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V2.0.5

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
- 2 ADMQ 客户端使用
  - 2.1 准备工作
  - 2.2 Java 客户端
    - 2.2.1 快速入门
    - 2.2.2 Spring Boot Starter 接入
    - 2.2.3 监听器方式消费消息
    - 2.2.4 异步发送和接收
    - 2.2.5 延迟消息
    - 2.2.6 消息重试和死信队列
  - 2.3 C# 客户端
  - 2.4 Go 客户端
  - 2.5 Python 客户端
- 3 RocketMQ客户端
  - 3.1 普通消息
    - 3.1.1 代码示例
  - 3.2 异步消息
    - 3.2.1 代码示例
  - 3.3 顺序消息
    - 3.3.1 代码示例
  - 3.4 死信队列
    - 3.4.1 代码示例
  - 3.5 消息重试
    - 3.5.1 代码示例
  - 3.6 广播消息
    - 3.6.1 代码示例
  - 3.7 批量消息
    - 3.7.1 代码示例
  - 3.8 事务消息
    - 3.8.1 流程介绍
    - 3.8.2 生产消息规则
    - 3.8.3 消费消息规则

- 3.8.4 代码示例
- 3.9 定时消息
  - 3.9.1 代码示例
- 3.10 延迟消息
  - 3.10.1 代码示例
- 4 RabbitMQ API
  - 4.1 Java客户端
    - 4.1.1 Connection API
    - 4.1.2 Channel API
  - 4.2 Consumer API
- 5 JMS客户端
- 6 流Java客户端
- 7 管理HTTP API
- 8 Kafka APIS
  - 8.1 Producer API
  - 8.2 Consumer API
  - 8.3 Streams API
    - 8.3.1 KStream API
    - 8.3.2 KGroupedStream API
    - 8.3.3 TimeWindowedKStream API
    - 8.3.4 SessionWindowedKStream API
    - 8.3.5 KTable API
  - 8.4 Connect API
    - 8.4.1 Connector API
      - 8.4.1.1 SourceConnector API
      - 8.4.1.2 SinkConnector API
    - 8.4.2 Task API
      - 8.4.2.1 SourceTask API
      - 8.4.2.2 SinkTask API
  - 8.5 Admin API

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

2 ADMQ 客户端使用

2.1 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**

2.2 Java 客户端

2.2.1 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();
```

2.2.2 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);
    }
}
```

2.2.3 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();
```

2.2.4 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();
            }
        }
    });
}
```

2.2.5 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();
```

2.2.6 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();
```

2.3 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);
}
```

2.4 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()
}
```

2.5 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))
```

ADMQ通过开启插件支持用户使用原生RocketMQ客户端、RabbitMQ客户端、Kafka客户端接收和发送消息，下面记录使用这些客户端时的功能支持情况。 插件操作请参考用户手册

3 RocketMQ客户端

RocketMQ是阿里开源的消息队列，详细介绍可参考：https://help.aliyun.com/product/29530.html

功能	是否支持	说明
普通消息	是	
异步消息	是	
顺序消息	是	
消息过滤	否	
死信队列	是	
消息重试	是	
广播消息	是	
批量消息	是	
事务消息	否	
定时消息	是	
延迟消息	是	
ACL鉴权	是	

3.1 普通消息

可以使用rocketmq客户端来创建消息的生产者、消费者，并进行普通消息的生产与消费。普通消息是指无特性的消息，区别于有特性的定时和延迟消息、顺序消息。

3.1.1 代码示例

```
public static void main(String[] args)throws Exception{
        // 实例化消息生产者Producer
         AclClientRPCHook aclClientRPCHook = new AclClientRPCHook(new Se
         DefaultMQProducer producer = new DefaultMQProducer("namespace",
         // 设置NameServer的地址
         producer.setNamesrvAddr("nameserver");
```

```
        // 启动Producer实例
        producer.start();
        for (int i = 0; i < 10; i++) {
            // 创建消息，并指定Topic，Tag和消息体
            Message msg = new Message("topic" /* Topic */,
                    "tag" /* Tag */,
                    ("Hello RocketMQ " + i).getBytes(RemotingHelper.DEF
            );
            // 发送消息到一个Broker
            SendResult sendResult = producer.send(msg);
            // 通过sendResult返回消息是否成功送达
            System.out.printf("%s%n",
(System.currentTimeMillis()/1000)+":"+sendResult);
        }
        // 如果不再发送消息，关闭Producer实例。
        producer.shutdown();
    }
```

3.2 异步消息

异步发送是指发送方发出一条消息后，不等服务端返回响应，接着发送下一条消息的通讯方式。rocketmq异步发送，需要您实现异步发送回调接口（SendCallback）。消息发送方在发送了一条消息后，不需要等待服务端响应即可发送第二条消息。发送方通过回调接口接收服务端响应，并处理响应结果。

3.2.1 代码示例

```
private void asyncSend(DefaultMQProducer producer, Message message,
String messageValue,
                       LongAdder leftCount) throws Exception {
        producer.send(message, new SendCallback() {
            @Override
            public void onSuccess(SendResult sendResult) {
                log.info("[{}] send message: {}, tag: {}, key: {},
result: {}", topic, messageValue,
                        tags == null ? "-" : tags,
                        keys == null ? "-" : keys,
```

```
                        sendResult);
                leftCount.decrement();
            }

            @Override
            public void onException(Throwable e) {
                log.error("Send failed.", e);
                leftCount.decrement();
            }
        });
    }
```

3.3 顺序消息

顺序消息是RocketMQ提供的一种对消息发送和消费顺序有严格要求的消息。对于一个指定的Topic，消息严格按照先进先出（FIFO）的原则进行消息发布和消费，即先发布的消息先消费，后发布的消息后消费。 现在只支持分区顺序消息，对于指定的一个Topic，所有消息根据Sharding Key进行区块分区，同一个分区内的消息按照严格的先进先出（FIFO）原则进行发布和消费。同一分区内的消息保证顺序，不同分区之间的消息顺序不做要求。

3.3.1 代码示例

```
public class Producer {
    public static void main(String[] args) throws
UnsupportedEncodingException {
        try {
            DefaultMQProducer producer = new
DefaultMQProducer("please_rename_unique_group_name");
            producer.start();

            String[] tags = new String[] {"TagA", "TagB", "TagC",
"TagD", "TagE"};
            for (int i = 0; i < 100; i++) {
                int orderId = i % 10;
                Message msg =
                    new Message("TopicTest", tags[i % tags.length],
"KEY" + i,
```

```
                        ("Hello RocketMQ " +
i).getBytes(RemotingHelper.DEFAULT_CHARSET));
                SendResult sendResult = producer.send(msg, new
MessageQueueSelector() {
                    @Override
                    public MessageQueue select(List<MessageQueue>
mqs, Message msg, Object arg) {
                        Integer id = (Integer) arg;
                        int index = id % mqs.size();
                        return mqs.get(index);
                    }
                }, orderId);

                System.out.printf("%s%n", sendResult);
            }

            producer.shutdown();
        } catch (MQClientException | RemotingException |
MQBrokerException | InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

3.4 死信队列

当一条消息初次消费失败，RocketMQ会自动进行消息重试，达到最大重试次数后，若消费依然失败，则表明消费者在正常情况下无法正确地消费该消息。此时，RocketMQ不会立刻将消息丢弃，而是将其发送到该消费者对应的特殊队列中。这种正常情况下无法被消费的消息称为死信消息（Dead-Letter Message），存储死信消息的特殊队列称为死信队列（Dead-Letter Queue）。

3.4.1 代码示例

```
  DefaultMQPushConsumer consumer = new
DefaultMQPushConsumer("group_name");
  consumer.setNamesrvAddr("localhost:9876");
```

```
  consumer.setMessageModel(MessageModel.CLUSTERING);
  consumer.subscribe("topic_name", "*");
  consumer.setConsumeThreadMax(1);
  consumer.setConsumeThreadMin(1);
  consumer.setInstanceName("consumer");

consumer.setConsumeFromWhere(ConsumeFromWhere.CONSUME_FROM_LAST_OFFSET)
  consumer.setConsumeTimeout(60);
  consumer.setAllocateMessageQueueStrategy(new
AllocateMessageQueueAveragely());
  consumer.setMessageListener(new MessageListenerConcurrently() {
    public ConsumeConcurrentlyStatus consumeMessage(List msgs,
ConsumeConcurrentlyContext context) {
      // business logic
      return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
    }
  });
  consumer.start();

  // 死信队列配制
  String groupName = "your_group_name";
  String topic = "your_topic_name";
  String origTopic = "your_topic_name";
  String subExpression = "your_sub_expression";
  int maxRetryTimes = 5;
  long retryInterval = 60L;

  String dlqTopic = MixAll.getRetryTopic(groupName);
  String dlqRealTopic = String.format("%s%s", dlqTopic,
System.currentTimeMillis());
  String dlqSubExpression = String.format("%s&&%s", subExpression,
MessageConst.PROPERTY_RETRY_TOPIC + "==" + dlqTopic);
  CreateTopicKey topicKey = new CreateTopicKey(dlqTopic,
org.apache.rocketmq.common.protocol.RequestCode.UPDATE_AND_CREATE_TOPIC
  TopicConfig topicConfig = new TopicConfig(dlqTopic);
  groupConfig.getTopicConfigTable().put(dlqTopic, topicConfig);
```

```
  table.put(topicKey, topicConfig.buildTopicSetting());

  producer.send(new Message(origTopic, dlqSubExpression, "", new
byte[] {1, 2, 3}), new SendCallback() {
    public void onSuccess(SendResult sendResult) {
      // do something
    }
    public void onException(Throwable e) {
      // do something
    }
  });
```

3.5 消息重试

RocketMQ消息收发过程中，若Consumer消费某条消息失败，则RocketMQ会在重试间隔时间后，将消息重新投递给Consumer消费，若达到最大重试次数后消息还没有成功被消费，则消息将被投递至死信队列。

