ALIBABA CLOUD

Alibaba Cloud

消息队列Kafka版

SDK参考

文档版本：20220708

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.SDK概述

本文介绍消息队列Kafka版SDK、支持的多语言SDK。

SDK简介

SDK（Software Development Kit）即软件开发工具包，包含示例Demo、库文件、编译工具链以及编译脚本等，不需要开发者进行任何其他的配置，直接可以在SDK对应目录环境下，进行开发、编译操作，方便开发者使用。

SDK列表

消息队列Kafka版提供了以下编程语言的SDK，您可以在获取地址中查看更新历史、获取安装包以及查看指导文档。

SDK	Demo地址	适用的协议	参考文档	说明文件
Java SDK	Java SDK Demo	• PLAINTEXT • SASL_SSL/PLAIN • SASL_PLAINTEXT/PLAIN • SASL_PLAINTEXT/SCRAM	使用实例接入点收发消息	README.md
Python SDK	Python SDK Demo	• PLAINTEXT • SASL_SSL/PLAIN	Python SDK收发消息	• VPC README.md • 公网README.md
C++ SDK	C++ SDK Demo	• PLAINTEXT • SASL_SSL/PLAIN	C++ SDK收发消息	• VPC README.md • 公网README.md
Go SDK	Go SDK Demo	• PLAINTEXT • SASL_SSL/PLAIN • SASL_PLAINTEXT/PLAIN • SASL_PLAINTEXT/SCRAM	Go SDK收发消息	README.md
PHP SDK	PHP SDK Demo	• PLAINTEXT • SASL_SSL/PLAIN	PHP SDK收发消息	• VPC README.md • 公网README.md
Ruby SDK	Ruby SDK Demo	• PLAINTEXT • SASL_SSL/PLAIN	Ruby SDK收发消息	README.md
Node.js SDK	Node.js SDK Demo	• PLAINTEXT • SASL_PLAIN	Node.js SDK收发消息	README.md

SDK	Demo地址	适用的协议	参考文档	说明文件
C# SDK	无	• PLAINTEXT • SASL_SSL/PLAIN • SASL_PLAINTEXT/PLAIN • SASL_PLAINTEXT/SCRAM	C# SDK收发消息	README.md

SDK说明

实例接入点说明

编程语言的客户端可以通过消息队列Kafka版提供的多种接入点接入并收发消息。

- **默认接入点**：通过默认接入点接入消息队列Kafka版并收发消息。
- **SSL接入点**：通过SSL接入点接入消息队列Kafka版并使用PLAIN机制收发消息。PLAIN机制是一种简单的用户名密码校验机制。消息队列Kafka版优化了PLAIN机制，支持不重启实例的情况下动态增加SASL用户。

  > **注意** 若您已部署实例且实例的SSL证书算法位数为1024，当您有更高的安全需求时，您可以升级实例的SSL证书算法位数至4096。详细操作，请参见SSL证书算法升级说明。

- **SASL接入点**：在VPC环境下通过SASL接入点接入消息队列Kafka版并使用PLAIN机制或者SCRAM机制收发消息。SASL支持两种机制验证身份：
  - PLAIN机制：一种简单的用户名密码校验机制。消息队列Kafka版的PLAIN机制，支持不重启实例的情况下动态增加SASL用户。
  - SCRAM-SHA-256：一种在服务端和客户端采用哈希算法对用户名与密码进行身份校验的安全认证机制。消息队列Kafka版使用SCRAM-SHA-256加密算法实现身份校验，比PLAIN机制安全性更高，同样支持不重启实例的情况下动态增加SASL用户。

关于接入点的详细信息，请参见接入点对比。

Demo使用说明

根据购买的实例信息选择Demo文件，通过配置后运行文件收发消息。具体操作，请参见对应语言页面。

相关链接

- 根据接入消息队列Kafka版网络类型，购买并部署消息队列Kafka版实例。具体操作，请参见VPC接入和公网和VPC接入。
- 创建收发消息使用的Topic和Group。具体操作，请参见步骤三：创建资源。
- 公网/VPC实例的默认SASL用户仅提供身份校验，支持所有Topic和Group的读写权限。如果需要更细致的权限控制，您需开启ACL，创建SASL用户，按需赋予SASL用户向消息队列Kafka版收发消息的权限。开启ACL之后，默认的SASL用户权限将失效。具体操作，请参见SASL用户授权。

2.SSL证书算法升级说明

您可以根据安全需求调整您的实例SSL证书算法位数。

前提条件

您已购买消息队列Kafka版公网实例，且实例处于服务中状态。

背景信息

您的实例开启公网时，会启动SSL相关的端口，在消息队列Kafka版控制台的实例配置信息页面可以看到SSL证书算法位数。请根据您的安全需求按需选择是否升级，如果需要升级可以通过以下方案将实例的SSL证书算法位数升级至4096。

> **注意** 仅在消息队列Kafka版的实例配置信息页面修改SSL证书算法位数，将会导致客户端无法使用，升级前请按照操作步骤下载新版证书，修改客户端证书配置后重启。

SSL证书下载

- 实例尚未被部署：Java语言请下载only.4096.client.truststore.jks，其他语言请在SDK列表中下载对应语言的only-4096-ca-cert。
- 实例已部署且实例的SSL证书算法位数为1024：Java语言请下载kafka.client.truststore.jks，其他语言请在SDK列表中下载对应语言的ca-cert.pem。
- 实例已部署且实例的SSL证书算法位数需要从1024升级至4096：Java语言请下载mix.4096.client.truststore.jks，其他语言请在SDK列表中下载对应语言的mix-4096-ca-cert。mix.4096.client.truststore.jks和mix-4096-ca-cert同时包含1024位和4096位的SSL证书，无论服务端是1024位还是4096位，都可以正常使用。

操作步骤

1. 根据您的客户端语言类型，下载对应的4096位算法的SSL证书。下载链接，请参见SSL证书下载。
2. 将下载的证书替换到您的客户端，然后重启客户端。
3. 在实例配置页面修改SSL证书算法位数为4096。详细操作，请参见变更消息配置。

3.Java SDK

3.1. 使用实例接入点收发消息

本文介绍如何使用Java SDK通过接入点接入消息队列Kafka版并收发消息。

环境准备

- 安装1.8或以上版本JDK。具体操作。请参见安装JDK。
- 安装2.5或以上版本Maven。具体操作，请参见安装Maven。

安装Java依赖库

1. 在*pom.xml*中添加以下依赖。

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>2.4.0</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.7.6</version>
</dependency>
```

> **说明** 建议您保持服务端和客户端版本一致，即保持客户端库版本和消息队列Kafka版实例的大版本一致。您可以在消息队列Kafka版控制台的**实例详情**页面获取消息队列Kafka版实例的大版本。

准备配置

1. （可选）下载SSL根证书。如果是SSL接入点，需下载该证书。
2. 访问aliware-kafka-demos，单击 下载 ZIP，下载Demo工程到本地并解压。
3. 在解压的Demo工程中，找到*kafka-java-demo*文件夹，将此文件导入IntelliJ IDEA。
4. （可选）如果是SSL接入点或者SASL接入点实例，修改配置文件kafka_client_jaas.conf。

```
KafkaClient {
  org.apache.kafka.common.security.plain.PlainLoginModule required
  username="xxxx"
  password="xxxx";
};
```

username和password为实例的用户名和密码。

- 如果实例未开启ACL，您可以在消息队列Kafka版控制台的**实例详情**页面**配置信息**区域获取默认用户的用户名和密码。
- 如果实例已开启ACL，请确保要使用的SASL用户为PLAIN类型且已授权收发消息的权限，详情请参见SASL用户授权。

5. 修改配置文件kafka.properties。

```
##==============================通用配置参数==============================
bootstrap.servers=xxxxxxxxxxxxxxxxxxxx
topic=xxx
group.id=xxx
##=======================以下内容请根据实际情况配置=======================
##SSL接入点配置
ssl.truststore.location=/xxxx/kafka.client.truststore.jks
java.security.auth.login.config=/xxxx/kafka_client_jaas.conf
##SASL接入点PLAIN机制配置
java.security.auth.login.config.plain=/xxxx/kafka_client_jaas_plain.conf
##SASL接入点SCRAM机制配置
java.security.auth.login.config.scram=/xxxx/kafka_client_jaas_scram.conf
```

参数	描述
bootstrap.servers	接入点信息。您可在消息队列Kafka版控制台的**实例详情**页面的**接入点信息**区域获取。
topic	实例的Topic名称。您可在消息队列Kafka版控制台的**Topic 管理**页面获取。
group.id	实例的Group。您可在消息队列Kafka版控制台的**Group 管理**页面获取。 **说明** 如果应用运行producer.go发送消息，该参数可以不配置；如果应用运行consumer.go订阅消息，该参数必须配置。
ssl.truststore.location	SSL根证书的路径。其中xxxx需要根据实际情况替换。例如，*/home/doc/project/kafka-java-demo/ssl/src/main/resources/kafka.client.truststore.jks*。 **说明** 如果是默认接入点实例或者SASL接入点实例，该参数不需要配置；如果是SSL接入点实例，该参数必须配置。
kafka_client_jaas.conf	JAAS配置文件所在路径，其中xxxx需要根据实际情况替换。例如，*/home/doc/project/kafka-java-demo/ssl/src/main/resources/kafka_client_jaas_scram.conf*。 **说明** 如果是默认接入点实例，该参数不需要配置；如果是SSL接入点实例或者SASL接入点实例，该参数必须配置。

发送消息

编译并运行KafkaProducerDemo.java发送消息。

示例代码

```
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
import java.util.concurrent.Future;
//如果是SSL接入点实例或者SASL接入点实例，请注释以下第一行代码。
```

```
//如果是SSL接入点实例或者SASL接入点实例，请取消注释以下两行代码。
import java.util.concurrent.TimeUnit;
import org.apache.kafka.clients.CommonClientConfigs;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;
/*
*如果是SSL接入点实例或者SASL接入点实例，请取消注释以下两行代码。
import org.apache.kafka.common.config.SaslConfigs;
import org.apache.kafka.common.config.SslConfigs;
*/
public class KafkaProducerDemo {
    public static void main(String args[]) {
        /*
         * 如果是SSL接入点实例，请取消注释以下一行代码。
         设置JAAS配置文件的路径。
         JavaKafkaConfigurer.configureSasl();
         */
        /*
         * 如果是SASL接入点PLAIN机制实例，请取消注释以下一行代码。
         设置JAAS配置文件的路径。
         JavaKafkaConfigurer.configureSaslPlain();
         */
        /*
         * 如果是SASL接入点SCRAM机制实例，请取消注释以下一行代码。
         设置JAAS配置文件的路径。
         JavaKafkaConfigurer.configureSaslScram();
         */
         //加载kafka.properties。
         Properties kafkaProperties =  JavaKafkaConfigurer.getKafkaProperties();
         Properties props = new Properties();
         //设置接入点，请通过控制台获取对应Topic的接入点。
         props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getProperty("boo
tstrap.servers"));
        /*
         * 如果是SSL接入点实例，请取消注释以下四行代码。
         * 与sasl路径类似，该文件也不能被打包到jar中。
         props.put(SslConfigs.SSL_TRUSTSTORE_LOCATION_CONFIG, kafkaProperties.getProperty("s
sl.truststore.location"));
         * 根证书store的密码，保持不变。
         props.put(SslConfigs.SSL_TRUSTSTORE_PASSWORD_CONFIG, "KafkaOnsClient");
         * 接入协议，目前支持使用SASL_SSL协议接入。
         props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_SSL");
         * SASL鉴权方式，保持不变。
         props.put(SaslConfigs.SASL_MECHANISM, "PLAIN");
         */
        /*
         * 如果是SASL接入点PLAIN机制实例，请取消注释以下两行代码。
         * 接入协议。
         props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
         * Plain方式。
         props.put(SaslConfigs.SASL_MECHANISM, "PLAIN");
         */
        /*
```

