdee Apusic Cloud Computing Co., Ltd. Shenzhen. Exclusive rights to the registered trademark are legally owned by Kingdee Apusic and protected by law. Without written consent of Kingdee Apusic, no entity or individual shall use, copy, modify, disseminate, transcribe, or bundle and sell any part of this trademark with other products in any manner or for any reason. Kingdee Apusic shall pursue legal liabilities against any infringement of its trademark rights. All other trademarks or registered trademarks mentioned in this document belong to their respective owners.

Contents

- 1 Preface
 - 1.1 Target Audience
 - 1.2 Related Documents
 - 1.3 Technical Support
- 2 Development Preparation
 - 2.1 Connection Parameters
 - 2.2 Client Dependencies
 - 2.2.1 Java
 - 2.2.2 Python
 - 2.2.3 JavaScript/Node.js
 - 2.2.4 Go
- 3 Java Development Example
 - 3.1 Producer
 - 3.2 Consumer
 - 3.3 Will Message
 - 3.4 Session Persistence
- 4 Python Development Example
 - 4.1 Producer
 - 4.2 Consumer
- 5 JavaScript/Node.js Development Example
 - 5.1 Producer
 - 5.2 Consumer
- 6 Go Development Example
 - 6.1 Producer
 - 6.2 Consumer
- 7 Advanced Features
 - 7.1 Retained Messages
 - 7.2 Shared Subscription
 - 7.3 WebSocket Connection
 - 7.4 SSL/TLS Connection
- 8 Best Practices
 - 8.1 ClientID Management

- [8.2 QoS Selection](#)
- [8.3 Session Management](#)
- [8.4 Exception Handling](#)
- [8.5 Performance Optimization](#)

1 Preface

This document is the development guide for the Kingdee Apusic Distributed Message Queue for MQTT (Apusic Distributed Message Queue, ADMQ for MQTT), helping users quickly learn how to develop with ADMQ for MQTT.

1.1 Target Audience

This document is intended for IT business leaders, R&D managers, software project managers, software architects, and operations engineers.

1.2 Related Documents

To learn more about ADMQ for MQTT, please refer to the following ADMQ for MQTT product documentation set:

No.	Document	Description
1	ADMQ for MQTT Quick Start Guide	Briefly introduces how to quickly get started with ADMQ for MQTT.
2	ADMQ for MQTT Installation Manual	Details how to install ADMQ for MQTT on various operating systems and how to start/stop ADMQ for MQTT services.
3	ADMQ for MQTT Message Engine User Manual	Details the usage, configuration, management, and supporting tools of ADMQ for MQTT message engine features.
4	ADMQ for MQTT Console User Manual	Details the usage and operation instructions of the ADMQ for MQTT console.
5	ADMQ for MQTT Development Manual	Details ADMQ for MQTT client application development based on various programming languages.
6	ADMQ for MQTT Migration Manual	Details the migration from MQTT Broker to ADMQ for MQTT.
7	ADMQ for MQTT Operations Manual	Details ADMQ for MQTT monitoring, operations, security hardening, and other operational tasks.
8	ADMQ for MQTT Performance Tuning Manual	Details ADMQ for MQTT performance tuning guidelines.

1.3 Technical Support

ADMQ for MQTT provides comprehensive technical support services. You can obtain technical support through the following channels:

- Website: www.apusic.com
- Phone: 400-855-5800
- Email: support@apusic.com
- Kingdee Cloud Community: <https://vip.kingdee.com/?productId=73&productLineId=14&lang=zh-CN>

When contacting technical support, please provide the following information:

1. Your name
2. Company information and contact details
3. Operating system and its version
4. Product version number
5. Detailed information such as logs and screenshots of the exception or error

2 Development Preparation

2.1 Connection Parameters

Parameter	Description	Example
Broker Address	MQTT Broker address	localhost
Port	MQTT port	1883 (TCP), 8883 (SSL), 8083 (WebSocket)
ClientID	Unique client identifier	device-001
username	Username	device-service
password	Password	your-password

2.2 Client Dependencies

2.2.1 Java

```
<dependency>
  <groupId>org.eclipse.paho</groupId>
  <artifactId>org.eclipse.paho.client.mqttv3</artifactId>
  <version>1.2.5</version>
</dependency>
```

2.2.2 Python

```
pip install paho-mqtt
```

2.2.3 JavaScript/Node.js

```
npm install mqtt
```

2.2.4 Go

```
go get github.com/eclipse/paho.mqtt.golang
```

3 Java Development Example

3.1 Producer

```
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttMessage;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;

public class MqttProducer {
    public static void main(String[] args) throws Exception {
        MqttClient client = new MqttClient("tcp://localhost:1883",
"publisher-001");

        MqttConnectOptions options = new MqttConnectOptions();
        options.setUserName("device-service");
        options.setPassword("your-password".toCharArray());
        options.setCleanSession(true);

        client.connect(options);

        String payload = "
{\"deviceId\":\"D001\", \"temperature\":24.5}";
        MqttMessage message = new MqttMessage(payload.getBytes("UTF-
8"));
        message.setQos(1);
        message.setRetained(false);

        client.publish("sensor/D001/temperature", message);
        System.out.println("Message sent");

        client.disconnect();
        client.close();
    }
}
```