- **重试间隔**：消息消费失败后再次被消息队列RocketMQ版投递给Consumer消费的间隔时间。
- **最大重试次数**：消息消费失败后，可被消息队列RocketMQ版重复投递的最大次数。

3.5.1 代码示例

```
public class RetryConsumer {
    public static void main(String[] args) throws MQClientException
{
        DefaultMQPushConsumer consumer = new
DefaultMQPushConsumer("group1");
        consumer.subscribe("TopicA", "*");
        consumer.setNamesrvAddr("192.168.42.112:9876");

consumer.setConsumeFromWhere(ConsumeFromWhere.CONSUME_FROM_LAST_OFFSET)
每次从最后一次消费的偏移量开始消费
        //重试次数(默认15次，先会进入%RETRY%group1中)
        consumer.setMaxReconsumeTimes(3);
        consumer.registerMessageListener(new
MessageListenerConcurrently() {
            public ConsumeConcurrentlyStatus
```

```
consumeMessage(List<MessageExt> list, ConsumeConcurrentlyContext
consumeConcurrentlyContext) {
                int count = list.get(0).getReconsumeTimes();
                System.out.println("重试第"+count+"次");
                System.out.println("queueID:" +
list.get(0).getQueueId() + ",ThreadName:" +
Thread.currentThread().getName()
                        + ",Messages:" + new
String(list.get(0).getBody()));
                //这里返回RECONSUME_LATER，这个消息会重新发送到Broker中的
RETRY topic
                return ConsumeConcurrentlyStatus.RECONSUME_LATER;
            }
        });
        consumer.start();
        System.out.println("ConsumerPartOrder Started.");
    }
}
```

3.6 广播消息

当使用广播消费模式时，RocketMQ会将每条消息推送给集群内所有的消费者，保证消息至少被每个消费者消费一次。

3.6.1 代码示例

```
//设置广播消费模式
pushConsumer.setMessageModel(MessageModel.BROADCASTING);
// 订阅topic
pushConsumer.subscribe(topic_name, "*");
// 注册回调实现类来处理从broker拉取回来的消息
pushConsumer.registerMessageListener((MessageListenerConcurrently)
(msgs, context) -> {
    // 消息处理逻辑
    System.out.printf("%s Receive New Messages: %s %n",
Thread.currentThread().getName(), msgs);
    // 标记该消息已经被成功消费，根据消费情况，返回处理状态
```

```
    return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
});
// 启动消费者实例
pushConsumer.start();
```

3.7 批量消息

如需提高消息的处理效率，或降低下游资源的API调用频率，可以使用批量发送消息。

3.7.1 代码示例

```
public class SimpleBatchProducer {

    public static void main(String[] args) throws Exception {
        DefaultMQProducer producer = new
DefaultMQProducer("BatchProducerGroupName");
        producer.start();

        //If you just send messages of no more than 1MiB at a time,
it is easy to use batch
        //Messages of the same batch should have: same topic, same
waitStoreMsgOK and no schedule support
        String topic = "BatchTest";
        List<Message> messages = new ArrayList<>();
        messages.add(new Message(topic, "Tag", "OrderID001", "Hello
world 0".getBytes()));
        messages.add(new Message(topic, "Tag", "OrderID002", "Hello
world 1".getBytes()));
        messages.add(new Message(topic, "Tag", "OrderID003", "Hello
world 2".getBytes()));

        producer.send(messages);
    }
}
```

3.8 事务消息

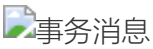
消息队列RocketMQ版分布式事务消息不仅可以实现应用之间的解耦，又能保证数据的最终一致性。同时，传统的大事务可以被拆分为小事务，不仅能提升效率，还不会因为某一个关联应用的不可用导致整体回滚，从而最大限度保证核心系统的可用性。在极端情况下，如果关联的某一个应用始终无法处理成功，也只需对当前应用进行补偿或数据订正处理，而无需对整体业务进行回滚。

事务消息：消息队列RocketMQ版提供类似XA或Open XA的分布式事务功能，通过消息队列RocketMQ版事务消息能达到分布式事务的最终一致。

半事务消息：暂不能投递的消息，生产者已经成功地将消息发送到了消息队列RocketMQ版服务端，但是消息队列RocketMQ版服务端未收到生产者对该消息的二次确认，此时该消息被标记成“暂不能投递”状态，处于该种状态下的消息即半事务消息。

消息回查：由于网络闪断、生产者应用重启等原因，导致某条事务消息的二次确认丢失，消息队列RocketMQ版服务端通过扫描发现某条消息长期处于“半事务消息”时，需要主动向消息生产者询问该消息的最终状态（Commit或是Rollback），该询问过程即消息回查。

3.8.1 流程介绍


事务消息

事务消息发送步骤如下：

1. 生产者将半事务消息发送至消息队列RocketMQ版服务端。
2. 消息队列RocketMQ版服务端将消息持久化成功之后，向生产者返回Ack确认消息已经发送成功，此时消息为半事务消息。
3. 生产者开始执行本地事务逻辑。
4. 生产者根据本地事务执行结果向服务端提交二次确认结果（Commit或是Rollback），服务端收到确认结果后处理逻辑如下：
    - 二次确认结果为Commit：服务端将半事务消息标记为可投递，并投递给消费者。
    - 二次确认结果为Rollback：服务端将回滚事务，不会将半事务消息投递给消费者。
5. 在断网或者是生产者应用重启的特殊情况下，若服务端未收到发送者提交的二次确认结果，或服务端收到的二次确认结果为Unknown未知状态，经过固定时间后，服务端将对消息生产者即生产者集群中任一生产者实例发起消息回查。

事务消息回查步骤如下：

1. 生产者收到消息回查后，需要检查对应消息的本地事务执行的最终结果。
2. 生产者根据检查得到的本地事务的最终状态再次提交二次确认，服务端仍按照步骤4对半事务消息进行处理。

3.8.2 生产消息规则

事务消息发送完成本地事务后，可在execute方法中返回以下三种状态：

- `TransactionStatus.CommitTransaction`：提交事务，允许消费者消费该消息。
- `TransactionStatus.RollbackTransaction`：回滚事务，消息将被丢弃不允许消费。
- `TransactionStatus.Unknow`：暂时无法判断状态，等待固定时间以后消息队列RocketMQ版服务端根据回查规则向生产者进行消息回查。
- 通过`ONSFactory.createTransactionProducer`创建事务消息的Producer时必须指定`LocalTransactionChecker`的实现类，处理异常情况下事务消息的回查。
- 回查规则：本地事务执行完成后，若服务端收到的本地事务返回状态为TransactionStatus.Unknow，或生产者应用退出导致本地事务未提交任何状态。则服务端会向消息生产者发起事务回查，第一次回查后仍未获取到事务状态，则之后每隔一段时间会再次回查。
  - 回查间隔时间：系统默认每隔30秒发起一次定时任务，对未提交的半事务消息进行回查，共持续12小时。
  - 第一次消息回查最快时间：该参数支持自定义设置。若指定消息未达到设置的最快回查时间前，系统默认每隔30秒一次的回查任务不会检查该消息。

    以Java为例，以下设置表示：第一次回查的最快时间为60秒。

    ```
    Message message = new Message();
    message.putUserProperties(PropertyKeyConst.CheckImmunityTime
    ```

3.8.3 消费消息规则

事务消息的Group ID不能与其他类型消息的Group ID共用。与其他类型的消息不同，事务消息有回查机制，回查时消息队列RocketMQ版服务端会根据Group ID去查询生产者客户端。

3.8.4 代码示例

```
public class TransactionProducer {
    public static void main(String[] args) throws MQClientException,
InterruptedException {
        TransactionListener transactionListener = new
```

```
TransactionListenerImpl();
        TransactionMQProducer producer = new
TransactionMQProducer("please_rename_unique_group_name");
        ExecutorService executorService = new ThreadPoolExecutor(2,
5, 100, TimeUnit.SECONDS, new ArrayBlockingQueue<Runnable>(2000),
new ThreadFactory() {
            @Override
            public Thread newThread(Runnable r) {
                Thread thread = new Thread(r);
                thread.setName("client-transaction-msg-check-
thread");
                return thread;
            }
        });

        producer.setExecutorService(executorService);
        producer.setTransactionListener(transactionListener);
        producer.start();

        String[] tags = new String[] {"TagA", "TagB", "TagC",
"TagD", "TagE"};
        for (int i = 0; i < 10; i++) {
            try {
                Message msg =
                    new Message("TopicTest", tags[i % tags.length],
"KEY" + i,
                        ("Hello RocketMQ " +
i).getBytes(RemotingHelper.DEFAULT_CHARSET));
                SendResult sendResult =
producer.sendMessageInTransaction(msg, null);
                System.out.printf("%s%n", sendResult);

                Thread.sleep(10);
            } catch (MQClientException |
UnsupportedEncodingException e) {
                e.printStackTrace();
```

```
            }
        }

        for (int i = 0; i < 100000; i++) {
            Thread.sleep(1000);
        }
        producer.shutdown();
    }

    static class TransactionListenerImpl implements
TransactionListener {
        private AtomicInteger transactionIndex = new
AtomicInteger(0);

        private ConcurrentHashMap<String, Integer> localTrans = new
ConcurrentHashMap<>();

        @Override
        public LocalTransactionState executeLocalTransaction(Message
msg, Object arg) {
            int value = transactionIndex.getAndIncrement();
            int status = value % 3;
            localTrans.put(msg.getTransactionId(), status);
            return LocalTransactionState.UNKNOW;
        }

        @Override
        public LocalTransactionState
checkLocalTransaction(MessageExt msg) {
            Integer status = localTrans.get(msg.getTransactionId());
            if (null != status) {
                switch (status) {
                    case 0:
                        return LocalTransactionState.UNKNOW;
                    case 1:
                        return LocalTransactionState.COMMIT_MESSAGE;
```

```
                    case 2:
                        return
LocalTransactionState.ROLLBACK_MESSAGE;
                    default:
                        return LocalTransactionState.COMMIT_MESSAGE;
                }
            }
            return LocalTransactionState.COMMIT_MESSAGE;
        }
    }
}
```