```
        * 如果是SASL接入点SCRAM机制实例，请取消注释以下两行代码。
        * 接入协议。
        props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
        * SCRAM方式。
        props.put(SaslConfigs.SASL_MECHANISM, "SCRAM-SHA-256");
        */
        //消息队列Kafka版消息的序列化方式。
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.seri
alization.StringSerializer");
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.se
rialization.StringSerializer");
        //请求的最长等待时间。
        props.put(ProducerConfig.MAX_BLOCK_MS_CONFIG, 30 * 1000);
        //设置客户端内部重试次数。
        props.put(ProducerConfig.RETRIES_CONFIG, 5);
        //设置客户端内部重试间隔。
        props.put(ProducerConfig.RETRY_BACKOFF_MS_CONFIG, 3000);
       /*
        * 如果是SSL接入点实例或，请取消注释以下一行代码。
        * Hostname校验改成空。
        props.put(SslConfigs.SSL_ENDPOINT_IDENTIFICATION_ALGORITHM_CONFIG, "");
        */
        //构造Producer对象，注意，该对象是线程安全的，一般来说，一个进程内一个Producer对象即可。
        //如果想提高性能，可以多构造几个对象，但不要太多，最好不要超过5个。
        KafkaProducer<String, String> producer = new KafkaProducer<String, String>(props);
        //构造一个消息队列Kafka版消息。
        String topic = kafkaProperties.getProperty("topic"); //消息所属的Topic，请在控制台申请
之后，填写在这里。
        String value = "this is the message's value"; //消息的内容。
        try {
            //批量获取Future对象可以加快速度，但注意，批量不要太大。
            List<Future<RecordMetadata>> futures = new ArrayList<Future<RecordMetadata>>(12
8);
            for (int i =0; i < 100; i++) {
                //发送消息，并获得一个Future对象。
                ProducerRecord<String, String> kafkaMessage =  new ProducerRecord<String, S
tring>(topic, value + ": " + i);
                Future<RecordMetadata> metadataFuture = producer.send(kafkaMessage);
                futures.add(metadataFuture);
            }
            producer.flush();
            for (Future<RecordMetadata> future: futures) {
                //同步获得Future对象的结果。
                try {
                    RecordMetadata recordMetadata = future.get();
                    System.out.println("Produce ok:" + recordMetadata.toString());
                } catch (Throwable t) {
                    t.printStackTrace();
                }
            }
        } catch (Exception e) {
            //客户端内部重试之后，仍然发送失败，业务要应对此类错误。
            System.out.println("error occurred");
            e.printStackTrace();
```

```
        }
    }
}
```

订阅消息

选择以下任意一种方式订阅消息。

- 单Consumer订阅消息：编译并运行KafkaConsumerDemo.java发送消息。

  示例代码

```
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.producer.ProducerConfig;
/*
* 如果是SSL接入点实例，请取消注释以下三行代码；如果是SASL接入点，请取消注释以下前两行代码。
import org.apache.kafka.clients.CommonClientConfigs;
import org.apache.kafka.common.config.SaslConfigs;
import org.apache.kafka.common.config.SslConfigs;
*/
public class KafkaConsumerDemo {
    public static void main(String args[]) {
        //设置JAAS配置文件的路径。
        /*
         * 如果是SSL接入点实例，请取消注释以下一行代码。
        JavaKafkaConfigurer.configureSasl();
         */
        /*
         * 如果是SASL接入点PLAIN机制实例，请取消注释以下一行代码。
        JavaKafkaConfigurer.configureSaslPlain();
         */
        /*
        * 如果是SASL接入点SCRAM机制实例，请取消注释以下一行代码。
        JavaKafkaConfigurer.configureSaslScram();
        */
        //加载kafka.properties
        Properties kafkaProperties =  JavaKafkaConfigurer.getKafkaProperties();
        Properties props = new Properties();
        //设置接入点，请通过控制台获取对应Topic的接入点。
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getProperty("b
ootstrap.servers"));
        //如果是SSL接入点实例，请注释以下第一行代码。
        //可更加实际拉去数据和客户的版本等设置此值，默认30s。
        props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
        /*
         * 如果是SSL接入点实例，请取消注释以下六行代码。
         * 设置SSL根证书的路径，请记得将XXX修改为自己的路径。
         * 与SASL路径类似，该文件也不能被打包到jar中。
         props.put(SslConfigs.SSL_TRUSTSTORE_LOCATION_CONFIG, kafkaProperties.getProperty
```

```
("ssl.truststore.location"));
         * 根证书存储的密码，保持不变。
         props.put(SslConfigs.SSL_TRUSTSTORE_PASSWORD_CONFIG, "KafkaOnsClient");
         * 接入协议，目前支持使用SASL_SSL协议接入。
         props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_SSL");
         * SASL鉴权方式，保持不变。
         props.put(SaslConfigs.SASL_MECHANISM, "PLAIN");
         * 两次Poll之间的最大允许间隔。
         * 消费者超过该值没有返回心跳，服务端判断消费者处于非存活状态，服务端将消费者从Group移除并触发Rebalance，默认30s。
         props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
         * 设置单次拉取的量，走公网访问时，该参数会有较大影响。
         props.put(ConsumerConfig.MAX_PARTITION_FETCH_BYTES_CONFIG, 32000);
         props.put(ConsumerConfig.FETCH_MAX_BYTES_CONFIG, 32000);
         */
        //如果是SASL接入点PLAIN机制实例，请注释以下一行代码。
       //可更加实际拉去数据和客户的版本等设置此值，默认30s。
        props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
        /*
         * 如果是SASL接入点PLAIN机制实例，请取消注释以下三行代码。
         * 接入协议。
         props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
         * Plain方式。
         props.put(SaslConfigs.SASL_MECHANISM, "PLAIN");
         * 两次Poll之间的最大允许间隔。
         * 消费者超过该值没有返回心跳，服务端判断消费者处于非存活状态，服务端将消费者从Group移除并触发Rebalance，默认30s。
         props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
         */
        //如果是SASL接入点SCRAM机制实例，请注释以下一行代码。
       //可更加实际拉去数据和客户的版本等设置此值，默认30s
        props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
        /*
         * 如果是SASL接入点SCRAM机制实例，请取消注释以下四行代码。
         * 接入协议。
         props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
         * SCRAM方式。
         props.put(SaslConfigs.SASL_MECHANISM, "SCRAM-SHA-256");
         * 两次Poll之间的最大允许间隔。
         * 消费者超过该值没有返回心跳，服务端判断消费者处于非存活状态，服务端将消费者从Group移除并触发Rebalance，默认30s。
         props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
         props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
         */
        //每次poll的最大数量。
        //注意该值不要改得太大，如果poll太多数据，而不能在下次poll之前消费完，则会触发一次负载均衡，产生卡顿。
        props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 30);
        //消息的反序列化方式
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.StringDeserializer");
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.StringDeserializer");
        //当前消费实例所属的消费组，请在控制台申请之后填写。
```

```
        //属于同一个组的消费实例，会负载消费消息。
        props.put(ConsumerConfig.GROUP_ID_CONFIG, kafkaProperties.getProperty("group.id")
);
        //如果是SSL接入点实例，请取消注释以下一行代码。
        //Hostname校验改成空。
        //props.put(SslConfigs.SSL_ENDPOINT_IDENTIFICATION_ALGORITHM_CONFIG, "");
        //构造消息对象，也即生成一个消费实例。
        KafkaConsumer<String, String> consumer = new org.apache.kafka.clients.consumer.Ka
fkaConsumer<String, String>(props);
        //设置消费组订阅的Topic，可以订阅多个。
        //如果GROUP_ID_CONFIG是一样，则订阅的Topic也建议设置成一样。
        List<String> subscribedTopics =  new ArrayList<String>();
        //如果是SSL接入点实例，请注释以下前五行代码，取消注释第六行代码。
        //如果需要订阅多个Topic，则在这里add进去即可。
        //每个Topic需要先在控制台进行创建。
        String topicStr = kafkaProperties.getProperty("topic");
        String[] topics = topicStr.split(",");
        for (String topic: topics) {
            subscribedTopics.add(topic.trim());
        }
        //subscribedTopics.add(kafkaProperties.getProperty("topic"));
        consumer.subscribe(subscribedTopics);
        //循环消费消息。
        while (true){
            try {
                ConsumerRecords<String, String> records = consumer.poll(1000);
                //必须在下次poll之前消费完这些数据，且总耗时不得超过SESSION_TIMEOUT_MS_CONFIG。
                //建议开一个单独的线程池来消费消息，然后异步返回结果。
                for (ConsumerRecord<String, String> record : records) {
                    System.out.println(String.format("Consume partition:%d offset:%d", re
cord.partition(), record.offset()));
                }
            } catch (Exception e) {
                try {
                    Thread.sleep(1000);
                } catch (Throwable ignore) {
                }
                //参考常见报错：使用消息队列Kafka版时客户端的报错及解决方案
                e.printStackTrace();
            }
        }
    }
}
```

- 多Consumer订阅消息：编译并运行KafkaMultiConsumerDemo.java消费消息。

  示例代码

```
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
import java.util.concurrent.atomic.AtomicBoolean;
//如果是SSL接入点实例或者SASL接入点实例，请取消注释以下第一行代码。
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecord;
```

```
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.producer.ProducerConfig;
/*
* 如果是SSL接入点实例，请取消注释以下前三行代码；如果是SASL接入点实例，请取消注释以下前两行代码。
import org.apache.kafka.clients.CommonClientConfigs;
import org.apache.kafka.common.config.SaslConfigs;
import org.apache.kafka.common.config.SslConfigs;
*/
import org.apache.kafka.common.errors.WakeupException;
/**
 * 本教程演示如何在一个进程内开启多个Consumer同时消费Topic。
 * 注意全局Consumer数量不要超过订阅的Topic总分区数。
 */
public class KafkaMultiConsumerDemo {
    public static void main(String args[]) throws InterruptedException {
        //设置JAAS配置文件的路径。
        /*
         * 如果是SSL接入点实例，请取消注释以下一行代码。
         JavaKafkaConfigurer.configureSasl();
         */
        /*
         * 如果是SASL接入点PLAIN机制实例，请取消注释以下一行代码。
         JavaKafkaConfigurer.configureSaslPlain();
         */
        /*
         * 如果是SASL接入点SCRAM机制实例，请取消注释以下一行代码。
         JavaKafkaConfigurer.configureSaslScram();
         */
        //加载kafka.properties。
        Properties kafkaProperties = JavaKafkaConfigurer.getKafkaProperties();
        Properties props = new Properties();
        //设置接入点，请通过控制台获取对应Topic的接入点。
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getProperty("b
ootstrap.servers"));
        /*
         * 如果是SSL接入点实例，请取消注释以下四行代码。
         * 与SASL路径类似，该文件也不能被打包到JAR中。
         props.put(SslConfigs.SSL_TRUSTSTORE_LOCATION_CONFIG, kafkaProperties.getProperty
("ssl.truststore.location"));
         * 根证书存储的密码，保持不变。
         props.put(SslConfigs.SSL_TRUSTSTORE_PASSWORD_CONFIG, "KafkaOnsClient");
         * 接入协议，目前支持使用SASL_SSL协议接入。
         props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_SSL");
         * SASL鉴权方式，保持不变。
         props.put(SaslConfigs.SASL_MECHANISM, "PLAIN");
         */
        /*
         * 如果是SASL接入点PLAIN机制实例，请取消注释以下两行代码。
         * 接入协议。
         props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
         * Plain方式。
         props.put(SaslConfigs.SASL_MECHANISM, "PLAIN");
         */
```

```
        /*
         * 如果是SASL接入点SCRAM机制实例，请取消注释以下两行代码。
         * 接入协议。
         props.put(CommonClientConfigs.SECURITY_PROTOCOL_CONFIG, "SASL_PLAINTEXT");
         * SCRAM方式。
         props.put(SaslConfigs.SASL_MECHANISM, "SCRAM-SHA-256");
         */
        //两次Poll之间的最大允许间隔。
        //消费者超过该值没有返回心跳，服务端判断消费者处于非存活状态，服务端将消费者从Group移除并触发Rebalance，默认30s。
        props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 30000);
        //每次Poll的最大数量。
        //注意该值不要改得太大，如果Poll太多数据，而不能在下次Poll之前消费完，则会触发一次负载均衡，产生卡顿。
        props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 30);
        //消息的反序列化方式。
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.StringDeserializer");
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.StringDeserializer");
        //当前消费实例所属的消费组，请在控制台申请之后填写。
        //属于同一个组的消费实例，会负载消费消息。
        props.put(ConsumerConfig.GROUP_ID_CONFIG, kafkaProperties.getProperty("group.id"));
        /*
         * 如果是SSL接入点实例，请取消注释以下一行代码。
         * 构造消费对象，也即生成一个消费实例。
         * Hostname校验改成空。
         props.put(SslConfigs.SSL_ENDPOINT_IDENTIFICATION_ALGORITHM_CONFIG, "");
         */
        int consumerNum = 2;
        Thread[] consumerThreads = new Thread[consumerNum];
        for (int i = 0; i < consumerNum; i++) {
            KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>(props);
            List<String> subscribedTopics = new ArrayList<String>();
            subscribedTopics.add(kafkaProperties.getProperty("topic"));
            consumer.subscribe(subscribedTopics);
            KafkaConsumerRunner kafkaConsumerRunner = new KafkaConsumerRunner(consumer);
            consumerThreads[i] = new Thread(kafkaConsumerRunner);
        }
        for (int i = 0; i < consumerNum; i++) {
            consumerThreads[i].start();
        }
        for (int i = 0; i < consumerNum; i++) {
            consumerThreads[i].join();
        }
    }
    static class KafkaConsumerRunner implements Runnable {
        private final AtomicBoolean closed = new AtomicBoolean(false);
        private final KafkaConsumer consumer;
        KafkaConsumerRunner(KafkaConsumer consumer) {
            this.consumer = consumer;
        }
```

