Apusic 金蝶天燕

生成时间：2026-05-07


# 开发手册

金蝶Apusic分布式消息队列V2.0.3

## 版权声明

本文档所涉及的软件著作权、版权等知识产权已依法进行了注册，由金蝶天燕云计算股份有限公司合法拥有。受《中华人民共和国著作权法》《计算机软件保护条例》《知识产权保护条例》和相关国际版权条约、法律、法规以及其它知识产权法律和条约的保护。未经授权许可，不得非法使用。

## 免责声明

本文档包含的版权信息由金蝶天燕云计算股份有限公司合法拥有，受法律的保护，金蝶天燕云计算股份有限公司对本文档可能涉及到的非金蝶天燕云计算股份有限公司的信息不承担任何责任。在法律允许的范围内，您可以查阅并仅能够在《中华人民共和国著作权法》规定的合法范围内复制和打印本文档。任何单位和个人未经金蝶天燕云计算股份有限公司书面授权许可，不得使用、修改、再发布本文档的任何部分和内容，否则将被视为侵权，金蝶天燕云计算股份有限公司有依法追究其责任的权利。

本文档如有更新，不另行通知。对本文档中的问题您可向金蝶天燕云计算股份有限公司告知或查询。未经本公司明确授予的任何权利均予保留。

## 商标声明

Apusic是深圳市金蝶天燕云计算股份有限公司向中华人民共和国国家商标局申请注册的注册商标，注册商标专用权由金蝶天燕合法拥有，受法律保护。未经金蝶天燕的书面许可，任何单位及个人不得以任何方式或理由对该商标的任何部分进行使用、复制、修改、传播、抄录或与其它产品捆绑使用销售。凡侵犯金蝶天燕商标权的，金蝶天燕将依法追究其法律责任。本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

# 目录

# 1 概述

本手册用于指导用户编写客户端接入 ADMQ 收发消息，客户端语言支持：Java、C#、Python 和 Go，其中 Java 为主要支持语言。下面针对每种语言的使用进行说明。

ADMQ 是基于 topic 的发布订阅模型，生产者客户端发送消息到某一个 topic 上，则所有订阅该 topic 的消费者组都能收到消息。生产者客户端和 ADMQ 通信过程主要包含以下流程：

- 客户端连接到任意一个 ADMQ 计算节点服务
- 发送请求获取 topic 所在的服务地址，并连接到该服务地址
- 客户端发送消息
- ADMQ 回复消息存储确认的通知
- 客户端继续发送消息

消费者客户端和 ADMQ 通信过程主要包含：

- 客户端连接到任意一个 ADMQ 计算节点服务
- 发送请求获取 topic 所在的服务地址，并连接到该服务地址
- 客户端发送订阅请求
- ADMQ 发送消息
- 客户端接收消息并返回消费确认
- ADMQ 继续发送消息

手册包含三部分：安装部署、管控台使用以及客户端使用说明。

# 2 安装部署

参考文档

# 3 管控台使用说明

参考文档

# 4 MQ 客户端使用说明

## 4.1 准备工作

ADMQ 部署时默认开启了权限认证，客户端接入 ADMQ 发送和接收消息之前首先需要在管控台上创建用户并配置资源权限。具体请参考：**第四章 用户** 和 **第五章 资源管理**

## 4.2 Java 客户端

### 4.2.1 快速入门

创建maven项目，并引入依赖

```
<pulsar.version>2.9.1</pulsar.version>
<dependency>
    <groupId>org.apache.pulsar</groupId>
    <artifactId>pulsar-client-admin</artifactId>
    <version>${pulsar.version}</version>
</dependency>
```

创建客户端

```
// 在管控台用户管理中可获取token内容
String token = "exaJe...";

// 集群的连接地址，可在集群信息中查看。通常为计算节点ip:6650，如果是多个节点则用逗号隔开
String service = "192.168.1.1:6650,192.168.1.2:6650";

ClientBuilder builder = PulsarClient.builder()
    .connectionTimeout(5, TimeUnit.MINUTES);
builder.authentication(AuthenticationFactory.token(token));
builder.serviceUrl("pulsar://" + service);
PulsarClient client = builder.build();
System.out.println("客户端创建成功");
```