3.9 定时消息

Producer将消息发送到消息队列RocketMQ版服务端，但并不期望立马投递这条消息，而是推迟到在当前时间点之后的某一个时间投递到Consumer进行消费，该消息即定时消息。

3.9.1 代码示例

```
public static void main(String[] args) throws Exception {
        //DefaultMQProducer producer = new
DefaultMQProducer("namespace","groupname",Global.getAclRPCHook());
        DefaultMQProducer producer = new DefaultMQProducer("test2");
        producer.setNamesrvAddr("nameserver");
        producer.start();
        final byte[] bytes = "this is cutomer 2m2s message"
.getBytes(StandardCharsets.UTF_8);
       Message message = new Message("topic", "*", bytes);
        message.putUserProperty("SPECIFY_DELAY_TIME_SECONDS",
"11");//延迟多少秒
        message.putUserProperty("SPECIFY_DELAY_TIME_TIMESTAMP",
"1663641660");
        producer.sendOneway(message);
        producer.shutdown();
    }
```

3.10 延迟消息

Producer将消息发送到消息队列RocketMQ版服务端，但并不期望立马投递这条消息，而是延迟一定时间后才投递到Consumer进行消费，该消息即延时消息。

3.10.1 代码示例

```
public class ScheduledMessageProducer {
    public static void main(String[] args) throws Exception {
        // Instantiate a producer to send scheduled messages
        DefaultMQProducer producer = new
DefaultMQProducer("ExampleProducerGroup");
        // Launch producer
        producer.start();
        int totalMessagesToSend = 100;
        for (int i = 0; i < totalMessagesToSend; i++) {
            Message message = new Message("TestTopic", ("Hello
scheduled message " + i).getBytes());
            // This message will be delivered to consumer 10 seconds
later.
            message.setDelayTimeLevel(3);
            // Send the message
            producer.send(message);
        }

        // Shutdown producer after use.
        producer.shutdown();
    }
}
```

4 RabbitMQ API

客户端API公开了AMQP 0-9-1协议模型中的关键实体，并提供了额外的抽象以方便使用。

RabbitMQ Java客户端使用com.rabbitmq.client作为顶层包。关键类和接口是:

1. Java客户端
    - Connection：表示AMQP 0-9-1连接。
    - Channel：表示一个AMQP 0-9-1通道，提供了大部分的操作(协议方法)。
    - Consumer：表示消息消费者。

      Connection用于打开通道，注册连接生命周期事件处理程序，并关闭不再需要的连接。连接通过ConnectionFactory实例化，这是您配置各种连接设置的方式，例如vhost或用户名。协议操作通过通道接口进行。

2. JMS客户端

   用于RabbitMQ的JMS客户端在RabbitMQ Java客户端之上实现了JMS规范，从而允许新的和现有的JMS应用程序连接到RabbitMQ协议处理器。

   该库支持2.7.0版的JMS 1.1和2.0。插件和JMS客户机应该一起工作和使用。

    - Connection Factory：ConnectionFactory对象封装了一组由管理员定义的连接配置参数。客户机使用它创建与JMS提供程序的连接。

      ConnectionFactory对象是一个JMS管理对象，支持并发使用。

    - Connection：Connection对象是客户机到其JMS提供程序的活动连接。它通常在Java虚拟机(JVM)之外分配提供者资源。

      连接支持并发使用。

    - Session：Session对象是用于生产和消费消息的单线程上下文。尽管它可以在Java虚拟机(JVM)之外分配提供者资源，但它被认为是轻量级JMS对象。
    - Message：Message接口是所有JMS消息的根接口。它定义了用于所有消息的消息头和确认方法。
    - Message Producer：客户端使用MessageProducer对象将消息发送到目的地。通过将Destination对象传递给会话提供的消息生成器创建方法，可以创建MessageProducer对象。

      MessageProducer是所有消息生产者的父接口。

- Message Consumer：客户端使用MessageConsumer对象从目的地接收消息。通过将Destination对象传递给会话提供的消息使用者创建方法，可以创建MessageConsumer对象。

  MessageConsumer是所有消息使用者的父接口。

- Destination：Destination对象封装特定于提供程序的地址。

3. 流Java客户端

   RabbitMQ Stream Java Client是一个与RabbitMQ Stream Plugin通信的Java库。它允许创建和删除流，以及向这些流发布和从这些流消费。

4. 管理 HTTP API

   RabbitMQ管理插件提供了一个基于HTTP的API，用于管理和监控RabbitMQ节点和集群。

由于目前生产环境大多数使用的是RabbitMQ为3.10.0及以上版本，3.11.0及以上版本生产环境使用并不多，下面API支持情况以RabbitMQ Java Client最稳定的版本5.16.0进行说明。

4.1 Java客户端

4.1.1 Connection API

API的核心类是Connection，表示AMQP 0-9-1连接。

修饰符和类型	方法和描述	是否支持
void	abort() Abort this connection and all its channels with the `AMQP.REPLY_SUCCESS` close code and message 'OK'.	支持
void	abort(int timeout) Abort this connection and all its channels with the [`AMQP.REPLY_SUCCESS` close code and message 'OK'.	支持
void	abort(int closeCode, String closeMessage) Abort this connection and all its channels.	支持
void	abort(int closeCode, String closeMessage, int timeout) Abort this connection and all its channels.	支持
BlockedListener	addBlockedListener(BlockedCallback blockedCallback, UnblockedCallback unblockedCallback) Add a lambda-based `BlockedListener`.	支持

void	addBlockedListener(BlockedListener listener) Add a `BlockedListener`.	支持
void	clearBlockedListeners() Remove all `BlockedListener`s.	支持
void	close() Close this connection and all its channels with the `AMQP.REPLY_SUCCESS`close code and message 'OK'.	支持
void	close(int timeout) Close this connection and all its channels with the `AMQP.REPLY_SUCCESS` close code and message 'OK'.	支持
void	close(int closeCode, String closeMessage) Close this connection and all its channels.	支持
void	close(int closeCode, String closeMessage, int timeout) Close this connection and all its channels.	支持
Channel	createChannel() Create a new channel, using an internally allocated channel number.	支持
Channel	createChannel(int channelNumber) Create a new channel, using the specified channel number if possible.	支持
InetAddress	getAddress() Retrieve the host.	支持
int	getChannelMax() Get the negotiated maximum channel number.	支持
Map<String,Object>	getClientProperties() Get a copy of the map of client properties sent to the server	支持
String	getClientProvidedName() Returns client-provided connection name, if any.	支持
ExceptionHandler	getExceptionHandler() Get the exception handler.	支持
int	getFrameMax() Get the negotiated maximum frame size.	支持
int	getHeartbeat() Get the negotiated heartbeat interval.	支持

String	getId() Returns a unique ID for this connection.	支持
int	getPort() Retrieve the port number.	支持
Map<String,Object>	getServerProperties() Retrieve the server properties.	支持
boolean	removeBlockedListener(BlockedListener listener) Remove a `BlockedListener`.	支持
void	setId(String id) Sets a unique ID for this connection.	支持

4.1.2 Channel API

核心API类是Channel，表示AMQP 0-9-1通道。

应该避免在线程之间共享Channel实例。应用程序应该每个线程使用一个通道，而不是跨多个线程共享同一个通道。

修饰符和类型	方法和描述	是否支持
void	abort() Abort this channel with the `AMQP.REPLY_SUCCESS` close code and message 'OK'.	支持
void	abort(int closeCode, String closeMessage) Abort this channel.	支持
ConfirmListener	addConfirmListener(ConfirmCallback ackCallback, ConfirmCallback nackCallback) Add a lambda-based `ConfirmListener`.	支持
void	addConfirmListener(ConfirmListener listener) Add a `ConfirmListener`.	支持
ReturnListener	addReturnListener(ReturnCallback returnCallback) Add a lambda-based `ReturnListener`.	支持
void	addReturnListener(ReturnListener listener) Add a `ReturnListener`.	支持

void	basicAck(long deliveryTag, boolean multiple) Acknowledge one or several received messages.	支持
void	basicCancel(String consumerTag) Cancel a consumer.	支持
String	basicConsume(String queue, boolean autoAck, Consumer callback) Start a non-nolocal, non-exclusive consumer, with a server-generated consumerTag.	支持
String	basicConsume(String queue, boolean autoAck, DeliverCallback deliverCallback, CancelCallback cancelCallback) Start a non-nolocal, non-exclusive consumer, with a server-generated consumerTag.	支持
String	basicConsume(String queue, boolean autoAck, DeliverCallback deliverCallback, CancelCallback cancelCallback, ConsumerShutdownSignalCallback shutdownSignalCallback) Start a non-nolocal, non-exclusive consumer, with a server-generated consumerTag.	支持
String	basicConsume(String queue, boolean autoAck, DeliverCallback deliverCallback, ConsumerShutdownSignalCallback shutdownSignalCallback) Start a non-nolocal, non-exclusive consumer, with a server-generated consumerTag.	支持
String	basicConsume(String queue, boolean autoAck, String consumerTag, boolean noLocal, boolean exclusive, Map<String,Object> arguments, Consumer callback) Start a consumer.	支持
String	basicConsume(String queue, boolean autoAck, String consumerTag, boolean noLocal, boolean exclusive, Map<String,Object> arguments, DeliverCallback deliverCallback, CancelCallback cancelCallback) Start a consumer.	支持
String	basicConsume(String queue, boolean autoAck, String consumerTag, boolean noLocal, boolean exclusive, Map<String,Object> arguments, DeliverCallback deliverCallback, CancelCallback cancelCallback, ConsumerShutdownSignalCallback shutdownSignalCallback) Start a consumer.	支持
String	basicConsume(String queue, boolean autoAck, String consumerTag, boolean noLocal, boolean exclusive, Map<String,Object> arguments, DeliverCallback deliverCallback, ConsumerShutdownSignalCallback	支持