```
        @Override
        public void run() {
            try {
                while (!closed.get()) {
                    try {
                        ConsumerRecords<String, String> records = consumer.poll(1000);
                        //必须在下次Poll之前消费完这些数据，且总耗时不得超过SESSION_TIMEOUT_MS_
CONFIG。
                        for (ConsumerRecord<String, String> record : records) {
                            System.out.println(String.format("Thread:%s Consume partition
:%d offset:%d", Thread.currentThread().getName(), record.partition(), record.offset()));
                        }
                    } catch (Exception e) {
                        try {
                            Thread.sleep(1000);
                        } catch (Throwable ignore) {
                        }
                        e.printStackTrace();
                    }
                }
            } catch (WakeupException e) {
                //如果关闭则忽略异常。
                if (!closed.get()) {
                    throw e;
                }
            } finally {
                consumer.close();
            }
        }
        //可以被另一个线程调用的关闭Hook。
        public void shutdown() {
            closed.set(true);
            consumer.wakeup();
        }
    }
}
```

3.2. 使用Spring Cloud框架收发消息

本文介绍如何使用Spring Cloud框架接入消息队列Kafka版并收发消息。

背景信息

Spring Cloud是用于构建消息驱动的微服务应用程序的框架。详细信息，请参见Spring Cloud Stream。

前提条件

- 安装1.8或以上版本JDK。
- 安装2.5或以上版本Maven。
- 下载kafka-spring-stream-demo，并将其上传在准备好的Linux操作系统。
- 确保您的消息队列Kafka版实例为2.x或以上版本。具体操作，请参见升级实例版本。

公网环境（消息传输需鉴权与加密）

公网环境，消息采用SASL_SSL协议进行鉴权并加密。客户端通过SSL接入点访问消息队列Kafka版。接入点的详细信息，请参见接入点对比。

本示例将Demo包上传在*/home/doc/project/aliware-kafka-demos/kafka-spring-stream-demo*路径。

1. 登录Linux系统，执行以下命令，进入Demo包所在路径*/home/doc/project/aliware-kafka-demos/kafka-spring-stream-demo*。

```
cd /home/doc/project/aliware-kafka-demos/kafka-spring-stream-demo
```

2. 执行以下命令，进入配置文件路径。

```
cd sasl-ssl/src/main/resources/
```

3. 执行以下命令，编辑*application.properties*文件，并根据参数列表配置实例信息。

```
vi application.properties
```

```
##以下参数，您需配置为实际使用的实例信息。
kafka.bootstrap-servers=alikafka-pre-cn-zv**********-1.alikafka.aliyuncs.com:9093,alika
fka-pre-cn-zv**********-2.alikafka.aliyuncs.com:9093,alikafka-pre-cn-zv**********-3.ali
kafka.aliyuncs.com:9093
kafka.consumer.group=test-spring
kafka.output.topic.name=test-output
kafka.input.topic.name=test-input
kafka.ssl.truststore.location=/home/doc/project/aliware-kafka-demos/kafka-spring-stream
-demo/sasl-ssl/src/main/resources/kafka.client.truststore.jks
### 配置Binding参数可以把消息队列Kafka版和Spring Cloud Stream的Binder绑定在一起，以下参数保持默认即可。
spring.cloud.stream.bindings.MyOutput.destination=${kafka.output.topic.name}
spring.cloud.stream.bindings.MyOutput.contentType=text/plain
spring.cloud.stream.bindings.MyInput.group=${kafka.consumer.group}
spring.cloud.stream.bindings.MyInput.destination=${kafka.input.topic.name}
spring.cloud.stream.bindings.MyInput.contentType=text/plain
### Binder是Spring Cloud对消息中间件的封装模块，以下参数保持默认即可。
spring.cloud.stream.kafka.binder.autoCreateTopics=false
spring.cloud.stream.kafka.binder.brokers=${kafka.bootstrap-servers}
spring.cloud.stream.kafka.binder.configuration.security.protocol=SASL_SSL
spring.cloud.stream.kafka.binder.configuration.sasl.mechanism=PLAIN
spring.cloud.stream.kafka.binder.configuration.ssl.truststore.location=${kafka.ssl.trus
tstore.location}
spring.cloud.stream.kafka.binder.configuration.ssl.truststore.password=KafkaOnsCl****
### 如果Demo中没有以下参数，请手动增加。该参数表示是否需要进行服务器主机名验证。因消息传输使用SASL身份校验，可设置为空字符串关闭服务器主机名验证。
### 服务器主机名验证是验证SSL证书的主机名与服务器的主机名是否匹配，默认为HTTPS。
spring.cloud.stream.kafka.binder.configuration.ssl.endpoint.identification.algorithm=
```

参数列表

参数	描述
kafka.bootstrap-servers	消息队列Kafka版实例接入点。您可在消息队列Kafka版控制台的**实例详情**页面的**接入点信息**区域获取。

参数	描述
kafka.consumer.group	订阅消息的Group。您可以在消息队列Kafka版控制台的**Group 管理**页面创建。具体操作，请参见步骤三：创建资源。
kafka.output.topic.name	消息的Topic。控制台程序通过此Topic每隔一段时间发送消息，内容是固定的。您可以在消息队列Kafka版控制台的**Topic 管理**页面创建。具体操作，请参见步骤三：创建资源。
kafka.input.topic.name	消息的Topic。您可以通过此Topic在控制台发送消息，Demo程序会消费消息，并将消息打印在日志中。
kafka.ssl.truststore.location	SSL根证书kafka.client.truststore.jks的存放路径。

4. 执行以下命令，打开*kafka_client_jaas.conf*文件，配置实例的用户名与密码。

```
vi kafka_client_jaas.conf
```

> **说明**
> - 如果实例未开启ACL，您可以在消息队列Kafka版控制台的**实例详情**页面获取默认用户的用户名和密码。
> - 如果实例已开启ACL，请确保要使用的SASL用户为PLAIN类型且已授权收发消息的权限。具体信息，请参见SASL用户授权。

```
KafkaClient {
  org.apache.kafka.common.security.plain.PlainLoginModule required
  username="XXX"
  password="XXX";
};
```

5. 进入*/home/doc/project/aliware-kafka-demos/kafka-spring-stream-demo/sasl-ss*路径，执行以下命令，运行Demo。

```
sh run_demo.sh
```

程序打印如下信息，说明接收到控制台程序通过kafka.output.topic.name配置的Topic所发送的消息。

```
Send: hello world !!
Send: hello world !!
Send: hello world !!
Send: hello world !!
```

6. 登录消息队列Kafka版控制台验证消息收发是否成功。
   - 查询kafka.output.topic.name配置的Topic是否接收到控制台程序发送的消息。具体操作，请参见查询消息。
   - 在kafka.input.topic.name配置的Topic发送消息，查看Demo程序日志中是否会打印消息。具体操作，请参见发送消息。

VPC环境（消息传输不鉴权不加密）

VPC环境，消息可以采用PLAINTEXT协议不鉴权不加密传输。客户端通过默认接入点访问消息队列Kafka版。接入点的详细信息，请参见接入点对比。

本示例将Demo包上传在*/home/doc/project/aliware-kafka-demos/kafka-spring-stream-demo*路径。

1. 登录Linux系统，执行以下命令，进入Demo包所在路径*/home/doc/project/aliware-kafka-demos/kafka-spring-stream-demo*。

   ```
   cd /home/doc/project/aliware-kafka-demos/kafka-spring-stream-demo
   ```

2. 执行以下命令，进入配置文件路径。

   ```
   cd vpc/src/main/resources/
   ```

3. 执行以下命令，编辑*application.properties*文件，并根据参数列表配置实例信息。

   ```
   vi application.properties
   ```

   ```
   ###以下参数请修改为实际使用的实例的信息。
   kafka.bootstrap-servers=alikafka-pre-cn-zv**********-1-vpc.alikafka.aliyuncs.com:9092,alikafka-pre-cn-zv**********-2-vpc.alikafka.aliyuncs.com:9092,alikafka-pre-cn-zv**********-3-vpc.alikafka.aliyuncs.com:9092
   kafka.consumer.group=test-spring
   kafka.output.topic.name=test-output
   kafka.input.topic.name=test-input
   ```

4. 进入*/home/doc/project/aliware-kafka-demos/kafka-spring-stream-demo/vpc*路径，执行以下命令，运行Demo。

   ```
   sh run_demo.sh
   ```

   程序打印如下信息。

   ```
   Send: hello world !!
   Send: hello world !!
   Send: hello world !!
   Send: hello world !!
   ```

5. 登录消息队列Kafka版控制台验证消息收发是否成功。
   - 查询kafka.output.topic.name配置的Topic是否接收到控制台程序发送的消息。具体操作，请参见查询消息。
   - 在kafka.input.topic.name配置的Topic发送消息，查看Demo程序日志中是否会打印消息。具体操作，请参见发送消息。

4.Python SDK收发消息

本文介绍如何使用Python SDK通过接入点接入消息队列Kafka版并收发消息。

环境准备

- 安装Python

  > **说明** Python版本为2.7、3.4、3.5、3.6或3.7。

- 安装pip

安装Confluent-kafka

1. 执行以下命令安装Confluent-kafka。

```
pip install confluent-kafka
```

准备配置

1. （可选）下载SSL根证书。如果是SSL接入点，需下载该证书。
2. 访问aliware-kafka-demos，单击 Code 图标，然后在下拉框选择**Download ZIP**，下载Demo包并解压。
3. 在解压的Demo工程中，找到*kafka-confluent-python-demo*文件夹，将此文件夹上传在Linux系统。
4. 登录Linux系统，进入文件目录，修改配置文件kafka.properties。

```
kafka_setting = {
    'sasl_plain_username': 'XXX',    #如果是默认接入点实例，请删除该配置。
    'sasl_plain_password': 'XXX',    #如果是默认接入点实例，请删除该配置。
    'bootstrap_servers': ["XXX", "XXX", "XXX"],
    'topic_name': 'XXX',
    'consumer_id': 'XXX'
}
```

参数	描述
sasl_plain_username	SASL用户名。 **说明** ○ 如果实例未开启ACL，您可以在消息队列Kafka版控制台的**实例详情**页面的**配置信息**区域获取默认的**用户名**和**密码**。 ○ 如果实例已开启ACL，请确保要使用的SASL用户已被授予向消息队列Kafka版实例收发消息的权限。具体操作，请参见SASL用户授权。
sasl_plain_password	SASL用户名密码。
bootstrap_servers	SSL接入点。您可在消息队列Kafka版控制台的**实例详情**页面的**接入点信息**区域获取。

参数	描述
topic_name	Topic名称。您可在消息队列Kafka版控制台的**Topic 管理**页面获取。
consumer_id	Group名称。您可在消息队列Kafka版控制台的**Group 管理**页面获取。

发送消息

执行以下命令发送消息。

```
python aliyun_kafka_producer.py
```

消息程序aliyun_kafka_producer.py示例代码如下：

- 默认接入点

```
#!/usr/bin/env python
# encoding: utf-8
import socket
from kafka import KafkaProducer
from kafka.errors import KafkaError
import setting
conf = setting.kafka_setting
print conf
producer = KafkaProducer(bootstrap_servers=conf['bootstrap_servers'],
                         api_version = (0,10),
                         retries=5)
partitions = producer.partitions_for(conf['topic_name'])
print 'Topic下分区：%s' % partitions
try:
    future = producer.send(conf['topic_name'], 'hello aliyun-kafka!')
    future.get()
    print 'send message succeed.'
except KafkaError, e:
    print 'send message failed.'
    print e
```

- SSL接入点

```
#!/usr/bin/env python
# encoding: utf-8
import ssl
import socket
from kafka import KafkaProducer
from kafka.errors import KafkaError
import setting
conf = setting.kafka_setting
print conf
context = ssl.create_default_context()
context = ssl.SSLContext(ssl.PROTOCOL_SSLv23)
context.verify_mode = ssl.CERT_REQUIRED
# context.check_hostname = True
context.load_verify_locations("ca-cert")
producer = KafkaProducer(bootstrap_servers=conf['bootstrap_servers'],
                         sasl_mechanism="PLAIN",
                         ssl_context=context,
                         security_protocol='SASL_SSL',
                         api_version = (0,10),
                         retries=5,
                         sasl_plain_username=conf['sasl_plain_username'],
                         sasl_plain_password=conf['sasl_plain_password'])
partitions = producer.partitions_for(conf['topic_name'])
print 'Topic下分区：%s' % partitions
try:
    future = producer.send(conf['topic_name'], 'hello aliyun-kafka!')
    future.get()
    print 'send message succeed.'
except KafkaError, e:
    print 'send message failed.'
    print e
```

订阅消息

执行以下命令订阅消息。

```
python aliyun_kafka_consumer.py
```

消息程序aliyun_kafka_consumer.py示例代码如下：

- 默认接入点

```
#!/usr/bin/env python
# encoding: utf-8
import socket
from kafka import KafkaConsumer
from kafka.errors import KafkaError
import setting
conf = setting.kafka_setting
consumer = KafkaConsumer(bootstrap_servers=conf['bootstrap_servers'],
                         group_id=conf['consumer_id'],
                         api_version = (0,10,2),
                         session_timeout_ms=25000,
                         max_poll_records=100,
                         fetch_max_bytes=1 * 1024 * 1024)
print 'consumer start to consuming...'
consumer.subscribe((conf['topic_name'], ))
for message in consumer:
    print message.topic, message.offset, message.key, message.value, message.value, messa
ge.partition
```

- SSL接入点