创建消费者

```
Consumer<String> consumer = client.newConsumer(Schema.STRING)
    // 主题名称，格式为：persistent://租户名称/命名空间名称/主题名称
    .topic("persistent://apusic/ns01/topic01")
    // 订阅名称
    .subscriptionName("sub-01")
    // 从最早的位置开始消费

.subscriptionInitialPosition(SubscriptionInitialPosition.Earliest)
    // 订阅模式，包含共享、独占、灾备和按Key共享四种模式
    .subscriptionType(SubscriptionType.Exclusive)
    .subscribe();
System.out.println("消费者创建成功");
```

创建生产者

```
Producer<String> producer = client.newProducer(Schema.STRING)
        // 主题名称，格式为：persistent://租户名称/命名空间名称/主题名称
        .topic("persistent://apusic/ns01/topic01")
        .create();
System.out.println("生产者创建成功");
```

生产消息

```
for (int i = 0; i < 10; i++) {
    String data = "admq-test-message " + i;
    MessageId id = producer.newMessage().value(data).send();
    System.out.println("send message: " + data + ", response id: " +
id);
}
producer.close();
```

消费消息

```
for (int i = 0; i < 10; i++) {
    Message<String> msg = consumer.receive();
    System.out.println("receive message: " + msg.getValue() + ", msg
id: " + msg.getMessageId());
    consumer.acknowledge(msg);
}
consumer.close();
client.close();
```

### 4.2.2 Spring Boot Starter 接入

添加依赖

```
<dependency>
  <groupId>io.github.majusko</groupId>
  <artifactId>pulsar-java-spring-boot-starter</artifactId>
  <version>1.1.2</version>
</dependency>
```

添加配置

```
# MQ 服务接入地址
pulsar.service-url=pulsar://localhost:6650
# 命名空间名称
pulsar.namespace=default
# 租户名称
pulsar.tenant=public
# token
pulsar.token-auth-value=43th4398gh340gf34gj349gh304ghryj34fh
```

生产消息

生产者配置

```
@Configuration
public class ProducerConfiguration {
```

```
    @Bean
    public ProducerFactory producerFactory() {
        return new ProducerFactory()
            .addProducer("topic01", String.class);
    }
}
```

创建生产者

```
@Service
class MyProducer {

    @Autowired
    private PulsarTemplate<String> producer;

    void sendHelloWorld() throws PulsarClientException {
        // 此处的主题必须是上边已经注册过的
        producer.send("topic01", "Hello world!");
    }
}
```

消费消息

```
@Service
class MyConsumer {

    @PulsarConsumer(topic="topic01", clazz=String.class)
    void consume(String msg) {
        System.out.println(msg);
    }
}
```

### 4.2.3 监听器方式消费消息

```
// 在管控台用户管理中可获取token内容
String token = "exaJe...";

// 集群的连接地址，可在集群信息中查看。通常为计算节点ip:6650，如果是多个节点则用逗号隔开
String service = "192.168.1.1:6650,192.168.1.2:6650";

ClientBuilder builder = PulsarClient.builder()
    .connectionTimeout(5, TimeUnit.MINUTES);
builder.authentication(AuthenticationFactory.token(token));
builder.serviceUrl("pulsar://" + service);
PulsarClient client = builder.build();
System.out.println("客户端创建成功");

client.newConsumer(Schema.STRING)
    .topic(topic)
    .subscriptionType(SubscriptionType.Shared)
    .subscriptionName(subName)
    .messageListener((c, msg) -> {
        try {
            System.out.println("receive message: " + msg.getValue()
+ ", msg id: " + msg.getMessageId());
            c.acknowledge(msg);
        } catch (Exception e) {
            log.error("", e);
        }
    })
    .subscribe();

Producer<String> producer = client.newProducer(Schema.STRING)
        // 主题名称，格式为：persistent://租户名称/命名空间名称/主题名称
        .topic("persistent://apusic/ns01/topic01")
        .create();
System.out.println("生产者创建成功");
```

```
for (int i = 0; i < 10; i++) {
    String data = "admq-test-message " + i;
    MessageId id = producer.newMessage().value(data).send();
    System.out.println("send message: " + data + ", response id: " +
id);
}
producer.close();
```