	shutdownSignalCallback) Start a consumer.	
String	basicConsume(String queue, boolean autoAck, String consumerTag, Consumer callback) Start a non-nolocal, non-exclusive consumer.	支持
String	basicConsume(String queue, boolean autoAck, String consumerTag, DeliverCallback deliverCallback, CancelCallback cancelCallback) Start a non-nolocal, non-exclusive consumer.	支持
String	basicConsume(String queue, boolean autoAck, String consumerTag, DeliverCallback deliverCallback, CancelCallback cancelCallback, ConsumerShutdownSignalCallback shutdownSignalCallback) Start a non-nolocal, non-exclusive consumer.	支持
String	basicConsume(String queue, boolean autoAck, String consumerTag, DeliverCallback deliverCallback, ConsumerShutdownSignalCallback shutdownSignalCallback) Start a non-nolocal, non-exclusive consumer.	支持
String	basicConsume(String queue, boolean autoAck, Map<String,Object> arguments, Consumer callback) Start a non-nolocal, non-exclusive consumer, with a server-generated consumerTag and specified arguments.	支持
String	basicConsume(String queue, boolean autoAck, Map<String,Object> arguments, DeliverCallback deliverCallback, CancelCallback cancelCallback) Start a non-nolocal, non-exclusive consumer, with a server-generated consumerTag and specified arguments.	支持
String	basicConsume(String queue, boolean autoAck, Map<String,Object> arguments, DeliverCallback deliverCallback, CancelCallback cancelCallback, ConsumerShutdownSignalCallback shutdownSignalCallback) Start a non-nolocal, non-exclusive consumer, with a server-generated consumerTag and specified arguments.	支持
String	basicConsume(String queue, boolean autoAck, Map<String,Object> arguments, DeliverCallback deliverCallback, ConsumerShutdownSignalCallback shutdownSignalCallback) Start a non-nolocal, non-exclusive consumer, with a server-generated consumerTag and specified arguments.	支持

String	basicConsume(String queue, Consumer callback) Start a non-nolocal, non-exclusive consumer, with explicit acknowledgement and a server-generated consumerTag.	支持
String	basicConsume(String queue, DeliverCallback deliverCallback, CancelCallback cancelCallback) Start a non-nolocal, non-exclusive consumer, with explicit acknowledgement and a server-generated consumerTag.	支持
String	basicConsume(String queue, DeliverCallback deliverCallback, CancelCallback cancelCallback, ConsumerShutdownSignalCallback shutdownSignalCallback) Start a non-nolocal, non-exclusive consumer, with explicit acknowledgement and a server-generated consumerTag.	支持
String	basicConsume(String queue, DeliverCallback deliverCallback, ConsumerShutdownSignalCallback shutdownSignalCallback) Start a non-nolocal, non-exclusive consumer, with explicit acknowledgement and a server-generated consumerTag.	支持
GetResponse	basicGet(String queue, boolean autoAck) Retrieve a message from a queue using `AMQP.Basic.Get`	支持
void	basicNack(long deliveryTag, boolean multiple, boolean requeue) Reject one or several received messages.	支持
void	basicPublish(String exchange, String routingKey, boolean mandatory, boolean immediate, AMQP.BasicProperties props, byte[] body) Publish a message.	支持
void	basicPublish(String exchange, String routingKey, boolean mandatory, AMQP.BasicProperties props, byte[] body) Publish a message.	支持
void	basicPublish(String exchange, String routingKey, AMQP.BasicProperties props, byte[] body) Publish a message.	支持
void	basicQos(int prefetchCount) Request a specific prefetchCount "quality of service" settings for this channel.	支持
void	basicQos(int prefetchCount, boolean global) Request a specific prefetchCount "quality of service" settings for this channel.	支持

void	basicQos(int prefetchSize, int prefetchCount, boolean global) Request specific "quality of service" settings.	支持
AMQP.Basic.RecoverOk	basicRecover() Ask the broker to resend unacknowledged messages.	支持
AMQP.Basic.RecoverOk	basicRecover(boolean requeue) Ask the broker to resend unacknowledged messages.	支持
void	basicReject(long deliveryTag, boolean requeue) Reject a message.	支持
void	clearConfirmListeners() Remove all `ConfirmListener`s.	支持
void	clearReturnListeners() Remove all `ReturnListener`s.	支持
void	close() Close this channel with the `AMQP.REPLY_SUCCESS` close code and message 'OK'.	支持
void	close(int closeCode, String closeMessage) Close this channel.	支持
AMQP.Confirm.SelectOk	confirmSelect() Enables publisher acknowledgements on this channel.	支持
long	consumerCount(String queue) Returns the number of consumers on a queue.	支持
AMQP.Exchange.BindOk	exchangeBind(String destination, String source, String routingKey) Bind an exchange to an exchange, with no extra arguments.	支持
AMQP.Exchange.BindOk	exchangeBind(String destination, String source, String routingKey, Map<String,Object> arguments) Bind an exchange to an exchange.	支持
void	exchangeBindNoWait(String destination, String source, String routingKey, Map<String,Object> arguments) Like `exchangeBind(String, String, String, java.util.Map)`) but sets nowait parameter to true and returns void (as there will be no response from the server).	支持
AMQP.Exchange.DeclareOk	exchangeDeclare(String exchange, BuiltinExchangeType type) Actively declare a non-autodelete, non-durable exchange with no extra arguments	支持

AMQP.Exchange.DeclareOk	exchangeDeclare(String exchange, BuiltinExchangeType type, boolean durable) Actively declare a non-autodelete exchange with no extra arguments	支持
AMQP.Exchange.DeclareOk	exchangeDeclare(String exchange, BuiltinExchangeType type, boolean durable, boolean autoDelete, boolean internal, Map<String,Object> arguments) Declare an exchange, via an interface that allows the complete set of arguments.	支持
AMQP.Exchange.DeclareOk	exchangeDeclare(String exchange, BuiltinExchangeType type, boolean durable, boolean autoDelete, Map<String,Object> arguments) Declare an exchange.	支持
AMQP.Exchange.DeclareOk	exchangeDeclare(String exchange, String type) Actively declare a non-autodelete, non-durable exchange with no extra arguments	支持
AMQP.Exchange.DeclareOk	exchangeDeclare(String exchange, String type, boolean durable) Actively declare a non-autodelete exchange with no extra arguments	支持
AMQP.Exchange.DeclareOk	exchangeDeclare(String exchange, String type, boolean durable, boolean autoDelete, boolean internal, Map<String,Object> arguments) Declare an exchange, via an interface that allows the complete set of arguments.	支持
AMQP.Exchange.DeclareOk	exchangeDeclare(String exchange, String type, boolean durable, boolean autoDelete, Map<String,Object> arguments) Declare an exchange.	支持
void	exchangeDeclareNoWait(String exchange, BuiltinExchangeType type, boolean durable, boolean autoDelete, boolean internal, Map<String,Object> arguments) Like `exchangeDeclare(String, String, boolean, boolean, java.util.Map)`) but sets nowait parameter to true and returns nothing (as there will be no response from the server).	支持
void	exchangeDeclareNoWait(String exchange, String type, boolean durable, boolean autoDelete, boolean internal, Map<String,Object> arguments) Like `exchangeDeclare(String, String, boolean, boolean, java.util.Map)`) but sets nowait parameter to true and returns nothing (as there will be no response from the server).	支持
AMQP.Exchange.DeclareOk	exchangeDeclarePassive(String name) Declare an exchange passively; that is, check if the named exchange	支持

	exists.	
AMQP.Exchange.DeleteOk	exchangeDelete(String exchange) Delete an exchange, without regard for whether it is in use or not	支持
AMQP.Exchange.DeleteOk	exchangeDelete(String exchange, boolean ifUnused) Delete an exchange	支持
void	exchangeDeleteNoWait(String exchange, boolean ifUnused) Like `exchangeDelete(String, boolean)` but sets nowait parameter to true and returns void (as there will be no response from the server).	支持
AMQP.Exchange.UnbindOk	exchangeUnbind(String destination, String source, String routingKey) Unbind an exchange from an exchange, with no extra arguments.	支持
AMQP.Exchange.UnbindOk	exchangeUnbind(String destination, String source, String routingKey, Map<String,Object> arguments) Unbind an exchange from an exchange.	支持
void	exchangeUnbindNoWait(String destination, String source, String routingKey, Map<String,Object> arguments) Same as `exchangeUnbind(String, String, String, java.util.Map)`) but sets no-wait parameter to true and returns nothing (as there will be no response from the server).	支持
int	getChannelNumber() Retrieve this channel's channel number.	支持
Connection	getConnection() Retrieve the connection which carries this channel.	支持
Consumer	getDefaultConsumer() Get the current default consumer.	支持
long	getNextPublishSeqNo() When in confirm mode, returns the sequence number of the next message to be published.	支持
long	messageCount(String queue) Returns the number of messages in a queue ready to be delivered to consumers.	支持
AMQP.Queue.BindOk	queueBind(String queue, String exchange, String routingKey) Bind a queue to an exchange, with no extra arguments.	支持
AMQP.Queue.BindOk	queueBind(String queue, String exchange, String routingKey, Map<String,Object> arguments)	支持