```
#!/usr/bin/env python
# encoding: utf-8
import ssl
import socket
from kafka import KafkaConsumer
from kafka.errors import KafkaError
import setting
conf = setting.kafka_setting
context = ssl.create_default_context()
context = ssl.SSLContext(ssl.PROTOCOL_SSLv23)
context.verify_mode = ssl.CERT_REQUIRED
# context.check_hostname = True
context.load_verify_locations("ca-cert")
consumer = KafkaConsumer(bootstrap_servers=conf['bootstrap_servers'],
                         group_id=conf['consumer_id'],
                         sasl_mechanism="PLAIN",
                         ssl_context=context,
                         security_protocol='SASL_SSL',
                         api_version = (0,10),
                         sasl_plain_username=conf['sasl_plain_username'],
                         sasl_plain_password=conf['sasl_plain_password'])
print 'consumer start to consuming...'
consumer.subscribe((conf['topic_name'], ))
for message in consumer:
    print message.topic, message.offset, message.key, message.value, message.value, messa
ge.partition
```

5.C++ SDK收发消息

本文介绍如何使用C++ SDK通过接入点接入消息队列Kafka版并收发消息。

环境准备

您已安装GCC。更多信息，请参见安装GCC。

安装C++依赖库

1. 执行以下命令切换到yum源配置目录*/etc/yum.repos.d/*。

```
cd /etc/yum.repos.d/
```

2. 创建yum源配置文件*confluent.repo*。

```
[Confluent.dist]
name=Confluent repository (dist)
baseurl=https://packages.confluent.io/rpm/5.1/7
gpgcheck=1
gpgkey=https://packages.confluent.io/rpm/5.1/archive.key
enabled=1
[Confluent]
name=Confluent repository
baseurl=https://packages.confluent.io/rpm/5.1
gpgcheck=1
gpgkey=https://packages.confluent.io/rpm/5.1/archive.key
enabled=1
```

3. 执行以下命令安装C++依赖库。

```
yum install librdkafka-devel
```

准备配置

1. （可选）如果是SSL接入点，需执行如下操作。
   i. 下载SSL根证书。
   ii. 执行以下命令安装SSL依赖库。

   ```
   yum install openssl openssl-devel
   ```

   iii. 执行以下命令安装SASL依赖库。

   ```
   yum install cyrus-sasl{,-plain}
   ```

2. 访问aliware-kafka-demos，单击 下载 ZIP ，下载Demo工程到本地并解压。

3. 在解压的Demo工程，找到*kafka-cpp-demo*文件夹，根据接入点类型找到对应文件夹，将此文件夹中的文件（如果是SSL接入点实例，包含证书SSL根证书文件）上传至服务器。

发送消息

执行如下命令编译kafka_producer.c并发送消息。

1. 执行如下命令编译kafka_producer.c。

```
gcc -lrdkafka ./kafka_producer.c -o kafka_producer
```

2. 执行如下命令，发送消息。
   - 默认接入点实例，执行如下命令。

     ```
     ./kafka_producer <bootstrap_servers> <topic>
     ```

   - SSL接入点实例，执行如下命令。

     ```
     ./kafka_producer <bootstrap_servers> <topic> <username> <password>
     ```

参数	描述
bootstrap_servers	实例接入点。您可在消息队列Kafka版控制台的**实例详情**页面的**接入点信息**区域获取。
topic	Topic名称。您可在消息队列Kafka版控制台的**Topic管理**页面获取。
username	SASL用户名。使用SSL接入点实例时，需配置该参数。 **说明** - 如果实例未开启ACL，您可以在消息队列Kafka版控制台的**实例详情**页面的**配置信息**区域获取默认的**用户名**和**密码**。 - 如果实例已开启ACL，请确保要使用的SASL用户已被授予向消息队列Kafka版实例收发消息的权限。具体操作，请参见SASL用户授权。
password	SASL用户名密码。使用SSL接入点实例时，需配置该参数。

消息程序kafka_producer.c示例代码如下：

说明 使用代码前，请先根据接入点类型，在代码中加粗文字描述处选择对应代码，或者根据加粗文字描述注释或删除对应代码。

```
/*
 * librdkafka - Apache Kafka C library
 *
 * Copyright (c) 2017, Magnus Edenhill
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
```

```
 *    and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */
/**
 * Simple Apache Kafka producer
 * using the Kafka driver from librdkafka
 * (https://github.com/edenhill/librdkafka)
 */
#include <stdio.h>
#include <signal.h>
#include <string.h>
/* Typical include path would be <librdkafka/rdkafka.h>, but this program
 * is builtin from within the librdkafka source tree and thus differs. */
#include "librdkafka/rdkafka.h"
static int run = 1;
/**
 * @brief Signal termination of program
 */
static void stop (int sig) {
        run = 0;
        fclose(stdin); /* abort fgets() */
}
/**
 * @brief Message delivery report callback.
 *
 * This callback is called exactly once per message, indicating if
 * the message was succesfully delivered
 * (rkmessage->err == RD_KAFKA_RESP_ERR_NO_ERROR) or permanently
 * failed delivery (rkmessage->err != RD_KAFKA_RESP_ERR_NO_ERROR).
 *
 * The callback is triggered from rd_kafka_poll() and executes on
 * the application's thread.
 */
static void dr_msg_cb (rd_kafka_t *rk,
                       const rd_kafka_message_t *rkmessage, void *opaque) {
        if (rkmessage->err)
                fprintf(stderr, "%% Message delivery failed: %s\n",
                        rd_kafka_err2str(rkmessage->err));
        else
                fprintf(stderr,
                        "%% Message delivered (%zd bytes, "
                        "partition %"PRId32")\n",
                        rkmessage->len, rkmessage->partition);
```

```
        /* The rkmessage is destroyed automatically by librdkafka */
}
int main (int argc, char **argv) {
        rd_kafka_t *rk;         /* Producer instance handle */
        rd_kafka_topic_t *rkt;  /* Topic object */
        rd_kafka_conf_t *conf;  /* Temporary configuration object */
        char errstr[512];       /* librdkafka API error reporting buffer */
        char buf[512];          /* Message value temporary buffer */
        const char *brokers;    /* Argument: broker list */
        const char *topic;      /* Argument: topic to produce to */
        const char *username;      /* Argument: topic to produce to */
        const char *password;      /* Argument: topic to produce to */
        /*
         * Argument validation
         */
        //请根据实例接入点类型选择对应的if语句代码。
        //如果是SSL接入点，请使用如下代码。
        if (argc != 5) {
                fprintf(stderr, "%% Usage: %s <broker> <topic> <username> <password>\n", ar
gv[0]);
                return 1;
        }
        //如果是默认接入点，请使用如下代码。
        if (argc != 3) {
                fprintf(stderr, "%% Usage: %s <broker> <topic>\n", argv[0]);
                return 1;
        }
        brokers = argv[1];
        topic   = argv[2];
        username = argv[3];     //如果是默认接入点，请注释或者删除该行代码。
        password = argv[4];     //如果是默认接入点，请注释或者删除该行代码。
        /*
         * Create Kafka client configuration place-holder
         */
        conf = rd_kafka_conf_new();
        /* Set bootstrap broker(s) as a comma-separated list of
         * host or host:port (default port 9092).
         * librdkafka will use the bootstrap brokers to acquire the full
         * set of brokers from the cluster. */
        if (rd_kafka_conf_set(conf, "bootstrap.servers", brokers,
                              errstr, sizeof(errstr)) != RD_KAFKA_CONF_OK) {
                fprintf(stderr, "%s\n", errstr);
                return 1;
        }
        //如果是默认接入点，请注释或者删除以下九行（if语句）代码。
        if (rd_kafka_conf_set(conf, "ssl.ca.location", "ca-cert.pem", errstr, sizeof(errstr
)) != RD_KAFKA_CONF_OK
                        || rd_kafka_conf_set(conf, "security.protocol", "sasl_ssl", errstr, siz
eof(errstr)) != RD_KAFKA_CONF_OK
                        || rd_kafka_conf_set(conf, "sasl.mechanism", "PLAIN", errstr, sizeof(er
rstr)) != RD_KAFKA_CONF_OK
                        || rd_kafka_conf_set(conf, "sasl.username", username, errstr, sizeof(er
rstr)) != RD_KAFKA_CONF_OK
                        || rd_kafka_conf_set(conf, "sasl.password", password, errstr, sizeof(er
```

```
rstr)) != RD_KAFKA_CONF_OK
                    ) {
                fprintf(stderr, "%s\n", errstr);
                return -1;
        }
        /* Set the delivery report callback.
         * This callback will be called once per message to inform
         * the application if delivery succeeded or failed.
         * See dr_msg_cb() above. */
        rd_kafka_conf_set_dr_msg_cb(conf, dr_msg_cb);
        /*
         * Create producer instance.
         *
         * NOTE: rd_kafka_new() takes ownership of the conf object
         *       and the application must not reference it again after
         *       this call.
         */
        rk = rd_kafka_new(RD_KAFKA_PRODUCER, conf, errstr, sizeof(errstr));
        if (!rk) {
                fprintf(stderr,
                        "%% Failed to create new producer: %s\n", errstr);
                return 1;
        }
        /* Create topic object that will be reused for each message
         * produced.
         *
         * Both the producer instance (rd_kafka_t) and topic objects (topic_t)
         * are long-lived objects that should be reused as much as possible.
         */
        rkt = rd_kafka_topic_new(rk, topic, NULL);
        if (!rkt) {
                fprintf(stderr, "%% Failed to create topic object: %s\n",
                        rd_kafka_err2str(rd_kafka_last_error()));
                rd_kafka_destroy(rk);
                return 1;
        }
        /* Signal handler for clean shutdown */
        signal(SIGINT, stop);
        fprintf(stderr,
                "%% Type some text and hit enter to produce message\n"
                "%% Or just hit enter to only serve delivery reports\n"
                "%% Press Ctrl-C or Ctrl-D to exit\n");
        while (run && fgets(buf, sizeof(buf), stdin)) {
                size_t len = strlen(buf);
                if (buf[len-1] == '\n') /* Remove newline */
                        buf[--len] = '\0';
                if (len == 0) {
                        /* Empty line: only serve delivery reports */
                        rd_kafka_poll(rk, 0/*non-blocking */);
                        continue;
                }
                /*
                 * Send/Produce message.
                 * This is an asynchronous call, on success it will only
```

```
                 * enqueue the message on the internal producer queue.
                 * The actual delivery attempts to the broker are handled
                 * by background threads.
                 * The previously registered delivery report callback
                 * (dr_msg_cb) is used to signal back to the application
                 * when the message has been delivered (or failed).
                 */
        retry:
                if (rd_kafka_produce(
                            /* Topic object */
                            rkt,
                            /* Use builtin partitioner to select partition*/
                            RD_KAFKA_PARTITION_UA,
                            /* Make a copy of the payload. */
                            RD_KAFKA_MSG_F_COPY,
                            /* Message payload (value) and length */
                            buf, len,
                            /* Optional key and its length */
                            NULL, 0,
                            /* Message opaque, provided in
                             * delivery report callback as
                             * msg_opaque. */
                            NULL) == -1) {
                        /**
                         * Failed to *enqueue* message for producing.
                         */
                        fprintf(stderr,
                                "%% Failed to produce to topic %s: %s\n",
                                rd_kafka_topic_name(rkt),
                                rd_kafka_err2str(rd_kafka_last_error()));
                        /* Poll to handle delivery reports */
                        if (rd_kafka_last_error() ==
                            RD_KAFKA_RESP_ERR__QUEUE_FULL) {
                                /* If the internal queue is full, wait for
                                 * messages to be delivered and then retry.
                                 * The internal queue represents both
                                 * messages to be sent and messages that have
                                 * been sent or failed, awaiting their
                                 * delivery report callback to be called.
                                 *
                                 * The internal queue is limited by the
                                 * configuration property
                                 * queue.buffering.max.messages */
                                rd_kafka_poll(rk, 1000/*block for max 1000ms*/);
                                goto retry;
                        }
                } else {
                        fprintf(stderr, "%% Enqueued message (%zd bytes) "
                                "for topic %s\n",
                                len, rd_kafka_topic_name(rkt));
                }
                /* A producer application should continually serve
                 * the delivery report queue by calling rd_kafka_poll()
                 * at frequent intervals.
                 * Either put the poll call in your main loop, or in a
```