3.2 Consumer

```
import org.eclipse.paho.client.mqttv3.*;

public class MqttConsumer {
    public static void main(String[] args) throws Exception {
        MqttClient client = new MqttClient("tcp://localhost:1883",
"subscriber-001");

        MqttConnectOptions options = new MqttConnectOptions();
        options.setUsername("device-service");
        options.setPassword("your-password".toCharArray());
        options.setCleanSession(true);

        client.setCallback(new MqttCallback() {
            @Override
            public void connectionLost(Throwable cause) {
                System.out.println("Connection lost: " +
cause.getMessage());
            }

            @Override
            public void messageArrived(String topic, MqttMessage
message) throws Exception {
                System.out.println("Received: " + topic + " -> " +
new String(message.getPayload(), "UTF-8"));
            }

            @Override
            public void deliveryComplete(IMqttDeliveryToken token) {
                // Used for QoS 1/2 publish acknowledgment
            }
        });

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

```

        client.subscribe("sensor/+/temperature", 1);

        System.out.println("Subscribed");
    }
}

```

3.3 Will Message

```

MqttConnectOptions options = new MqttConnectOptions();
options.setUserName("device-service");
options.setPassword("your-password".toCharArray());

// Set will message
options.setWill("device/D001/status", "offline".getBytes(), 1,
true);

client.connect(options);

// Send online status after going online
client.publish("device/D001/status", "online".getBytes(), 1, true);

```

3.4 Session Persistence

```

MqttConnectOptions options = new MqttConnectOptions();
options.setUserName("device-service");
options.setPassword("your-password".toCharArray());
options.setCleanSession(false); // Keep session

client.connect(options);
client.subscribe("sensor/D001/temperature", 1);

```

4 Python Development Example

4.1 Producer

```
import paho.mqtt.client as mqtt

client = mqtt.Client(client_id="publisher-001")
client.username_pw_set("device-service", "your-password")

client.connect("localhost", 1883, 60)

payload = '{"deviceId":"D001","temperature":24.5}'
client.publish("sensor/D001/temperature", payload, qos=1,
retain=False)
print("Message sent")

client.disconnect()
```

4.2 Consumer

```
import paho.mqtt.client as mqtt

def on_connect(client, userdata, flags, rc):
    print(f"Connected with result code {rc}")
    client.subscribe("sensor+/temperature", qos=1)

def on_message(client, userdata, msg):
    print(f"Received: {msg.topic} -> {msg.payload.decode('utf-8')}")

client = mqtt.Client(client_id="subscriber-001")
client.username_pw_set("device-service", "your-password")
client.on_connect = on_connect
client.on_message = on_message
```

```
client.connect("localhost", 1883, 60)
client.loop_forever()
```

5 JavaScript/Node.js Development Example

5.1 Producer

```
const mqtt = require('mqtt');

const client = mqtt.connect('mqtt://localhost:1883', {
  clientId: 'publisher-001',
  username: 'device-service',
  password: 'your-password'
});

client.on('connect', () => {
  const payload = JSON.stringify({ deviceId: 'D001', temperature: 24.5 });
  client.publish('sensor/D001/temperature', payload, { qos: 1 }, (err) => {
    if (err) {
      console.error('Publish error:', err);
    } else {
      console.log('Message sent');
    }
    client.end();
  });
});
```

5.2 Consumer

```
const mqtt = require('mqtt');

const client = mqtt.connect('mqtt://localhost:1883', {
  clientId: 'subscriber-001',
  username: 'device-service',
```

```
    password: 'your-password'
  });

client.on('connect', () => {
  client.subscribe('sensor/+/temperature', { qos: 1 }, (err) => {
    if (err) {
      console.error('Subscribe error:', err);
    } else {
      console.log('Subscribed');
    }
  });
});

client.on('message', (topic, message) => {
  console.log(`Received: ${topic} -> ${message.toString()}`);
});
```

6 Go Development Example

6.1 Producer