	Bind a queue to an exchange.	
void	queueBindNoWait(String queue, String exchange, String routingKey, Map<String,Object> arguments) Same as `queueBind(String, String, String, java.util.Map)`) but sets nowait parameter to true and returns void (as there will be no response from the server).	支持
AMQP.Queue.DeclareOk	queueDeclare() Actively declare a server-named exclusive, autodelete, non-durable queue.	支持
AMQP.Queue.DeclareOk	queueDeclare(String queue, boolean durable, boolean exclusive, boolean autoDelete, Map<String,Object> arguments) Declare a queue	支持
void	queueDeclareNoWait(String queue, boolean durable, boolean exclusive, boolean autoDelete, Map<String,Object> arguments) Like `queueDeclare(String, boolean, boolean, boolean, java.util.Map)`) but sets nowait flag to true and returns no result (as there will be no response from the server).	支持
AMQP.Queue.DeclareOk	queueDeclarePassive(String queue) Declare a queue passively; i.e., check if it exists.	支持
AMQP.Queue.DeleteOk	queueDelete(String queue) Delete a queue, without regard for whether it is in use or has messages on it	支持
AMQP.Queue.DeleteOk	queueDelete(String queue, boolean ifUnused, boolean ifEmpty) Delete a queue	支持
void	queueDeleteNoWait(String queue, boolean ifUnused, boolean ifEmpty) Like `queueDelete(String, boolean, boolean)`) but sets nowait parameter to true and returns nothing (as there will be no response from the server).	支持
AMQP.Queue.PurgeOk	queuePurge(String queue) Purges the contents of the given queue.	支持
AMQP.Queue.UnbindOk	queueUnbind(String queue, String exchange, String routingKey) Unbinds a queue from an exchange, with no extra arguments.	支持
AMQP.Queue.UnbindOk	queueUnbind(String queue, String exchange, String routingKey, Map<String,Object> arguments) Unbind a queue from an exchange.	支持

boolean	removeConfirmListener(ConfirmListener listener) Remove a `ConfirmListener`.	支持
boolean	removeReturnListener(ReturnListener listener) Remove a `ReturnListener`.	支持
void	setDefaultConsumer(Consumer consumer) Set the current default consumer.	支持
boolean	waitForConfirms() Wait until all messages published since the last call have been either ack'd or nack'd by the broker.	支持
boolean	waitForConfirms(long timeout) Wait until all messages published since the last call have been either ack'd or nack'd by the broker; or until timeout elapses.	支持
void	waitForConfirmsOrDie() Wait until all messages published since the last call have been either ack'd or nack'd by the broker.	支持
void	waitForConfirmsOrDie(long timeout) Wait until all messages published since the last call have been either ack'd or nack'd by the broker; or until timeout elapses.	支持

> *提示*
>
> *由于不支持prefetchSize，调用basicQos(int prefetchSize, int prefetchCount, boolean global)prefetchSize不能传大于0的参数，否则会报"prefetchSize not supported "错误。*

4.2 Consumer API

应用程序回调对象通过订阅从队列接收通知和消息的接口。大多数实现都将子类化DefaultConsumer。

此接口的方法在调度线程中调用，该调度线程与连接的线程是分离的。这允许消费者调用通道或连接方法而不会导致死锁。

修饰符和类型	方法和描述	是否支持
void	handleCancel(String consumerTag) Called when the consumer is cancelled for reasons *other than* by a call to `Channel.basicCancel(java.lang.String)`.	支持

void	handleCancelOk(String consumerTag) Called when the consumer is cancelled by a call to `Channel.basicCancel(java.lang.String)`.	支持
void	handleConsumeOk(String consumerTag) Called when the consumer is registered by a call to any of the `Channel.basicConsume(java.lang.String, com.rabbitmq.client.Consumer)`) methods.	支持
void	handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) Called when a `**basic.deliver**` is received for this consumer.	支持
void	handleRecoverOk(String consumerTag) Called when a `**basic.recover-ok**` is received in reply to a `**basic.recover**`.	支持
void	handleShutdownSignal(String consumerTag, ShutdownSignalException sig) Called when either the channel or the underlying connection has been shut down.	支持

5 JMS客户端

不支持。

6 流Java客户端

不支持。

7 管理HTTP API

不支持。

8 Kafka APIS

Kafka包括五个核心api:

1. Producer API允许应用程序向Kafka集群中的主题发送数据流。
2. Consumer API允许应用程序从Kafka集群中的主题读取数据流。
3. Streams API允许将数据流从输入主题转换为输出主题。
4. Connect API允许实现连接器，这些连接器可以不断地从一些源系统或应用程序拉入Kafka，或从Kafka推入一些接收器系统或应用程序。
5. Admin API允许管理和检查主题、代理和其他Kafka对象。

   由于目前生产环境大多数使用的是Kafka为2.0.0及以上版本，3.0.0及以上版本生产环境使用并不多，下面APIS支持情况以Kafka 2.0.0最稳定的版本2.7.2进行说明。

8.1 Producer API

一个Kafka客户端，它向Kafka集群发布记录。

生产者是线程安全的，跨线程共享一个生产者实例通常比拥有多个实例更快。

修饰符和类型	方法和描述	是否支持
`void`	`abortTransaction()` Aborts the ongoing transaction.	支持
`void`	`beginTransaction()` Should be called before the start of each new transaction.	支持
`void`	`close()` Close this producer.	支持
`void`	`close(Duration timeout)` This method waits up to `timeout` for the producer to complete the sending of all incomplete requests.	支持
`void`	`commitTransaction()` Commits the ongoing transaction.	支持

void	flush() Invoking this method makes all buffered records immediately available to send (even if linger.ms is greater than 0) and blocks on the completion of the requests associated with these records.	支持
void	initTransactions() Needs to be called before any other methods when the transactional.id is set in the configuration.	支持
Map<MetricName,? extends Metric>	metrics() Get the full set of internal metrics maintained by the producer.	支持
List<PartitionInfo>	partitionsFor(String topic) Get the partition metadata for the given topic.	支持
Future<RecordMetadata>	send(ProducerRecord<K,V> record) Asynchronously send a record to a topic.	支持
Future<RecordMetadata>	send(ProducerRecord<K,V> record, Callback callback) Asynchronously send a record to a topic and invoke the provided callback when the send has been acknowledged.	支持
void	sendOffsetsToTransaction(Map<TopicPartition,OffsetAndMetadata> offsets, ConsumerGroupMetadata groupMetadata) Sends a list of specified offsets to the consumer group coordinator, and also marks those offsets as part of the current transaction.	支持
void	sendOffsetsToTransaction(Map<TopicPartition,OffsetAndMetadata> offsets, String consumerGroupId) Sends a list of specified offsets to the consumer group coordinator, and also marks those offsets as part of the current transaction.	支持

8.2 Consumer API

消费Kafka集群记录的客户端。这个客户端透明地处理Kafka代理的失败，并透明地适应它获取的主题分区在集群内的迁移。此客户机还与代理交互，以允许消费者组使用消费者组来负载平衡消费。

消费者维护必要的代理的TCP连接以获取数据。使用后关闭消费者失败将会释放这些连接。消费者不是线程安全的。

修饰符和类型	方法和描述	是否支持

void	assign(Collection<TopicPartition> partitions) Manually assign a list of partitions to this consumer.	支持
Set<TopicPartition>	assignment() Get the set of partitions currently assigned to this consumer.	支持
Map<TopicPartition,Long>	beginningOffsets(Collection<TopicPartition> partitions) Get the first offset for the given partitions.	支持
Map<TopicPartition,Long>	beginningOffsets(Collection<TopicPartition> partitions, Duration timeout) Get the first offset for the given partitions.	支持
void	close() Close the consumer, waiting for up to the default timeout of 30 seconds for any needed cleanup.	支持
void	close(Duration timeout) Tries to close the consumer cleanly within the specified timeout.	支持
void	close(long timeout, TimeUnit timeUnit) Deprecated. Since 2.0. Use close(Duration) or close().	支持
void	commitAsync() Commit offsets returned on the last poll(Duration) for all the subscribed list of topics and partition.	支持
void	commitAsync(Map<TopicPartition,OffsetAndMetadata> offsets, OffsetCommitCallback callback) Commit the specified offsets for the specified list of topics and partitions to Kafka.	支持
void	commitAsync(OffsetCommitCallback callback) Commit offsets returned on the last poll() for the subscribed list of topics and partitions.	支持
void	commitSync() Commit offsets returned on the last poll() for all the subscribed list of topics and partitions.	支持
void	commitSync(Duration timeout) Commit offsets returned on the last poll() for all the	支持

	subscribed list of topics and partitions.	
void	commitSync(Map<TopicPartition,OffsetAndMetadata> offsets) Commit the specified offsets for the specified list of topics and partitions.	支持
void	commitSync(Map<TopicPartition,OffsetAndMetadata> offsets, Duration timeout) Commit the specified offsets for the specified list of topics and partitions.	支持
Map<TopicPartition,OffsetAndMetadata>	committed(Set<TopicPartition> partitions) Get the last committed offsets for the given partitions (whether the commit happened by this process or another).	支持
Map<TopicPartition,OffsetAndMetadata>	committed(Set<TopicPartition> partitions, Duration timeout) Get the last committed offsets for the given partitions (whether the commit happened by this process or another).	支持
OffsetAndMetadata	committed(TopicPartition partition) Deprecated. since 2.4 Use committed(Set) instead	支持
OffsetAndMetadata	committed(TopicPartition partition, Duration timeout) Deprecated. since 2.4 Use committed(Set, Duration) instead	支持
Map<TopicPartition,Long>	endOffsets(Collection<TopicPartition> partitions) Get the end offsets for the given partitions.	支持
Map<TopicPartition,Long>	endOffsets(Collection<TopicPartition> partitions, Duration timeout) Get the end offsets for the given partitions.	支持
void	enforceRebalance() Alert the consumer to trigger a new rebalance by rejoining the group.	支持
ConsumerGroupMetadata	groupMetadata() Return the current group metadata associated with this consumer.	支持