```
                 * Either put the poll call in your main loop, or in a
                 * dedicated thread, or call it after every
                 * rd_kafka_produce() call.
                 * Just make sure that rd_kafka_poll() is still called
                 * during periods where you are not producing any messages
                 * to make sure previously produced messages have their
                 * delivery report callback served (and any other callbacks
                 * you register). */
                rd_kafka_poll(rk, 0/*non-blocking*/);
        }
        /* Wait for final messages to be delivered or fail.
         * rd_kafka_flush() is an abstraction over rd_kafka_poll() which
         * waits for all messages to be delivered. */
        fprintf(stderr, "%% Flushing final messages..\n");
        rd_kafka_flush(rk, 10*1000 /* wait for max 10 seconds */);
        /* Destroy topic object */
        rd_kafka_topic_destroy(rkt);
        /* Destroy the producer instance */
        rd_kafka_destroy(rk);
        return 0;
}
```

订阅消息

编译kafka_consumer.c并订阅消息。

1. 执行以下命令编译kafka_consumer.c。

   ```
   gcc -lrdkafka ./kafka_consumer.c -o kafka_consumer
   ```

2. 根据接入点类型，选择命令执行以订阅消息。
   - 默认接入点：执行如下命令订阅消息。

     ```
     ./kafka_consumer -g <group> -b <bootstrap_servers> <topic>
     ```

   - SSL接入点：执行如下命令订阅消息。

     ```
     ./kafka_consumer -g <group> -b <bootstrap_servers> -u <username> -p <password> <topic>
     ```

参数	描述
group	Group名称。您可在消息队列Kafka版控制台的**Group 管理**页面获取。
bootstrap_servers	SSL接入点。您可在消息队列Kafka版控制台的**实例详情**页面的**接入点信息**区域获取。

参数	描述
username	SASL用户名。使用SSL接入点时，需配置该参数。 **说明** • 如果实例未开启ACL，您可以在消息队列Kafka版控制台的**实例详情**页面的**配置信息**区域获取默认的**用户名**和**密码**。 • 如果实例已开启ACL，请确保要使用的SASL用户已被授予向消息队列Kafka版实例收发消息的权限。具体操作，请参见SASL用户授权。
password	SASL用户名密码。使用SSL接入点时，需配置该参数。
topic	Topic名称。您可在消息队列Kafka版控制台的**Topic 管理**页面获取。

消息程序kafka_consumer.c示例代码如下：

说明 使用代码前，请先根据接入点类型，在代码中加粗文字描述处选择对应代码，或者根据加粗文字描述注释或删除对应代码。

```
/*
 * librdkafka - Apache Kafka C library
 *
 * Copyright (c) 2015, Magnus Edenhill
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */
/**
 * Apache Kafka high level consumer example program
 * using the Kafka driver from librdkafka
 * (https://github.com/edenhill/librdkafka)
```

```
 */
#include <ctype.h>
#include <signal.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <syslog.h>
#include <sys/time.h>
#include <errno.h>
#include <getopt.h>
/* Typical include path would be <librdkafka/rdkafka.h>, but this program
 * is builtin from within the librdkafka source tree and thus differs. */
#include "librdkafka/rdkafka.h"  /* for Kafka driver */
static int run = 1;
static rd_kafka_t *rk;
static int exit_eof = 0;
static int wait_eof = 0;  /* number of partitions awaiting EOF */
static int quiet = 0;
static     enum {
    OUTPUT_HEXDUMP,
    OUTPUT_RAW,
} output = OUTPUT_HEXDUMP;
static void stop (int sig) {
        if (!run)
                exit(1);
    run = 0;
    fclose(stdin); /* abort fgets() */
}
static void hexdump (FILE *fp, const char *name, const void *ptr, size_t len) {
    const char *p = (const char *)ptr;
    unsigned int of = 0;
    if (name)
        fprintf(fp, "%s hexdump (%zd bytes):\n", name, len);
    for (of = 0 ; of < len ; of += 16) {
        char hexen[16*3+1];
        char charen[16+1];
        int hof = 0;
        int cof = 0;
        int i;
        for (i = of ; i < (int)of + 16 && i < (int)len ; i++) {
            hof += sprintf(hexen+hof, "%02x ", p[i] & 0xff);
            cof += sprintf(charen+cof, "%c",
                      isprint((int)p[i]) ? p[i] : '.');
        }
        fprintf(fp, "%08x: %-48s %-16s\n",
            of, hexen, charen);
    }
}
/**
 * Kafka logger callback (optional)
 */
static void logger (const rd_kafka_t *rk, int level,
            const char *fac, const char *buf) {
    struct timeval tv;
```

```
    gettimeofday(&tv, NULL);
    fprintf(stdout, "%u.%03u RDKAFKA-%i-%s: %s: %s\n",
        (int)tv.tv_sec, (int)(tv.tv_usec / 1000),
        level, fac, rd_kafka_name(rk), buf);
}
/**
 * Handle and print a consumed message.
 * Internally crafted messages are also used to propagate state from
 * librdkafka to the application. The application needs to check
 * the `rkmessage->err` field for this purpose.
 */
static void msg_consume (rd_kafka_message_t *rkmessage) {
    if (rkmessage->err) {
        if (rkmessage->err == RD_KAFKA_RESP_ERR__PARTITION_EOF) {
            fprintf(stderr,
                "%% Consumer reached end of %s [%"PRId32"] "
                   "message queue at offset %"PRId64"\n",
                   rd_kafka_topic_name(rkmessage->rkt),
                   rkmessage->partition, rkmessage->offset);
            if (exit_eof && --wait_eof == 0) {
                                fprintf(stderr,
                                        "%% All partition(s) reached EOF: "
                                        "exiting\n");
                run = 0;
                        }
            return;
        }
                if (rkmessage->rkt)
                        fprintf(stderr, "%% Consume error for "
                                "topic \"%s\" [%"PRId32"] "
                                "offset %"PRId64": %s\n",
                                rd_kafka_topic_name(rkmessage->rkt),
                                rkmessage->partition,
                                rkmessage->offset,
                                rd_kafka_message_errstr(rkmessage));
                else
                        fprintf(stderr, "%% Consumer error: %s: %s\n",
                                rd_kafka_err2str(rkmessage->err),
                                rd_kafka_message_errstr(rkmessage));
                if (rkmessage->err == RD_KAFKA_RESP_ERR__UNKNOWN_PARTITION ||
                    rkmessage->err == RD_KAFKA_RESP_ERR__UNKNOWN_TOPIC)
                        run = 0;
        return;
    }
    if (!quiet)
        fprintf(stdout, "%% Message (topic %s [%"PRId32"], "
                        "offset %"PRId64", %zd bytes):\n",
                        rd_kafka_topic_name(rkmessage->rkt),
                        rkmessage->partition,
            rkmessage->offset, rkmessage->len);
    if (rkmessage->key_len) {
        if (output == OUTPUT_HEXDUMP)
            hexdump(stdout, "Message Key",
                rkmessage->key, rkmessage->key_len);
        else
```

```
            printf("Key: %.*s\n",
                   (int)rkmessage->key_len, (char *)rkmessage->key);
    }
    if (output == OUTPUT_HEXDUMP)
        hexdump(stdout, "Message Payload",
            rkmessage->payload, rkmessage->len);
    else
        printf("%.*s\n",
               (int)rkmessage->len, (char *)rkmessage->payload);
}
static void print_partition_list (FILE *fp,
                                  const rd_kafka_topic_partition_list_t
                                  *partitions) {
        int i;
        for (i = 0 ; i < partitions->cnt ; i++) {
                fprintf(stderr, "%s %s [%"PRId32"] offset %"PRId64,
                        i > 0 ? ",":"",
                        partitions->elems[i].topic,
                        partitions->elems[i].partition,
            partitions->elems[i].offset);
        }
        fprintf(stderr, "\n");
}
static void rebalance_cb (rd_kafka_t *rk,
                          rd_kafka_resp_err_t err,
              rd_kafka_topic_partition_list_t *partitions,
                          void *opaque) {
    fprintf(stderr, "%% Consumer group rebalanced: ");
    switch (err)
    {
    case RD_KAFKA_RESP_ERR__ASSIGN_PARTITIONS:
        fprintf(stderr, "assigned:\n");
        print_partition_list(stderr, partitions);
        rd_kafka_assign(rk, partitions);
        wait_eof += partitions->cnt;
        break;
    case RD_KAFKA_RESP_ERR__REVOKE_PARTITIONS:
        fprintf(stderr, "revoked:\n");
        print_partition_list(stderr, partitions);
        rd_kafka_assign(rk, NULL);
        wait_eof = 0;
        break;
    default:
        fprintf(stderr, "failed: %s\n",
                        rd_kafka_err2str(err));
                rd_kafka_assign(rk, NULL);
        break;
    }
}
static int describe_groups (rd_kafka_t *rk, const char *group) {
        rd_kafka_resp_err_t err;
        const struct rd_kafka_group_list *grplist;
        int i;
        err = rd_kafka_list_groups(rk, group, &grplist, 10000);
```

```
        if (err) {
                fprintf(stderr, "%% Failed to acquire group list: %s\n",
                        rd_kafka_err2str(err));
                return -1;
        }
        for (i = 0 ; i < grplist->group_cnt ; i++) {
                const struct rd_kafka_group_info *gi = &grplist->groups[i];
                int j;
                printf("Group \"%s\" in state %s on broker %d (%s:%d)\n",
                       gi->group, gi->state,
                       gi->broker.id, gi->broker.host, gi->broker.port);
                if (gi->err)
                        printf(" Error: %s\n", rd_kafka_err2str(gi->err));
                printf(" Protocol type \"%s\", protocol \"%s\", "
                       "with %d member(s):\n",
                       gi->protocol_type, gi->protocol, gi->member_cnt);
                for (j = 0 ; j < gi->member_cnt ; j++) {
                        const struct rd_kafka_group_member_info *mi;
                        mi = &gi->members[j];
                        printf("  \"%s\", client id \"%s\" on host %s\n",
                               mi->member_id, mi->client_id, mi->client_host);
                        printf("    metadata: %d bytes\n",
                               mi->member_metadata_size);
                        printf("    assignment: %d bytes\n",
                               mi->member_assignment_size);
                }
                printf("\n");
        }
        if (group && !grplist->group_cnt)
                fprintf(stderr, "%% No matching group (%s)\n", group);
        rd_kafka_group_list_destroy(grplist);
        return 0;
}
static void sig_usr1 (int sig) {
    rd_kafka_dump(stdout, rk);
}
int main (int argc, char **argv) {
        char mode = 'C';
    char *brokers = "localhost:9092";
    char *username = "123";
    char *password = "123";
    int opt;
    rd_kafka_conf_t *conf;
    rd_kafka_topic_conf_t *topic_conf;
    char errstr[512];
    const char *debug = NULL;
    int do_conf_dump = 0;
    char tmp[16];
    rd_kafka_resp_err_t err;
    char *group = NULL;
    rd_kafka_topic_partition_list_t *topics;
    int is_subscription;
    int i;
    quiet = !isatty(STDIN_FILENO);
```

```
    /* Kafka configuration */
    conf = rd_kafka_conf_new();
        /* Set logger */
        rd_kafka_conf_set_log_cb(conf, logger);
    /* Quick termination */
    snprintf(tmp, sizeof(tmp), "%i", SIGIO);
    rd_kafka_conf_set(conf, "internal.termination.signal", tmp, NULL, 0);
    /* Topic configuration */
    topic_conf = rd_kafka_topic_conf_new();
     while ((opt = getopt(argc, argv, "g:b:qd:eX:ADO")) != -1) {    //如果是默认接入点，请使用该语句。
    while ((opt = getopt(argc, argv, "g:b:u:p:qd:eX:ADO")) != -1) {     //如果是SSL接入点，请使用该语句。
        switch (opt) {
        case 'b':
            brokers = optarg;
            break;
        case 'g':
            group = optarg;
            break;
    //如果是默认接入点，请注释或者删除以下六行代码（用户名和密码）。
        case 'u':
            username = optarg;
            break;
        case 'p':
            password = optarg;
            break;
        case 'e':
            exit_eof = 1;
            break;
        case 'd':
            debug = optarg;
            break;
        case 'q':
            quiet = 1;
            break;
        case 'A':
            output = OUTPUT_RAW;
            break;
        case 'X':
        {
            char *name, *val;
            rd_kafka_conf_res_t res;
            if (!strcmp(optarg, "list") ||
                !strcmp(optarg, "help")) {
                rd_kafka_conf_properties_show(stdout);
                exit(0);
            }
            if (!strcmp(optarg, "dump")) {
                do_conf_dump = 1;
                continue;
            }
            name = optarg;
            if (!(val = strchr(name, '='))) {
```

```
                fprintf(stderr, "%% Expected "
                    "-X property=value, not %s\n", name);
                exit(1);
            }
            *val = '\0';
            val++;
            res = RD_KAFKA_CONF_UNKNOWN;
            /* Try "topic." prefixed properties on topic
             * conf first, and then fall through to global if
             * it didnt match a topic configuration property. */
            if (!strncmp(name, "topic.", strlen("topic.")))
                res = rd_kafka_topic_conf_set(topic_conf,
                                  name+
                                  strlen("topic."),
                                  val,
                                  errstr,
                                  sizeof(errstr));
            if (res == RD_KAFKA_CONF_UNKNOWN)
                res = rd_kafka_conf_set(conf, name, val,
                            errstr, sizeof(errstr));
            if (res != RD_KAFKA_CONF_OK) {
                fprintf(stderr, "%% %s\n", errstr);
                exit(1);
            }
        }
        break;
                case 'D':
                case 'O':
                        mode = opt;
                        break;
        default:
            goto usage;
        }
    }
    if (do_conf_dump) {
        const char **arr;
        size_t cnt;
        int pass;
        for (pass = 0 ; pass < 2 ; pass++) {
            if (pass == 0) {
                arr = rd_kafka_conf_dump(conf, &cnt);
                printf("# Global config\n");
            } else {
                printf("# Topic config\n");
                arr = rd_kafka_topic_conf_dump(topic_conf,
                                  &cnt);
            }
            for (i = 0 ; i < (int)cnt ; i += 2)
                printf("%s = %s\n",
                       arr[i], arr[i+1]);
            printf("\n");
            rd_kafka_conf_dump_free(arr, cnt);
        }
        exit(0);
```