```
package main

import (
    "fmt"
    "time"
    mqtt "github.com/eclipse/paho.mqtt.golang"
)

func main() {
    opts := mqtt.NewClientOptions().
        AddBroker("tcp://localhost:1883").
        SetClientID("publisher-001").
        SetUsername("device-service").
        SetPassword("your-password")

    client := mqtt.NewClient(opts)
    if token := client.Connect(); token.Wait() && token.Error() !=
nil {
        panic(token.Error())
    }

    payload := `{"deviceId":"D001","temperature":24.5}`
    token := client.Publish("sensor/D001/temperature", 1, false,
payload)
    token.Wait()

    fmt.Println("Message sent")
    client.Disconnect(250)
}
```

6.2 Consumer

```
package main

import (
    "fmt"
    mqtt "github.com/eclipse/paho.mqtt.golang"
)

func main() {
    opts := mqtt.NewClientOptions().
        AddBroker("tcp://localhost:1883").
        SetClientID("subscriber-001").
        SetUsername("device-service").
        SetPassword("your-password")

    client := mqtt.NewClient(opts)
    if token := client.Connect(); token.Wait() && token.Error() !=
nil {
        panic(token.Error())
    }

    if token := client.Subscribe("sensor/+/temperature", 1,
func(client mqtt.Client, msg mqtt.Message) {
        fmt.Printf("Received: %s -> %s\n", msg.Topic(),
string(msg.Payload()))
    }); token.Wait() && token.Error() != nil {
        panic(token.Error())
    }

    select {}
}
```

7 Advanced Features

7.1 Retained Messages

```
MqttMessage message = new MqttMessage(payload.getBytes());
message.setQos(1);
message.setRetained(true); // Set as retained message
client.publish("sensor/D001/status", message);

// Clear retained message
MqttMessage clearMessage = new MqttMessage(new byte[0]);
clearMessage.setRetained(true);
client.publish("sensor/D001/status", clearMessage);
```

7.2 Shared Subscription

MQTT 5.0 shared subscription format:

```
$share/group1/sensor+/temperature
```

Subscribers within the same shared group receive messages in a round-robin fashion, achieving load balancing.

7.3 WebSocket Connection

```
const client = mqtt.connect('ws://localhost:8083/mqtt', {
  clientId: 'web-client-001',
  username: 'device-service',
  password: 'your-password'
});
```

7.4 SSL/TLS Connection

```
MqttClient client = new MqttClient("ssl://localhost:8883",
    "publisher-001");

MqttConnectOptions options = new MqttConnectOptions();
options.setUserName("device-service");
options.setPassword("your-password".toCharArray());

// Configure SSL context
// options.setSocketFactory(sslSocketFactory);

client.connect(options);
```

8 Best Practices

8.1 ClientID Management

1. **Globally unique:** Each client's ClientID must be unique within the cluster
2. **Readability:** Use meaningful prefixes such as `device-` , `app-` , `service-`
3. **Avoid special characters:** Use only letters, numbers, underscores, and hyphens

8.2 QoS Selection

Scenario	Recommended QoS
High-frequency Telemetry data, loss-tolerant	QoS 0
General business messages	QoS 1
Critical control commands requiring exactly-once delivery	QoS 2

8.3 Session Management

1. **Clean Session = true:** Suitable for scenarios that do not require offline messages, lower resource consumption
2. **Clean Session = false:** Suitable for scenarios that need to maintain subscriptions and receive offline messages
3. **MQTT 5.0:** Use Clean Start and Session Expiry Interval to precisely control session behavior

8.4 Exception Handling

1. **Reconnection:** Implement automatic reconnection mechanism after connection loss
2. **Message deduplication:** QoS 1 messages may be duplicated, implement idempotent processing on the business side
3. **Timeout handling:** Set connection timeout and Keepalive interval appropriately

8.5 Performance Optimization

1. **Batch sending:** Reduce frequent small message sends
2. **Set Keepalive appropriately:** Avoid excessively short heartbeat intervals that increase network load
3. **Use shared subscriptions:** Multiple consumers balance message processing
4. **Limit subscription count:** A single client subscribing to too many topics affects performance
5. **Avoid wildcard abuse:** Prefer exact topic matching where possible

UNIFIED SERVICE HOTLINE

4008-555-800

Kingdee Apusic Cloud Computing Co., Ltd. (referred to as “Kingdee Apusic”) was founded in 2000. Its predecessor was “Kingdee Middleware Company.” It is a basic software platform provider under Kingdee Group, a national standards-setting enterprise for cloud computing, and a core software enterprise in the national IT application innovation industry. Kingdee Apusic is an enterprise undertaking the National 863 Key R&D Program and the National Major Science and Technology Project for Core Electronic Components, High-end General-purpose Chips and Fundamental Software, and is also a basic platform provider for the national key projects of “Two Networks, One Website, Four Databases, and Twelve Golden Projects.” Its products are widely used in key industries such as government, military, finance, and energy, and it has cumulatively served more than 100,000 customers.



National standard setter for cloud computing
Basic software enterprise under Kingdee Group
Core enterprise for IT application innovation
Website: www.apusic.com