`Map<String,List<PartitionInfo>>`	`listTopics()` Get metadata about partitions for all topics that the user is authorized to view.	支持
`Map<String,List<PartitionInfo>>`	`listTopics(Duration timeout)` Get metadata about partitions for all topics that the user is authorized to view.	支持
`Map<MetricName,? extends Metric>`	`metrics()` Get the metrics kept by the consumer	支持
`Map<TopicPartition,OffsetAndTimestamp>`	`offsetsForTimes(Map<TopicPartition,Long> timestampsToSearch)` Look up the offsets for the given partitions by timestamp.	支持
`Map<TopicPartition,OffsetAndTimestamp>`	`offsetsForTimes(Map<TopicPartition,Long> timestampsToSearch, Duration timeout)` Look up the offsets for the given partitions by timestamp.	支持
`List<PartitionInfo>`	`partitionsFor(String topic)` Get metadata about the partitions for a given topic.	支持
`List<PartitionInfo>`	`partitionsFor(String topic, Duration timeout)` Get metadata about the partitions for a given topic.	支持
`void`	`pause(Collection<TopicPartition> partitions)` Suspend fetching from the requested partitions.	支持
`Set<TopicPartition>`	`paused()` Get the set of partitions that were previously paused by a call to `pause(Collection)`.	支持
`ConsumerRecords<K,V>`	`poll(Duration timeout)` Fetch data for the topics or partitions specified using one of the subscribe/assign APIs.	支持
`ConsumerRecords<K,V>`	`poll(long timeoutMs)` **Deprecated.** Since 2.0. Use `poll(Duration)`, which does not block beyond the timeout awaiting partition assignment. See KIP-266 for more information.	支持
`long`	`position(TopicPartition partition)` Get the offset of the *next record* that will be fetched (if a record with that offset exists).	支持

long	position(TopicPartition partition, Duration timeout) Get the offset of the *next record* that will be fetched (if a record with that offset exists).	支持
void	resume(Collection<TopicPartition> partitions) Resume specified partitions which have been paused with pause(Collection).	支持
void	seek(TopicPartition partition, long offset) Overrides the fetch offsets that the consumer will use on the next poll(timeout).	支持
void	seek(TopicPartition partition, OffsetAndMetadata offsetAndMetadata) Overrides the fetch offsets that the consumer will use on the next poll(timeout).	支持
void	seekToBeginning(Collection<TopicPartition> partitions) Seek to the first offset for each of the given partitions.	支持
void	seekToEnd(Collection<TopicPartition> partitions) Seek to the last offset for each of the given partitions.	支持
void	subscribe(Collection<String> topics) Subscribe to the given list of topics to get dynamically assigned partitions.	支持
void	subscribe(Collection<String> topics, ConsumerRebalanceListener listener) Subscribe to the given list of topics to get dynamically assigned partitions.	支持
void	subscribe(Pattern pattern) Subscribe to all topics matching specified pattern to get dynamically assigned partitions.	支持
void	subscribe(Pattern pattern, ConsumerRebalanceListener listener) Subscribe to all topics matching specified pattern to get dynamically assigned partitions.	支持
Set<String>	subscription() Get the current subscription.	支持

void	unsubscribe() Unsubscribe from topics currently subscribed with subscribe(Collection) or subscribe(Pattern).	支持
void	wakeup() Wakeup the consumer.	支持

8.3 Streams API

一个Kafka客户端，允许对来自一个或多个输入主题的输入执行连续计算，并将输出发送到零个、一个或多个输出主题。计算逻辑可以通过使用拓扑来定义处理器的DAG拓扑，或者通过使用提供高级DSL来定义转换的StreamsBuilder来指定。

一个KafkaStreams实例可以包含一个或多个在配置中指定的线程处理工作。一个KafkaStreams实例可以作为一个单独的(可能是分布式的)流处理应用程序与具有相同应用程序ID的任何其他实例(无论是在同一进程中，在本机的其他进程上，还是在远程机器上)进行协调。这些实例将根据输入主题分区的分配来划分工作，以便使用所有分区。如果实例被添加或失败，所有(剩余的)实例将在它们之间重新平衡分区分配，以平衡处理负载并确保处理所有输入主题分区。

在内部，KafkaStreams实例包含一个普通的KafkaProducer和KafkaConsumer实例，用于读取输入和写入输出。

Streams API主要包括KSteam（KGroupedStream、TimeWindowedKStream 、SessionWindowedKStream ）、KTable等API。

8.3.1 KStream API

修饰符和类型	方法和描述	是否支持
<KR> KStream<KR,V>	selectKey(KeyValueMapper<? super K,? super V,? extends KR> mapper) Set a new key (with possibly new type) for each input record.	支持
<KR> KStream<KR,V>	selectKey(KeyValueMapper<? super K,? super V,? extends KR> mapper, Named named) Set a new key (with possibly new type) for each input record.	支持
<VR> KStream<K,VR>	mapValues(ValueMapper<? super V,? extends VR> mapper) Transform the value of each input record into a new value (with possible new type) of the output record.	支持
<VR> KStream<K,VR>	mapValues(ValueMapper<? super V,? extends VR> mapper, Named named) Transform the value of each input record into a new value (with possible new type) of the output record.	支持

<VR> KStream<K,VR>	mapValues(ValueMapperWithKey<? super K,? super V,? extends VR> mapper) Transform the value of each input record into a new value (with possible new type) of the output record.	支持
<VR> KStream<K,VR>	mapValues(ValueMapperWithKey<? super K,? super V,? extends VR> mapper, Named named) Transform the value of each input record into a new value (with possible new type) of the output record.	支持
<VR> KStream<K,VR>	flatMapValues(ValueMapper<? super V,? extends Iterable<? extends VR>> mapper) Create a new KStream by transforming the value of each record in this stream into zero or more values with the same key in the new stream.	支持
<VR> KStream<K,VR>	flatMapValues(ValueMapper<? super V,? extends Iterable<? extends VR>> mapper, Named named) Create a new KStream by transforming the value of each record in this stream into zero or more values with the same key in the new stream.	支持
<VR> KStream<K,VR>	flatMapValues(ValueMapperWithKey<? super K,? super V,? extends Iterable<? extends VR>> mapper) Create a new KStream by transforming the value of each record in this stream into zero or more values with the same key in the new stream.	支持
<VR> KStream<K,VR>	flatMapValues(ValueMapperWithKey<? super K,? super V,? extends Iterable<? extends VR>> mapper, Named named) Create a new KStream by transforming the value of each record in this stream into zero or more values with the same key in the new stream.	支持
void	foreach(ForeachAction<? super K,? super V> action) Perform an action on each record of KStream.	支持
void	foreach(ForeachAction<? super K,? super V> action, Named named) Perform an action on each record of KStream.	支持
void	to(String topic) Materialize this stream to a topic using default serializers specified in the config and producer's DefaultPartitioner.	支持
void	to(String topic, Produced<K,V> produced) Materialize this stream to a topic using the provided Produced instance.	支持
void	to(TopicNameExtractor<K,V> topicExtractor) Dynamically materialize this stream to topics using default serializers	支持

	specified in the config and producer's `DefaultPartitioner`.	
`void`	`to(TopicNameExtractor<K,V> topicExtractor, Produced<K,V> produced)` Dynamically materialize this stream to topics using the provided `Produced` instance.	支持
`<KR>` `KGroupedStream<KR,V>`	`groupBy(KeyValueMapper<? super K,? super V,KR> keySelector)` Group the records of this `KStream` on a new key that is selected using the provided `KeyValueMapper` and default serializers and deserializers.	支持
`<KR>` `KGroupedStream<KR,V>`	`groupBy(KeyValueMapper<? super K,? super V,KR> keySelector, Grouped<KR,V> grouped)` Group the records of this `KStream` on a new key that is selected using the provided `KeyValueMapper` and `Serde`s as specified by `Grouped`.	支持
`<KR>` `KGroupedStream<KR,V>`	`groupBy(KeyValueMapper<? super K,? super V,KR> keySelector, Serialized<KR,V> serialized)` **Deprecated.** since 2.1. Use `groupBy(KeyValueMapper, Grouped)` instead	支持
`KGroupedStream<K,V>`	`groupByKey()` Group the records by their current key into a `KGroupedStream` while preserving the original values and default serializers and deserializers.	支持
`KGroupedStream<K,V>`	`groupByKey(Grouped<K,V> grouped)` Group the records by their current key into a `KGroupedStream` while preserving the original values and using the serializers as defined by `Grouped`.	支持
`KGroupedStream<K,V>`	`groupByKey(Serialized<K,V> serialized)` **Deprecated.** since 2.1. Use `groupByKey(Grouped)` instead	支持
`<GK,GV,RV>` `KStream<K,RV>`	`join(GlobalKTable<GK,GV> globalTable, KeyValueMapper<? super K,? super V,? extends GK> keySelector, ValueJoiner<? super V,? super GV,? extends RV> joiner)` Join records of this stream with `GlobalKTable`'s records using non-windowed inner equi join.	支持
`<GK,GV,RV>` `KStream<K,RV>`	`join(GlobalKTable<GK,GV> globalTable, KeyValueMapper<? super K,? super V,? extends GK> keySelector, ValueJoiner<? super V,? super GV,? extends RV> joiner, Named named)` Join records of this stream with `GlobalKTable`'s records using non-windowed inner equi join.	支持
`<GK,GV,RV>` `KStream<K,RV>`	`leftJoin(GlobalKTable<GK,GV> globalTable, KeyValueMapper<? super K,? super V,? extends GK> keySelector, ValueJoiner<? super V,? super GV,?`	支持

	extends RV> valueJoiner) Join records of this stream with GlobalKTable's records using non-windowed left equi join.	
<GK,GV,RV> KStream<K,RV>	leftJoin(GlobalKTable<GK,GV> globalTable, KeyValueMapper<? super K,? super V,? extends GK> keySelector, ValueJoiner<? super V,? super GV,? extends RV> valueJoiner, Named named) Join records of this stream with GlobalKTable's records using non-windowed left equi join.	支持
<K1,V1> KStream<K1,V1>	transform(TransformerSupplier<? super K,? super V,KeyValue<K1,V1>> transformerSupplier, Named named, String... stateStoreNames) Transform each record of the input stream into zero or one record in the output stream (both key and value type can be altered arbitrarily).	支持
<K1,V1> KStream<K1,V1>	transform(TransformerSupplier<? super K,? super V,KeyValue<K1,V1>> transformerSupplier, String... stateStoreNames) Transform each record of the input stream into zero or one record in the output stream (both key and value type can be altered arbitrarily).	支持
<K1,V1> KStream<K1,V1>	flatTransform(TransformerSupplier<? super K,? super V,Iterable<KeyValue<K1,V1>>> transformerSupplier, Named named, String... stateStoreNames) Transform each record of the input stream into zero or more records in the output stream (both key and value type can be altered arbitrarily).	支持
<K1,V1> KStream<K1,V1>	flatTransform(TransformerSupplier<? super K,? super V,Iterable<KeyValue<K1,V1>>> transformerSupplier, String... stateStoreNames) Transform each record of the input stream into zero or more records in the output stream (both key and value type can be altered arbitrarily).	支持