```
    }
    if (strchr("OC", mode) && optind == argc) {
    usage:
        fprintf(stderr,
            "Usage: %s [options] <topic[:part]> <topic[:part]>..\n"
            "\n"
            "librdkafka version %s (0x%08x)\n"
            "\n"
            " Options:\n"
            "  -g <group>      Consumer group (%s)\n"
            "  -b <brokers>    Broker address (%s)\n"
            "  -u <username>   sasl plain username (%s)\n"    //如果是默认接入点，请注释或删除该行代码。
            "  -p <password>   sasl plain password (%s)\n"     //如果是默认接入点，请注释或删除该行代码。
            "  -e              Exit consumer when last message\n"
            "                  in partition has been received.\n"
                        "  -D              Describe group.\n"
                        "  -O              Get commmitted offset(s)\n"
            "  -d [facs..]     Enable debugging contexts:\n"
            "                  %s\n"
            "  -q              Be quiet\n"
            "  -A              Raw payload output (consumer)\n"
            "  -X <prop=name> Set arbitrary librdkafka "
            "configuration property\n"
            "                  Properties prefixed with \"topic.\" "
            "will be set on topic object.\n"
            "                  Use '-X list' to see the full list\n"
            "                  of supported properties.\n"
            "\n"
                        "For balanced consumer groups use the 'topic1 topic2..'"
                        " format\n"
                        "and for static assignment use "
                        "'topic1:part1 topic1:part2 topic2:part1..'\n"
            "\n",
            argv[0],
            rd_kafka_version_str(), rd_kafka_version(),
                        group, brokers, username, password,      //如果是默认接入点实例，请删除该行代码中后面的用户名和密码（username, password,）。
            RD_KAFKA_DEBUG_CONTEXTS);
        exit(1);
    }
    signal(SIGINT, stop);
    signal(SIGUSR1, sig_usr1);
    if (debug &&
        rd_kafka_conf_set(conf, "debug", debug, errstr, sizeof(errstr)) !=
        RD_KAFKA_CONF_OK) {
        fprintf(stderr, "%% Debug configuration failed: %s: %s\n",
            errstr, debug);
        exit(1);
    }
        /*
         * Client/Consumer group
         */
        if (strchr("CO", mode)) {
```

```
        if (strchr("CO", mode)) {
                /* Consumer groups require a group id */
                if (!group)
                        group = "rdkafka_consumer_example";
                if (rd_kafka_conf_set(conf, "group.id", group,
                                      errstr, sizeof(errstr)) !=
                    RD_KAFKA_CONF_OK) {
                        fprintf(stderr, "%% %s\n", errstr);
                        exit(1);
                }
                /* Consumer groups always use broker based offset storage */
                if (rd_kafka_topic_conf_set(topic_conf, "offset.store.method",
                                            "broker",
                                            errstr, sizeof(errstr)) !=
                    RD_KAFKA_CONF_OK) {
                        fprintf(stderr, "%% %s\n", errstr);
                        exit(1);
                }
                /* Set default topic config for pattern-matched topics. */
                rd_kafka_conf_set_default_topic_conf(conf, topic_conf);
                /* Callback called on partition assignment changes */
                rd_kafka_conf_set_rebalance_cb(conf, rebalance_cb);
                rd_kafka_conf_set(conf, "enable.partition.eof", "true",
                                  NULL, 0);
        }
    //如果是默认接入点，请注释或删除以下九行（if语句）代码。
        if (rd_kafka_conf_set(conf, "ssl.ca.location", "ca-cert.pem", errstr, sizeof(errstr
)) != RD_KAFKA_CONF_OK
                || rd_kafka_conf_set(conf, "security.protocol", "sasl_ssl", errstr, sizeof(
errstr)) != RD_KAFKA_CONF_OK
                || rd_kafka_conf_set(conf, "sasl.mechanism", "PLAIN", errstr, sizeof(errstr
)) != RD_KAFKA_CONF_OK
                || rd_kafka_conf_set(conf, "sasl.username", username, errstr, sizeof(errstr
)) != RD_KAFKA_CONF_OK
                || rd_kafka_conf_set(conf, "sasl.password", password, errstr, sizeof(errstr
)) != RD_KAFKA_CONF_OK
           ) {
            fprintf(stderr, "%s\n", errstr);
            return -1;
        }
        /* Create Kafka handle */
        if (!(rk = rd_kafka_new(RD_KAFKA_CONSUMER, conf,
                                errstr, sizeof(errstr)))) {
                fprintf(stderr,
                        "%% Failed to create new consumer: %s\n",
                        errstr);
                exit(1);
        }
        /* Add brokers */
        if (rd_kafka_brokers_add(rk, brokers) == 0) {
                fprintf(stderr, "%% No valid brokers specified\n");
                exit(1);
        }
        if (mode == 'D') {
                int r;
```

```
                /* Describe groups */
                r = describe_groups(rk, group);
                rd_kafka_destroy(rk);
                exit(r == -1 ? 1 : 0);
        }
        /* Redirect rd_kafka_poll() to consumer_poll() */
        rd_kafka_poll_set_consumer(rk);
        topics = rd_kafka_topic_partition_list_new(argc - optind);
        is_subscription = 1;
        for (i = optind ; i < argc ; i++) {
                /* Parse "topic[:part] */
                char *topic = argv[i];
                char *t;
                int32_t partition = -1;
                if ((t = strstr(topic, ":"))) {
                        *t = '\0';
                        partition = atoi(t+1);
                        is_subscription = 0; /* is assignment */
                        wait_eof++;
                }
                rd_kafka_topic_partition_list_add(topics, topic, partition);
        }
        if (mode == 'O') {
                /* Offset query */
                err = rd_kafka_committed(rk, topics, 5000);
                if (err) {
                        fprintf(stderr, "%% Failed to fetch offsets: %s\n",
                                rd_kafka_err2str(err));
                        exit(1);
                }
                for (i = 0 ; i < topics->cnt ; i++) {
                        rd_kafka_topic_partition_t *p = &topics->elems[i];
                        printf("Topic \"%s\" partition %"PRId32,
                               p->topic, p->partition);
                        if (p->err)
                                printf(" error %s",
                                       rd_kafka_err2str(p->err));
                        else {
                                printf(" offset %"PRId64"",
                                       p->offset);
                                if (p->metadata_size)
                                        printf(" (%d bytes of metadata)",
                                               (int)p->metadata_size);
                        }
                        printf("\n");
                }
                goto done;
        }
        if (is_subscription) {
                fprintf(stderr, "%% Subscribing to %d topics\n", topics->cnt);
                if ((err = rd_kafka_subscribe(rk, topics))) {
                        fprintf(stderr,
                                "%% Failed to start consuming topics: %s\n",
                                rd_kafka_err2str(err));
```

```
                        exit(1);
                }
        } else {
                fprintf(stderr, "%% Assigning %d partitions\n", topics->cnt);
                if ((err = rd_kafka_assign(rk, topics))) {
                        fprintf(stderr,
                                "%% Failed to assign partitions: %s\n",
                                rd_kafka_err2str(err));
                }
        }
        while (run) {
                rd_kafka_message_t *rkmessage;
                rkmessage = rd_kafka_consumer_poll(rk, 1000);
                if (rkmessage) {
                        msg_consume(rkmessage);
                        rd_kafka_message_destroy(rkmessage);
                }
        }
done:
        err = rd_kafka_consumer_close(rk);
        if (err)
                fprintf(stderr, "%% Failed to close consumer: %s\n",
                        rd_kafka_err2str(err));
        else
                fprintf(stderr, "%% Consumer closed\n");
        rd_kafka_topic_partition_list_destroy(topics);
        /* Destroy handle */
        rd_kafka_destroy(rk);
    /* Let background threads clean up and terminate cleanly. */
    run = 5;
    while (run-- > 0 && rd_kafka_wait_destroyed(1000) == -1)
        printf("Waiting for librdkafka to decommission\n");
    if (run <= 0)
        rd_kafka_dump(stdout, rk);
    return 0;
}
```

6.Go SDK收发消息

本文介绍如何使用Go SDK通过接入点接入消息队列Kafka版并收发消息。

环境准备

您已安装Go。更多信息，请参见安装Go。

> **说明** 该kafka-confluent-go-demo不支持Windows系统。

准备配置

1. 访问aliware-kafka-demos，单击 Code 图标，然后在下拉框选择**Download ZIP**，下载Demo工程并解压。
2. 在解压的Demo工程中，找到*kafka-confluent-go-demo*文件夹，将此文件夹上传在Linux系统的*/home*路径下。
3. 登录Linux系统，进入*/home/kafka-confluent-go-demo*路径，修改配置文件*conf/kafka.json*。

```
{
  "topic": "XXX",
  "group.id": "XXX",
  "bootstrap.servers" : "XXX:XX,XXX:XX,XXX:XX",
  "security.protocol" : "plaintext",
  "sasl.mechanism" : "XXX",
  "sasl.username" : "XXX",
  "sasl.password" : "XXX"
}
```

参数	描述
topic	实例的Topic名称。您可在消息队列Kafka版控制台的**Topic 管理**页面获取。
group.id	实例的Group。您可在消息队列Kafka版控制台的**Group 管理**页面获取。 **说明** 如果应用运行producer.go发送消息，该参数可以不配置；如果应用运行consumer.go订阅消息，该参数必须配置。
bootstrap.servers	SSL接入点的IP地址以及端口。您可在消息队列Kafka版控制台的**实例详情**页面的**接入点信息**区域获取。
security.protocol	SASL用户认证协议，默认为plaintext。各类型接入点对应取值如下： ○ 默认接入点：plaintext。 ○ SSL接入点：sasl_ssl。 ○ SASL接入点：sasl_plaintext。

参数	描述
sasl.mechanism	消息收发的机制。各类型接入点对应取值如下： ○ 默认接入点：不涉及，无需配置。 ○ SSL接入点：PLAIN。 ○ SASL接入点：PLAIN机制需配置为PLAIN；SCRAM机制需配置为SCRAM-SHA-256。
sasl.username	SASL用户名。如果是SSL接入点或SASL接入点，需配置该参数。 **说明** ○ 如果实例未开启ACL，您可以在消息队列Kafka版控制台的**实例详情**页面的**配置信息**区域获取默认的**用户名**和**密码**。 ○ 如果实例已开启ACL，请确保要使用的SASL用户已被授予向消息队列Kafka版实例收发消息的权限。具体操作，请参见SASL用户授权。
sasl.password	SASL用户密码。如果是SSL接入点或SASL接入点，需配置该参数。

发送消息

执行以下命令运行*producer.go*发送消息。

```
go run -mod=vendor producer/producer.go
```

消息程序*producer.go*示例代码如下：

```
package main
import (
    "encoding/json"
    "fmt"
    "github.com/confluentinc/confluent-kafka-go/kafka"
    "os"
    "path/filepath"
)
type KafkaConfig struct {
    Topic      string `json:"topic"`
    GroupId    string `json:"group.id"`
    BootstrapServers    string `json:"bootstrap.servers"`
    SecurityProtocol string `json:"security.protocol"`
    SslCaLocation string `json:"ssl.ca.location"`
    SaslMechanism string `json:"sasl.mechanism"`
    SaslUsername string `json:"sasl.username"`
    SaslPassword string `json:"sasl.password"`
}
// config should be a pointer to structure, if not, panic
func loadJsonConfig() *KafkaConfig {
    workPath, err := os.Getwd()
    if err != nil {
        panic(err)
    }
```