#### 4.2.4 异步发送和接收

```
// 创建客户端、生产者和消费者

for (int i = 0; i < 10; i++) {
    String data = "admq-test-message " + i;

producer.newMessage().value(data).sendAsync().whenCompleteAsync((msgId,
ex) -> {
        if (ex != null) {
            // 发送失败，需要业务处理
        } else {
            System.out.println("send message: " + data + ", response
id: " + msgId);
        }
    });
}

for (int i = 0; i < 10; i++) {
    consumer.receiveAsync().whenCompleteAsync((msg, ex) -> {
        if (ex != null) {
        } else {
            System.out.println("receive message: " + msg.getValue()
+ ", msg id: " + msg.getMessageId());
            try {
                consumer.acknowledge(msg);
            } catch (PulsarClientException e) {
```

```
                e.printStackTrace();
            }
        }
    });
}
```

#### 4.2.5 延迟消息

支持指定消息在哪个时刻或者延迟多长时间后被消费者消费。

```
Producer<String> producer = client.newProducer(Schema.STRING)
                // 主题名称，格式为：persistent://租户名称/命名空间名称/主题名称
                .topic("persistent://apusic/ns01/topic01")
                .create();
System.out.println("生产者创建成功");

// 10秒后消费者才能收到消息
producer.newMessage().value("delay message").deliverAfter(10, TimeUnit.SECONDS).send();

// 消费者在指定时刻收到消息
long time = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").parse("2022-06-11 00:00:00").getTime();
producer.newMessage().value("delay message").deliverAt(time).send();
```

#### 4.2.6 消息重试和死信队列

当消息没有被消费者成功消费时，会把消息保存到重试主题中，等待一段时间后会重新发送给消费者，当重试达到一定次数后把消息保存到死信主题中。

```
client.newConsumer(Schema.STRING)
    .topic(topic)
    .subscriptionName("sub01")
    .subscriptionType(SubscriptionType.Shared)
    // 支持重试
```

```
    .enableRetry(true)
    .receiverQueueSize(200)
    .deadLetterPolicy(DeadLetterPolicy.builder()
                      // 最大重试次数，超过后进入死信队列
                      .maxRedeliverCount(5)
                      // 指定消费失败的消息保存的主题名称。不指定的话使用默认
的

.retryLetterTopic("persistent://apsuic/ns01/topic-retry")
                      // 指定死信队列的主题名称。不指定的话使用默认的

.deadLetterTopic("persistent://apusic/ns01/topic-dlq")
                      .build())

.subscriptionInitialPosition(SubscriptionInitialPosition.Earliest)
    .subscribe();
```

## 4.3 C# 客户端

### 添加依赖

引入NuGet程序包：DotPulsar

### 创建客户端连接

```
var token =
"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1xIn0.ybJge7zTfy_RDdAtB3w6nIPDHPT6-
kbB6sNzgPt8sKQ";
var client = PulsarClient.Builder()
    .ServiceUrl(new Uri("pulsar://172.20.140.23:6650"))
    .Authentication(AuthenticationFactory.Token(token))
    .RetryInterval(TimeSpan.FromSeconds(3))
    .Build();
```

### 启动消费者并消费消息

```
var consumer = client.NewConsumer()
    .Topic("persistent://public/default/topic-02")
    .SubscriptionName("sub01")
    .Create();

Console.WriteLine("start consumer ...");
await foreach (var message in consumer.Messages())
{
    Console.WriteLine("Received: " +
Encoding.UTF8.GetString(message.Data.ToArray()));
    // 确认消息
    await consumer.AcknowledgeCumulative(message);
}
```