> *提示*
>
> *由于KStream API方法比较多，以上列出的方法只包括提供单元测试和自测的，没列出的方法大多数应该也支持，具体待实际应用中验证。*

8.3.2 KGroupedStream API

修饰符和类型	方法和描述

<VR> KTable<K,VR>	aggregate(Initializer<VR> initializer, Aggregator<? super K,? super V,VR> aggregator) Aggregate the values of records in this stream by the grouped key.
<VR> KTable<K,VR>	aggregate(Initializer<VR> initializer, Aggregator<? super K,? super V,VR> aggregator, Materialized<K,VR,KeyValueStore<org.apache.kafka.common.utils.Bytes,byte[]> materialized) Aggregate the values of records in this stream by the grouped key.
<VR> KTable<K,VR>	aggregate(Initializer<VR> initializer, Aggregator<? super K,? super V,VR> aggregator, Named named, Materialized<K,VR,KeyValueStore<org.apache.kafka.common.utils.Bytes,byte[]> materialized) Aggregate the values of records in this stream by the grouped key.
KTable<K,Long>	count() Count the number of records in this stream by the grouped key.
KTable<K,Long>	count(Materialized<K,Long,KeyValueStore<org.apache.kafka.common.utils.Byte materialized) Count the number of records in this stream by the grouped key.
KTable<K,Long>	count(Named named) Count the number of records in this stream by the grouped key.
KTable<K,Long>	count(Named named, Materialized<K,Long,KeyValueStore<org.apache.kafka.common.utils.Bytes,byte materialized) Count the number of records in this stream by the grouped key.
KTable<K,V>	reduce(Reducer<V> reducer) Combine the values of records in this stream by the grouped key.
KTable<K,V>	reduce(Reducer<V> reducer, Materialized<K,V,KeyValueStore<org.apache.kafka.common.utils.Bytes,byte[]> materialized) Combine the value of records in this stream by the grouped key.
KTable<K,V>	reduce(Reducer<V> reducer, Named named, Materialized<K,V,KeyValueStore<org.apache.kafka.common.utils.Bytes,byte[]> materialized) Combine the value of records in this stream by the grouped key.

SessionWindowedKStream<K,V>	windowedBy(SessionWindows windows) Create a new SessionWindowedKStream instance that can be used to perform sess windowed aggregations.
TimeWindowedKStream<K,V>	windowedBy(SlidingWindows windows) Create a new TimeWindowedKStream instance that can be used to perform sliding aggregations
<W extends Window> TimeWindowedKStream<K,V>	windowedBy(Windows<W> windows) Create a new TimeWindowedKStream instance that can be used to perform windov aggregations.

> *提示*
>
> *除了 `<VOut> CogroupedKStream<K,VOut>` cogroup(Aggregator<? super K,? super V,VOut> aggregator)方法暂未验证外，其它方法都支持。*

8.3.3 TimeWindowedKStream API

修饰符和类型	方法和描述
<VR> KTable<Windowed<K>,VR>	aggregate(Initializer<VR> initializer, Aggregator<? super K,? super V,VR> aggregator) Aggregate the values of records in this stream by the grouped key and defined win
<VR> KTable<Windowed<K>,VR>	aggregate(Initializer<VR> initializer, Aggregator<? super K,? super V,VR> aggregator, Materialized<K,VR,WindowStore<org.apache.kafka.common.utils.Bytes,byte[]>> materialized) Aggregate the values of records in this stream by the grouped key and defined win
<VR> KTable<Windowed<K>,VR>	aggregate(Initializer<VR> initializer, Aggregator<? super K,? super V,VR> aggregator, Named named) Aggregate the values of records in this stream by the grouped key and defined win
<VR> KTable<Windowed<K>,VR>	aggregate(Initializer<VR> initializer, Aggregator<? super K,? super V,VR> aggregator, Named named, Materialized<K,VR,WindowStore<org.apache.kafka.common.utils.Bytes,byte[]>> materialized) Aggregate the values of records in this stream by the grouped key and defined win

KTable<Windowed<K>,Long>	count() Count the number of records in this stream by the grouped key and defined windo
KTable<Windowed<K>,Long>	count(Materialized<K,Long,WindowStore<org.apache.kafka.common.utils.Bytes,byt materialized) Count the number of records in this stream by the grouped key and defined windo
KTable<Windowed<K>,Long>	count(Named named) Count the number of records in this stream by the grouped key and defined windo
KTable<Windowed<K>,Long>	count(Named named, Materialized<K,Long,WindowStore<org.apache.kafka.common.utils.Bytes,byte[]>> materialized) Count the number of records in this stream by the grouped key and defined windo
KTable<Windowed<K>,V>	reduce(Reducer<V> reducer) Combine the values of records in this stream by the grouped key and defined wind
KTable<Windowed<K>,V>	reduce(Reducer<V> reducer, Materialized<K,V,WindowStore<org.apache.kafka.common.utils.Bytes,byte[]>> materialized) Combine the values of records in this stream by the grouped key and defined wind
KTable<Windowed<K>,V>	reduce(Reducer<V> reducer, Named named) Combine the values of records in this stream by the grouped key and defined wind
KTable<Windowed<K>,V>	reduce(Reducer<V> reducer, Named named, Materialized<K,V,WindowStore<org.apache.kafka.common.utils.Bytes,byte[]>> materialized) Combine the values of records in this stream by the grouped key and defined wind

8.3.4 SessionWindowedKStream API

修饰符和类型	方法和描述
KTable<Windowed<K>,Long>	count() Count the number of records in this stream by the grouped key and defined sessio
KTable<Windowed<K>,Long>	count(Materialized<K,Long,SessionStore<org.apache.kafka.common.utils.Bytes,by materialized) Count the number of records in this stream by the grouped key and defined sessio

KTable<Windowed<K>,Long>	count(Named named) Count the number of records in this stream by the grouped key and defined sessio
KTable<Windowed<K>,Long>	count(Named named, Materialized<K,Long,SessionStore<org.apache.kafka.common.utils.Bytes,byte[]>> materialized) Count the number of records in this stream by the grouped key and defined sessio
KTable<Windowed<K>,V>	reduce(Reducer<V> reducer) Combine the values of records in this stream by the grouped key and defined sessi
KTable<Windowed<K>,V>	reduce(Reducer<V> reducer, Materialized<K,V,SessionStore<org.apache.kafka.common.utils.Bytes,byte[]>> materialized) Combine the values of records in this stream by the grouped key and defined sessi
KTable<Windowed<K>,V>	reduce(Reducer<V> reducer, Named named) Combine the values of records in this stream by the grouped key and defined sessi
KTable<Windowed<K>,V>	reduce(Reducer<V> reducer, Named named, Materialized<K,V,SessionStore<org.apache.kafka.common.utils.Bytes,byte[]>> materialized) Combine the values of records in this stream by the grouped key and defined sessi

> *提示*
>
> *除了* `<VR> KTable<Windowed<K>,VR> aggregate(Initializer<VR> initializer, Aggregator<? super K,? super V,VR> aggregator, Merger<? super K,VR> sessionMerger, ...)` *系列方法暂未验证外（但从TimeWindowedKStream API支持情况推断，aggregate系列方法应该也支持），其它方法都支持。*

8.3.5 KTable API

.

修饰符和类型	方法和描述	是否支持
<VR> KTable<K,VR>	mapValues(ValueMapper<? super V,? extends VR> mapper) Create a new `KTable` by transforming the value of each record in this `KTable` into a new value (with possibly a new type) in the new `KTable`, with default serializers, deserializers, and state store.	支持

<VR> KTable<K,VR>	mapValues(ValueMapper<? super V,? extends VR> mapper, Materialized<K,VR,KeyValueStore<org.apache.kafka.common.utils.Bytes,byte[]>> materialized) Create a new KTable by transforming the value of each record in this KTable into a new value (with possibly a new type) in the new KTable, with the key serde, value serde, and the underlying materialized state storage configured in the Materialized instance.	支持
<VR> KTable<K,VR>	mapValues(ValueMapper<? super V,? extends VR> mapper, Named named) Create a new KTable by transforming the value of each record in this KTable into a new value (with possibly a new type) in the new KTable, with default serializers, deserializers, and state store.	支持
<VR> KTable<K,VR>	mapValues(ValueMapper<? super V,? extends VR> mapper, Named named, Materialized<K,VR,KeyValueStore<org.apache.kafka.common.utils.Bytes,byte[]>> materialized) Create a new KTable by transforming the value of each record in this KTable into a new value (with possibly a new type) in the new KTable, with the key serde, value serde, and the underlying materialized state storage configured in the Materialized instance.	支持
<VR> KTable<K,VR>	mapValues(ValueMapperWithKey<? super K,? super V,? extends VR> mapper) Create a new KTable by transforming the value of each record in this KTable into a new value (with possibly a new type) in the new KTable, with default serializers, deserializers, and state store.	支持
<VR> KTable<K,VR>	mapValues(ValueMapperWithKey<? super K,? super V,? extends VR> mapper, Materialized<K,VR,KeyValueStore<org.apache.kafka.common.utils.Bytes,byte[]>> materialized) Create a new KTable by transforming the value of each record in this KTable into a new value (with possibly a new type) in the new KTable, with the key serde, value serde, and the underlying materialized state storage configured in the Materialized instance.	支持
<VR> KTable<K,VR>	mapValues(ValueMapperWithKey<? super K,? super V,? extends VR> mapper, Named named) Create a new KTable by transforming the value of each record in this KTable into a new value (with possibly a new type) in the new KTable, with default serializers, deserializers, and state store.	支持
<VR> KTable<K,VR>	mapValues(ValueMapperWithKey<? super K,? super V,? extends VR> mapper, Named named, Materialized<K,VR,KeyValueStore<org.apache.kafka.common.utils.Bytes,byte[]>> materialized)	支持