```
    configPath := filepath.Join(workPath, "conf")
    fullPath := filepath.Join(configPath, "kafka.json")
    file, err := os.Open(fullPath);
    if (err != nil) {
        msg := fmt.Sprintf("Can not load config at %s. Error: %v", fullPath, err)
        panic(msg)
    }
    defer file.Close()
    decoder := json.NewDecoder(file)
    var config = &KafkaConfig{}
    err = decoder.Decode(config);
    if (err != nil) {
        msg := fmt.Sprintf("Decode json fail for config file at %s. Error: %v", fullPath, e
rr)
        panic(msg)
    }
    json.Marshal(config)
    return  config
}
func doInitProducer(cfg *KafkaConfig) *kafka.Producer {
    fmt.Print("init kafka producer, it may take a few seconds to init the connection\n")
    //common arguments
    var kafkaconf = &kafka.ConfigMap{
        "api.version.request": "true",
        "message.max.bytes": 1000000,
        "linger.ms": 10,
        "retries": 30,
        "retry.backoff.ms": 1000,
        "acks": "1"}
    kafkaconf.SetKey("bootstrap.servers", cfg.BootstrapServers)
    switch cfg.SecurityProtocol {
        case "plaintext" :
            kafkaconf.SetKey("security.protocol", "plaintext");
        case "sasl_ssl":
            kafkaconf.SetKey("security.protocol", "sasl_ssl");
            kafkaconf.SetKey("ssl.ca.location", "conf/ca-cert.pem");
            kafkaconf.SetKey("sasl.username", cfg.SaslUsername);
            kafkaconf.SetKey("sasl.password", cfg.SaslPassword);
            kafkaconf.SetKey("sasl.mechanism", cfg.SaslMechanism)
    case "sasl_plaintext":
            kafkaconf.SetKey("sasl.mechanism", "PLAIN")
            kafkaconf.SetKey("security.protocol", "sasl_plaintext");
            kafkaconf.SetKey("sasl.username", cfg.SaslUsername);
            kafkaconf.SetKey("sasl.password", cfg.SaslPassword);
            kafkaconf.SetKey("sasl.mechanism", cfg.SaslMechanism)
    default:
            panic(kafka.NewError(kafka.ErrUnknownProtocol, "unknown protocol", true))
    }
    producer, err := kafka.NewProducer(kafkaconf)
    if err != nil {
        panic(err)
    }
    fmt.Print("init kafka producer success\n")
    return producer;
```

```
}
func main() {
    // Choose the correct protocol
    // 9092 for PLAINTEXT
    // 9093 for SASL_SSL, need to provide sasl.username and sasl.password
    // 9094 for SASL_PLAINTEXT, need to provide sasl.username and sasl.password
    cfg := loadJsonConfig();
    producer := doInitProducer(cfg)
    defer producer.Close()
    // Delivery report handler for produced messages
    go func() {
        for e := range producer.Events() {
            switch ev := e.(type) {
            case *kafka.Message:
                if ev.TopicPartition.Error != nil {
                    fmt.Printf("Delivery failed: %v\n", ev.TopicPartition)
                } else {
                    fmt.Printf("Delivered message to %v\n", ev.TopicPartition)
                }
            }
        }
    }()
    // Produce messages to topic (asynchronously)
    topic := cfg.Topic
    for _, word := range []string{"Welcome", "to", "the", "Confluent", "Kafka", "Golang", "
client"} {
        producer.Produce(&kafka.Message{
            TopicPartition: kafka.TopicPartition{Topic: &topic, Partition: kafka.PartitionA
ny},
            Value:          []byte(word),
        }, nil)
    }
    // Wait for message deliveries before shutting down
    producer.Flush(15 * 1000)
}
```

订阅消息

执行以下命令运行*consumer.go*订阅消息。

```
go run -mod=vendor consumer/consumer.go
```

消息程序*consumer.go*示例代码如下：

```
package main
import (
    "encoding/json"
    "fmt"
    "github.com/confluentinc/confluent-kafka-go/kafka"
    "os"
    "path/filepath"
)
```

```
type KafkaConfig struct {
    Topic      string `json:"topic"`
    GroupId    string `json:"group.id"`
    BootstrapServers    string `json:"bootstrap.servers"`
    SecurityProtocol string `json:"security.protocol"`
    SaslMechanism string `json:"sasl.mechanism"`
    SaslUsername string `json:"sasl.username"`
    SaslPassword string `json:"sasl.password"`
}
// config should be a pointer to structure, if not, panic
func loadJsonConfig() *KafkaConfig {
    workPath, err := os.Getwd()
    if err != nil {
        panic(err)
    }
    configPath := filepath.Join(workPath, "conf")
    fullPath := filepath.Join(configPath, "kafka.json")
    file, err := os.Open(fullPath);
    if (err != nil) {
        msg := fmt.Sprintf("Can not load config at %s. Error: %v", fullPath, err)
        panic(msg)
    }
    defer file.Close()
    decoder := json.NewDecoder(file)
    var config = &KafkaConfig{}
    err = decoder.Decode(config);
    if (err != nil) {
        msg := fmt.Sprintf("Decode json fail for config file at %s. Error: %v", fullPath, e
rr)
        panic(msg)
    }
    json.Marshal(config)
    return  config
}
func doInitConsumer(cfg *KafkaConfig) *kafka.Consumer {
    fmt.Print("init kafka consumer, it may take a few seconds to init the connection\n")
    //common arguments
    var kafkaconf = &kafka.ConfigMap{
        "api.version.request": "true",
        "auto.offset.reset": "latest",
        "heartbeat.interval.ms": 3000,
        "session.timeout.ms": 30000,
        "max.poll.interval.ms": 120000,
        "fetch.max.bytes": 1024000,
        "max.partition.fetch.bytes": 256000}
    kafkaconf.SetKey("bootstrap.servers", cfg.BootstrapServers);
    kafkaconf.SetKey("group.id", cfg.GroupId)
    switch cfg.SecurityProtocol {
    case "plaintext" :
        kafkaconf.SetKey("security.protocol", "plaintext");
    case "sasl_ssl":
        kafkaconf.SetKey("security.protocol", "sasl_ssl");
        kafkaconf.SetKey("ssl.ca.location", "./conf/ca-cert.pem");
        kafkaconf.SetKey("sasl.username", cfg.SaslUsername);
```

```
        kafkaconf.SetKey("sasl.password", cfg.SaslPassword);
        kafkaconf.SetKey("sasl.mechanism", cfg.SaslMechanism)
    case "sasl_plaintext":
        kafkaconf.SetKey("security.protocol", "sasl_plaintext");
        kafkaconf.SetKey("sasl.username", cfg.SaslUsername);
        kafkaconf.SetKey("sasl.password", cfg.SaslPassword);
        kafkaconf.SetKey("sasl.mechanism", cfg.SaslMechanism)
    default:
        panic(kafka.NewError(kafka.ErrUnknownProtocol, "unknown protocol", true))
    }
    consumer, err := kafka.NewConsumer(kafkaconf)
    if err != nil {
        panic(err)
    }
    fmt.Print("init kafka consumer success\n")
    return consumer;
}
func main() {
    // Choose the correct protocol
    // 9092 for PLAINTEXT
    // 9093 for SASL_SSL, need to provide sasl.username and sasl.password
    // 9094 for SASL_PLAINTEXT, need to provide sasl.username and sasl.password
    cfg := loadJsonConfig();
    consumer := doInitConsumer(cfg)
    consumer.SubscribeTopics([]string{cfg.Topic}, nil)
    for {
        msg, err := consumer.ReadMessage(-1)
        if err == nil {
            fmt.Printf("Message on %s: %s\n", msg.TopicPartition, string(msg.Value))
        } else {
            // The client will automatically try to recover from all errors.
            fmt.Printf("Consumer error: %v (%v)\n", err, msg)
        }
    }
    consumer.Close()
}
```

7.PHP SDK收发消息

本文介绍如何使用PHP SDK通过接入点接入消息队列Kafka版并收发消息。

环境准备

- 安装GCC
- 安装PHP
- 安装PECL

安装C++依赖库

1. 执行以下命令切换到yum源配置目录*/etc/yum.repos.d/*。

```
cd /etc/yum.repos.d/
```

2. 创建yum源配置文件*confluent.repo*。

```
[Confluent.dist]
name=Confluent repository (dist)
baseurl=https://packages.confluent.io/rpm/5.1/7
gpgcheck=1
gpgkey=https://packages.confluent.io/rpm/5.1/archive.key
enabled=1
[Confluent]
name=Confluent repository
baseurl=https://packages.confluent.io/rpm/5.1
gpgcheck=1
gpgkey=https://packages.confluent.io/rpm/5.1/archive.key
enabled=1
```

3. 执行以下命令安装C++依赖库。

```
yum install librdkafka-devel
```

安装PHP依赖库

1. 执行以下命令安装PHP依赖库。

```
pecl install rdkafka
```

2. 在PHP的初始化文件*php.ini*中添加以下一行语句以开启扩展。

```
extension=rdkafka.so
```

准备配置

1. （可选）下载SSL根证书。如果是SSL接入点，需下载该证书。
2. 访问aliware-kafka-demos，单击 下载 ZIP，下载Demo工程到本地并解压。
3. 在解压的Demo工程找到*kafka-php-demo*文件夹，根据接入点类型打开对应的文件夹，配置*setting.php*文件。

```
<?php
return [
    'sasl_plain_username' => 'xxx',
    'sasl_plain_password' => 'xxx',
    'bootstrap_servers' => "xxx:xx,xxx:xx",
    'topic_name' => 'xxx',
    'consumer_id' => 'xxx'
];
```

参数	描述
sasl_plain_username	SASL用户名。如果是默认接入点，则无此配置项。 **说明** ○ 如果实例未开启ACL，您可以在消息队列Kafka版控制台的**实例详情**页面的**配置信息**区域获取默认的**用户名**和**密码**。 ○ 如果实例已开启ACL，请确保要使用的SASL用户已被授予向消息队列Kafka版实例收发消息的权限。具体操作，请参见SASL用户授权。
sasl_plain_password	SASL用户名密码。如果是默认接入点，则无此配置项。
bootstrap_servers	SSL接入点。您可在消息队列Kafka版控制台的**实例详情**页面的**接入点信息**区域获取。
topic_name	Topic名称。您可在消息队列Kafka版控制台的**Topic 管理**页面获取。
consumer_id	Group名称。您可在消息队列Kafka版控制台的**Group 管理**页面获取。

4. 配置完成后，将配置文件所在文件夹下的全部文件（如果是SSL接入点实例，包含证书SSL根证书文件）上传至服务器PHP安装目录下。

发送消息

执行以下命令发送消息。

```
php kafka-producer.php
```

消息程序*kafka-producer.php*示例代码如下：

说明 示例代码为SSL接入点的代码。如果是默认接入点，无SASL相关代码，即删除如下代码中包含 `sasl.` 和 `ssl.` 相关的代码即可。

```
<?php
$setting = require __DIR__ . '/setting.php';
$conf = new RdKafka\Conf();
$conf->set('sasl.mechanisms', 'PLAIN');
$conf->set('api.version.request', 'true');
$conf->set('sasl.username', $setting['sasl_plain_username']);
$conf->set('sasl.password', $setting['sasl_plain_password']);
$conf->set('security.protocol', 'SASL_SSL');
$conf->set('ssl.ca.location', __DIR__ . '/ca-cert.pem');
$conf->set('message.send.max.retries', 5);
$rk = new RdKafka\Producer($conf);
# if want to debug, set log level to LOG_DEBUG
$rk->setLogLevel(LOG_INFO);
$rk->addBrokers($setting['bootstrap_servers']);
$topic = $rk->newTopic($setting['topic_name']);
$a = $topic->produce(RD_KAFKA_PARTITION_UA, 0, "Message hello kafka");
$rk->poll(0);
while ($rk->getOutQLen() > 0) {
    $rk->poll(50);
}
echo "send succ" . PHP_EOL;
```

代码示例详情，请参见php-rdkafka。

订阅消息

执行如下命令订阅消息。

```
php kafka-consumer.php
```

消息程序*kafka-consumer.php*示例代码如下：

> **说明** 示例代码为SSL接入点的代码。如果是默认接入点，无SASL相关代码，即删除如下代码中包含 `sasl.` 和 `ssl.` 相关的代码即可。

```
<?php
$setting = require __DIR__ . '/setting.php';
$conf = new RdKafka\Conf();
$conf->set('sasl.mechanisms', 'PLAIN');
$conf->set('api.version.request', 'true');
$conf->set('sasl.username', $setting['sasl_plain_username']);
$conf->set('sasl.password', $setting['sasl_plain_password']);
$conf->set('security.protocol', 'SASL_SSL');
$conf->set('ssl.ca.location', __DIR__ . '/ca-cert.pem');
$conf->set('group.id', $setting['consumer_id']);
$conf->set('metadata.broker.list', $setting['bootstrap_servers']);
$topicConf = new RdKafka\TopicConf();
$conf->setDefaultTopicConf($topicConf);
$consumer = new RdKafka\KafkaConsumer($conf);
$consumer->subscribe([$setting['topic_name']]);
echo "Waiting for partition assignment... (make take some time when\n";
echo "quickly re-joining the group after leaving it.)\n";
while (true) {
    $message = $consumer->consume(30 * 1000);
    switch ($message->err) {
        case RD_KAFKA_RESP_ERR_NO_ERROR:
            var_dump($message);
            break;
        case RD_KAFKA_RESP_ERR__PARTITION_EOF:
            echo "No more messages; will wait for more\n";
            break;
        case RD_KAFKA_RESP_ERR__TIMED_OUT:
            echo "Timed out\n";
            break;
        default:
            throw new \Exception($message->errstr(), $message->err);
            break;
    }
}
?>
```

代码示例详情，请参见php-rdkafka。

8.Ruby SDK收发消息

本文介绍如何使用Ruby SDK通过接入点接入消息队列Kafka版并收发消息。

环境准备

您已安装Ruby。更多信息，请参见安装Ruby。

安装Ruby依赖库

1. 执行以下命令安装Ruby依赖库。