启动生产者并发送消息

```
var producer = client.NewProducer()
    .Topic("persistent://public/default/topic-02")
    .Create();
for (int i = 0; i < 10; i++)
{
    var dataStr = "c# message test " + i;
    producer.Send(Encoding.UTF8.GetBytes(dataStr));
    Console.WriteLine("send message: " + dataStr);
    Thread.Sleep(1000);
}
```

## 4.4 Go 客户端

添加依赖

github.com/apache/pulsar-client-go/pulsar

创建客户端连接

```
service := flag.String("service", "172.20.140.23:6650", "admq
service address")
token := flag.String("token", "-", "token")

client, err := pulsar.NewClient(pulsar.ClientOptions{
    URL:               "pulsar://" + *service,
    OperationTimeout:  60 * time.Second,
    ConnectionTimeout: 60 * time.Second,
    Authentication:    pulsar.NewAuthenticationToken(*token),
})
```

启动消费者并消费消息

```
func consume(client pulsar.Client, topic *string, subName *string) {
    consumer, err := client.Subscribe(pulsar.ConsumerOptions{
        Topic:                       *topic,
        SubscriptionName:            *subName,
        SubscriptionInitialPosition:
pulsar.SubscriptionPositionLatest,
    })
    if err != nil {
        log.Fatal(err)
    }
    defer consumer.Close()

    ctx := context.Background()

    for i := 0; i < 10; i++ {
        msg, err := consumer.Receive(ctx)

        fmt.Println("Receive mesagge: ", msg.ID(),
string(msg.Payload()))
        if err != nil {
            log.Fatal(err)
        }
```

```
        consumer.Ack(msg)
    }
    consumer.Close()
}
```

启动生产者并发送消息

```
func produce(client pulsar.Client, topic *string, message *string) {
    producer, err := client.CreateProducer(pulsar.ProducerOptions{
        Topic: *topic,
    })
    if err != nil {
        log.Fatal(err)
    }

    ctx := context.Background()

    for i := 0; i < 10; i++ {
        sendData := fmt.Sprintf("%s-%d", *message, i)
        if msgId, err := producer.Send(ctx, &pulsar.ProducerMessage{
            Payload: []byte(sendData),
        }); err != nil {
            log.Fatal(err)
        } else {
            log.Println("Send message: ", msgId, sendData)
        }
    }

    if err != nil {
        log.Fatal(err)
    }
    producer.Close()
}
```

## 4.5 Python 客户端

### 引入依赖

pip3 install pulsar-client==2.9.1

### 创建客户端

```
token =
"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1xIn0.ybJge7zTfy_RDdAtB3w6nIPDHPT6-
kbB6sNzgPt8sKQ";
client = pulsar.Client("pulsar://172.20.140.23:6650",
authentication=pulsar.AuthenticationToken(token))
```

### 创建消费者

```
def consumer():
    consumer =
client.subscribe('persistent://public/default/topic04', 'sub01')

    while True:
        msg = consumer.receive()
        try:
            print("receive message '{}' id='{}'".format(msg.data(),
msg.message_id()))
            consumer.acknowledge(msg)
        except:
            # 会重新收到消息
            consumer.negative_acknowledge(msg)
```

### 创建生产者

```
def producer():
    producer =
client.create_producer('persistent://public/default/topic04')

    for i in range(10):
        data = 'Hello-%d' % i
```

```
        producer.send(data.encode('utf-8'))
        print("send message: '{}'".format(data))
```

全国统一服务热线

4008-555-800

金蝶天燕云计算股份有限公司(简称“金蝶天燕云”)成立于2000年，前身为“金蝶中间件公司”，是金蝶集团旗下新一代软件基础云平台服务商，云计算国家标准制定企业，国家信创产业核心软件企业。金蝶天燕是国家863重点研发计划与核高基重大专项承接企业，也是“两网一站四库十二金”国家重点工程的基础平台提供商，产品广泛应用于政府、军工、金融、能源等关键行业，累计服务客户总数超过10万家。

Apusic 金蝶天燕

云计算国家标准制定企业

金蝶集团旗下基础软件企业

信息技术应用创新核心企业

官网：www.apusic.com