	Create a new `KTable` by transforming the value of each record in this `KTable` into a new value (with possibly a new type) in the new `KTable`, with the `key serde`, `value serde`, and the underlying `materialized state storage` configured in the `Materialized` instance.	
`KStream<K,V>`	`toStream()` Convert this changelog stream to a `KStream`.	支持
`<KR> KStream<KR,V>`	`toStream(KeyValueMapper<? super K,? super V,? extends KR> mapper)` Convert this changelog stream to a `KStream` using the given `KeyValueMapper` to select the new key.	支持
`<KR> KStream<KR,V>`	`toStream(KeyValueMapper<? super K,? super V,? extends KR> mapper, Named named)` Convert this changelog stream to a `KStream` using the given `KeyValueMapper` to select the new key.	支持
`KStream<K,V>`	`toStream(Named named)` Convert this changelog stream to a `KStream`.	支持

> *提示*
>
> *由于KTable API方法比较多，以上列出的方法只包括提供单元测试和自测的，没列出的方法大多数应该也支持，具体待实际应用中验证。*

8.4 Connect API

Connect API允许实现连接器，可以不断地从一些源数据系统拉入Kafka，或者从Kafka推入一些汇聚数据系统。

但是，Connect的许多用户不需要直接使用这个API，他们可以使用预先构建的连接器，而不需要编写任何代码。

Connect API主要包括Connector（SourceConnector、SinkConnector）和Task（SourceTask、SinkTask） API。

8.4.1 Connector API

修饰符和类型	方法和描述	是否支持
`abstract ConfigDef`	`config()` Define the configuration for the connector.	支持
`protected ConnectorContext`	`context()` Returns the context object used to interact with the Kafka Connect runtime.	支持

void	initialize(ConnectorContext ctx) Initialize this connector, using the provided ConnectorContext to notify the runtime of input configuration changes.	支持
void	initialize(ConnectorContext ctx, List<Map<String,String>> taskConfigs) Initialize this connector, using the provided ConnectorContext to notify the runtime of input configuration changes and using the provided set of Task configurations.	支持
void	reconfigure(Map<String,String> props) Reconfigure this Connector.	支持
abstract void	start(Map<String,String> props) Start this Connector.	支持
abstract void	stop() Stop this connector.	支持
abstract Class<? extends Task>	taskClass() Returns the Task implementation for this Connector.	支持
abstract List<Map<String,String>>	taskConfigs(int maxTasks) Returns a set of configurations for Tasks based on the current configuration, producing at most count configurations.	支持
Config	validate(Map<String,String> connectorConfigs) Validate the connector configuration values against configuration definitions.	支持
String	version() Get the version of this component.	支持

8.4.1.1 SourceConnector API

修饰符和类型	方法和描述	是否支持
protected SourceConnectorContext	context() Returns the context object used to interact with the Kafka Connect runtime.	支持

8.4.1.2 SinkConnector API

修饰符和类型	方法和描述	是否支持

protected SinkConnectorContext	context() Returns the context object used to interact with the Kafka Connect runtime.	支持

8.4.2 Task API

修饰符和类型	方法和描述	是否支持
void	start(Map<String,String> props) Start the Task	支持
void	stop() Stop this task.	支持
String	version() Get the version of this task.	支持

8.4.2.1 SourceTask API

修饰符和类型	方法和描述	是否支持
void	commit() Commit the offsets, up to the offsets that have been returned by poll().	支持
void	commitRecord(SourceRecord record) Deprecated. Use commitRecord(SourceRecord, RecordMetadata) instead.	支持
void	commitRecord(SourceRecord record, RecordMetadata metadata) Commit an individual SourceRecord when the callback from the producer client is received.	支持
void	initialize(SourceTaskContext context) Initialize this SourceTask with the specified context object.	支持
abstract List<SourceRecord>	poll() Poll this source task for new records.	支持
abstract void	start(Map<String,String> props) Start the Task.	支持
abstract void	stop() Signal this SourceTask to stop.	支持

8.4.2.2 SinkTask API

修饰符和类型	方法和描述	是否支持
void	close(Collection<TopicPartition> partitions) The SinkTask use this method to close writers for partitions that are no longer assigned to the SinkTask.	支持
void	flush(Map<TopicPartition,OffsetAndMetadata> currentOffsets) Flush all records that have been put(Collection) for the specified topic-partitions.	支持
void	initialize(SinkTaskContext context) Initialize the context of this task.	支持
void	onPartitionsAssigned(Collection<TopicPartition> partitions) Deprecated. Use open(Collection) for partition initialization.	支持
void	onPartitionsRevoked(Collection<TopicPartition> partitions) Deprecated. Use close(Collection) instead for partition cleanup.	支持
void	open(Collection<TopicPartition> partitions) The SinkTask use this method to create writers for newly assigned partitions in case of partition rebalance.	支持
Map<TopicPartition,OffsetAndMetadata>	preCommit(Map<TopicPartition,OffsetAndMetadata> currentOffsets) Pre-commit hook invoked prior to an offset commit.	支持
abstract void	put(Collection<SinkRecord> records) Put the records in the sink.	支持
abstract void	start(Map<String,String> props) Start the Task.	支持
abstract void	stop() Perform any cleanup to stop this task.	支持

8.5 Admin API

Admin API支持管理和检查主题、代理、访问控制列表和其他Kafka对象。

修饰符和类型	方法和描述
default AlterConfigsResult	alterConfigs(Map<ConfigResource,Config> configs) Deprecated. Since 2.3. Use incrementalAlterConfigs(Map).
AlterConfigsResult	alterConfigs(Map<ConfigResource,Config> configs, AlterConfigsOptions options) Deprecated. Since 2.3. Use incrementalAlterConfigs(Map, AlterConfigsOptions).
default void	close() Close the Admin and release all associated resources.
void	close(Duration timeout) Close the Admin client and release all associated resources.
default void	close(long duration, TimeUnit unit) Deprecated. Since 2.2. Use close(Duration) or close().
static Admin	create(Map<String,Object> conf) Create a new Admin with the given configuration.
static Admin	create(Properties props) Create a new Admin with the given configuration.
default CreatePartitionsResult	createPartitions(Map<String,NewPartitions> newPartitions) Increase the number of partitions of the topics given as the keys of newPartitions according to the corresponding values.
CreatePartitionsResult	createPartitions(Map<String,NewPartitions> newPartitions, CreatePartitionsOptions options) Increase the number of partitions of the topics given as the keys of newPartitions according to the corresponding values.
default CreateTopicsResult	createTopics(Collection<NewTopic> newTopics) Create a batch of new topics with the default options.

CreateTopicsResult	createTopics(Collection<NewTopic> newTopics, CreateTopicsOptions options) Create a batch of new topics.
default DeleteRecordsResult	deleteRecords(Map<TopicPartition,RecordsToDelete> recordsToDelete) Delete records whose offset is smaller than the given offset of the corresponding partition.
DeleteRecordsResult	deleteRecords(Map<TopicPartition,RecordsToDelete> recordsToDelete, DeleteRecordsOptions options) Delete records whose offset is smaller than the given offset of the corresponding partition.
default DeleteTopicsResult	deleteTopics(Collection<String> topics) This is a convenience method for deleteTopics(Collection, DeleteTopicsOptions) with default options.
DeleteTopicsResult	deleteTopics(Collection<String> topics, DeleteTopicsOptions options) Delete a batch of topics.
default DescribeClusterResult	describeCluster() Get information about the nodes in the cluster, using the default options.
DescribeClusterResult	describeCluster(DescribeClusterOptions options) Get information about the nodes in the cluster.
default DescribeConfigsResult	describeConfigs(Collection<ConfigResource> resources) Get the configuration for the specified resources with the default options.
DescribeConfigsResult	describeConfigs(Collection<ConfigResource> resources, DescribeConfigsOptions options) Get the configuration for the specified resources.
default DescribeConsumerGroupsResult	describeConsumerGroups(Collection<String> groupIds) Describe some group IDs in the cluster, with the default options.
DescribeConsumerGroupsResult	describeConsumerGroups(Collection<String> groupIds, DescribeConsumerGroupsOptions options) Describe some group IDs in the cluster.
default DescribeTopicsResult	describeTopics(Collection<String> topicNames) Describe some topics in the cluster, with the default options.
DescribeTopicsResult	describeTopics(Collection<String> topicNames, DescribeTopicsOptions options) Describe some topics in the cluster.

default AlterConfigsResult	incrementalAlterConfigs(Map<ConfigResource,Collection<AlterConfigOp>> configs) Incrementally updates the configuration for the specified resources with default options.
AlterConfigsResult	incrementalAlterConfigs(Map<ConfigResource,Collection<AlterConfigOp>> configs, AlterConfigsOptions options) Incrementally update the configuration for the specified resources.
default ListConsumerGroupOffsetsResult	listConsumerGroupOffsets(String groupId) List the consumer group offsets available in the cluster with the default options.
ListConsumerGroupOffsetsResult	listConsumerGroupOffsets(String groupId, ListConsumerGroupOffsetsOptions options) List the consumer group offsets available in the cluster.
default ListTopicsResult	listTopics() List the topics available in the cluster with the default options.
ListTopicsResult	listTopics(ListTopicsOptions options) List the topics available in the cluster.
Map<MetricName,? extends Metric>	metrics() Get the metrics kept by the adminClient

> *提示*
>
> *由于Admin API方法比较多，以上列出的方法只包括提供单元测试和自测的，没列出的方法大多数应该也支持，具体待实际应用中验证。*

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