```
gem install ruby-kafka -v 0.6.8
```

准备配置

1. （可选）下载SSL根证书。如果是SSL接入点，需下载该证书。
2. 访问aliware-kafka-demos，单击 下载 ZIP，下载Demo工程到本地并解压。
3. 在解压的Demo工程找到*kafka-ruby-demo*文件夹，根据接入点类型打开对应的文件夹，配置producer.ruby文件和consumer.ruby文件。

   配置项说明

参数	描述
brokers	SSL接入点。您可在消息队列Kafka版控制台的**实例详情**页面的**接入点信息**区域获取。
topic	Topic名称。您可在消息队列Kafka版控制台的**Topic 管理**页面获取。
username	SASL用户名。如果是默认接入点，则无此配置项。 **说明** ○ 如果实例未开启ACL，您可以在消息队列Kafka版控制台的**实例详情**页面的**配置信息**区域获取默认的**用户名**和**密码**。 ○ 如果实例已开启ACL，请确保要使用的SASL用户已被授予向消息队列Kafka版实例收发消息的权限。具体操作，请参见SASL用户授权。
password	SASL用户名密码。如果是默认接入点，则无此配置项。
consumerGroup	Group名称。您可在消息队列Kafka版控制台的**Group 管理**页面获取。

4. 配置完成后，将配置文件所在文件夹下的全部文件（如果是SSL接入点实例，包含证书SSL根证书文件），上传至服务器Ruby安装目录下。

发送消息

执行以下命令发送消息。

```
ruby producer.ruby
```

关于代码中配置项说明，请参见配置项说明。

消息程序*producer.ruby*代码示例如下：

> **说明** 示例代码为SSL接入点的代码。您需要根据实际接入点类型，删除或者修改配置项，其余代码请根据加粗代码注释修改。

```
# frozen_string_literal: true
$LOAD_PATH.unshift(File.expand_path("../../lib", __FILE__))
require "kafka"
logger = Logger.new($stdout)
#logger.level = Logger::DEBUG
logger.level = Logger::INFO
brokers = "xxx:xx,xxx:xx"
topic = "xxx"
username = "xxx"
password = "xxx"
kafka = Kafka.new(
    seed_brokers: brokers,
    client_id: "sasl-producer",      #如果是默认接入点，取值需修改为“simple-producer”。
    logger: logger,
    # put "./cert.pem" to anywhere this can read
    #如果是默认接入点，删除以下三行代码。
    ssl_ca_cert: File.read('./cert.pem'),
    sasl_plain_username: username,
    sasl_plain_password: password,
    )
producer = kafka.producer
begin
    $stdin.each_with_index do |line, index|
    producer.produce(line, topic: topic)
    producer.deliver_messages
end
ensure
    producer.deliver_messages
    producer.shutdown
end
```

订阅消息

执行以下命令消费消息。

```
ruby consumer.ruby
```

消息程序*consumer.ruby*示例代码如下：

关于代码中配置项说明，请参见配置项说明。

说明 示例代码为SSL接入点的代码。您需要根据实际接入点类型，删除或者修改配置项，其余代码请根据加粗代码注释修改。

```
# frozen_string_literal: true
$LOAD_PATH.unshift(File.expand_path("../../lib", __FILE__))
require "kafka"
logger = Logger.new(STDOUT)
#logger.level = Logger::DEBUG
logger.level = Logger::INFO
brokers = "xxx:xx,xxx:xx"
topic = "xxx"
username = "xxx"
password = "xxx"
consumerGroup = "xxx"
kafka = Kafka.new(
        seed_brokers: brokers,
        client_id: "sasl-consumer",     #如果是默认接入点，取值需修改为“test”
        socket_timeout: 20,
        logger: logger,
        # put "./cert.pem" to anywhere this can read
        #如果是默认接入点，删除以下三行代码。
        ssl_ca_cert: File.read('./cert.pem'),
        sasl_plain_username: username,
        sasl_plain_password: password,
        )
consumer = kafka.consumer(group_id: consumerGroup)
consumer.subscribe(topic, start_from_beginning: false)
trap("TERM") { consumer.stop }
trap("INT") { consumer.stop }
begin
    consumer.each_message(max_bytes: 64 * 1024) do |message|
    logger.info("Get message: #{message.value}")
    end
rescue Kafka::ProcessingError => e
    warn "Got error: #{e.cause}"
    consumer.pause(e.topic, e.partition, timeout: 20)
    retry
end
```

9.Node.js SDK收发消息

本文介绍如何使用Node.js SDK通过接入点接入消息队列Kafka版并收发消息。

环境配置

- 安装GCC
- 安装Node.js

> 注意 Node.js版本必须大于等于4.0.0。

- 安装OpenSSL

安装C++依赖库

1. 执行以下命令切换到yum源配置目录*/etc/yum.repos.d/*。

```
cd /etc/yum.repos.d/
```

2. 创建yum源配置文件*confluent.repo*。

```
[Confluent.dist]
name=Confluent repository (dist)
baseurl=https://packages.confluent.io/rpm/5.1/7
gpgcheck=1
gpgkey=https://packages.confluent.io/rpm/5.1/archive.key
enabled=1
[Confluent]
name=Confluent repository
baseurl=https://packages.confluent.io/rpm/5.1
gpgcheck=1
gpgkey=https://packages.confluent.io/rpm/5.1/archive.key
enabled=1
```

3. 执行以下命令安装C++依赖库。

```
yum install librdkafka-devel
```

安装Node.js依赖库

1. 执行以下命令为预处理器指定OpenSSL头文件路径。

```
# 配置为您的系统安装的OpenSSL头文件路径。
export CPPFLAGS=-I</usr/local/opt/openssl/include>
```

2. 执行以下命令为连接器指定OpenSSL库路径。

```
# 配置为您的系统安装的OpenSSL库路径。
export LDFLAGS=-L</usr/local/opt/openssl/lib>
```

3. 执行以下命令安装Node.js依赖库。

```
npm install i --unsafe-perm node-rdkafka
```

> **说明** 在命令行窗口执行whereis openssl命令获取OpenSSL头文件路径和库路径。

准备配置

1. （可选）下载SSL根证书。如果是SSL接入点，需下载该证书。
2. 访问aliware-kafka-demos，单击 下载 ZIP，下载Demo工程到本地并解压。
3. 在解压的Demo工程中找到*kafka-nodejs-demo*文件夹，根据接入点类型打开对应的文件夹，配置*setting.js*文件。

```
module.exports = {
    'sasl_plain_username': 'XXX',
    'sasl_plain_password': 'XXX',
    'bootstrap_servers': ["XXX"],
    'topic_name': 'XXX',
    'consumer_id': 'XXX'
}
```

参数	描述
sasl_plain_username	SASL用户名。如果是默认接入点，则无此配置项。 **说明** ○ 如果实例未开启ACL，您可以在消息队列Kafka版控制台的**实例详情**页面的**配置信息**区域获取默认的**用户名**和**密码**。 ○ 如果实例已开启ACL，请确保要使用的SASL用户已被授予向消息队列Kafka版实例收发消息的权限。具体操作，请参见SASL用户授权。
sasl_plain_password	SASL用户名密码。如果是默认接入点，则无此配置项。
bootstrap_servers	SSL接入点。您可在消息队列Kafka版控制台的**实例详情**页面的**接入点信息**区域获取。
topic_name	Topic名称。您可在消息队列Kafka版控制台的**Topic 管理**页面获取。
consumer_id	Group名称。您可在消息队列Kafka版控制台的**Group 管理**页面获取。

4. 配置完成后，将配置文件所在文件夹下的全部文件（如果是SSL接入点实例，包含证书SSL根证书文件），上传至服务器Node.js依赖库安装目录下。

发送消息

执行如下命令发送消息。

```
node producer.js
```

代码示例

- 默认接入点

```
const Kafka = require('node-rdkafka');
const config = require('./setting');
console.log("features:" + Kafka.features);
console.log(Kafka.librdkafkaVersion);
var producer = new Kafka.Producer({
    /*'debug': 'all', */
    'api.version.request': 'true',
    'bootstrap.servers': config['bootstrap_servers'],
    'dr_cb': true,
    'dr_msg_cb': true
});
var connected = false
producer.setPollInterval(100);
producer.connect();
producer.on('ready', function() {
  connected = true
  console.log("connect ok")
});
producer.on("disconnected", function() {
  connected = false;
  producer.connect();
})
producer.on('event.log', function(event) {
      console.log("event.log", event);
});
producer.on("error", function(error) {
    console.log("error:" + error);
});
function produce() {
  try {
    producer.produce(
      config['topic_name'],
      null,
      new Buffer('Hello Ali Kafka'),
      null,
      Date.now()
    );
  } catch (err) {
    console.error('A problem occurred when sending our message');
    console.error(err);
  }
}
producer.on('delivery-report', function(err, report) {
    console.log("delivery-report: producer ok");
});
producer.on('event.error', function(err) {
    console.error('event.error:' + err);
})
setInterval(produce,1000,"Interval");
```

- SSL接入点

```
const Kafka = require('node-rdkafka');
const config = require('./setting');
```

```
console.log("features:" + Kafka.features);
console.log(Kafka.librdkafkaVersion);
var producer = new Kafka.Producer({
    /*'debug': 'all', */
    'api.version.request': 'true',
    'bootstrap.servers': config['bootstrap_servers'],
    'dr_cb': true,
    'dr_msg_cb': true,
    'security.protocol' : 'sasl_ssl',
    'ssl.ca.location' : './ca-cert.pem',
    'sasl.mechanisms' : 'PLAIN',
    'sasl.username' : config['sasl_plain_username'],
    'sasl.password' : config['sasl_plain_password']
});
var connected = false
producer.setPollInterval(100);
producer.connect();
producer.on('ready', function() {
  connected = true
  console.log("connect ok")
  });
function produce() {
  try {
    producer.produce(
      config['topic_name'],
      new Buffer('Hello Ali Kafka'),
      null,
      Date.now()
    );
  } catch (err) {
    console.error('A problem occurred when sending our message');
    console.error(err);
  }
}
producer.on("disconnected", function() {
  connected = false;
  producer.connect();
})
producer.on('event.log', function(event) {
      console.log("event.log", event);
});
producer.on("error", function(error) {
    console.log("error:" + error);
});
producer.on('delivery-report', function(err, report) {
    console.log("delivery-report: producer ok");
});
// Any errors we encounter, including connection errors
producer.on('event.error', function(err) {
    console.error('event.error:' + err);
})
setInterval(produce,1000,"Interval");
```

订阅消息

执行如下命令消费消息。

```
node consumer.js
```

代码示例

- 默认接入点

```
const Kafka = require('node-rdkafka');
const config = require('./setting');
console.log(Kafka.features);
console.log(Kafka.librdkafkaVersion);
console.log(config)
var consumer = new Kafka.KafkaConsumer({
    /*'debug': 'all',*/
    'api.version.request': 'true',
    'bootstrap.servers': config['bootstrap_servers'],
    'group.id' : config['consumer_id']
});
consumer.connect();
consumer.on('ready', function() {
  console.log("connect ok");
  consumer.subscribe([config['topic_name']]);
  consumer.consume();
})
consumer.on('data', function(data) {
  console.log(data);
});
consumer.on('event.log', function(event) {
      console.log("event.log", event);
});
consumer.on('error', function(error) {
    console.log("error:" + error);
});
consumer.on('event', function(event) {
        console.log("event:" + event);
});
```

- SSL接入点

```
const Kafka = require('node-rdkafka');
const config = require('./setting');
console.log(Kafka.features);
console.log(Kafka.librdkafkaVersion);
console.log(config)
var consumer = new Kafka.KafkaConsumer({
    /*'debug': 'all',*/
    'api.version.request': 'true',
    'bootstrap.servers': config['bootstrap_servers'],
    'security.protocol' : 'sasl_ssl',
    'ssl.ca.location' : './ca-cert.pem',
    'sasl.mechanisms' : 'PLAIN',
    'message.max.bytes': 32000,
    'fetch.max.bytes' : 32000,
    'fetch.message.max.bytes': 32000,
    'max.partition.fetch.bytes': 32000,
    'sasl.username' : config['sasl_plain_username'],
    'sasl.password' : config['sasl_plain_password'],
    'group.id' : config['consumer_id']
});
consumer.connect();
consumer.on('ready', function() {
  console.log("connect ok");
  consumer.subscribe([config['topic_name']]);
  consumer.consume();
})
consumer.on('data', function(data) {
  console.log(data);
});
consumer.on('event.log', function(event) {
      console.log("event.log", event);
});
consumer.on('error', function(error) {
    console.log("error:" + error);
});
consumer.on('event', function(event) {
        console.log("event:" + event);
});
```

10.C# SDK收发消息

本文介绍如何使用C# SDK通过接入点接入消息队列Kafka版并收发消息。

环境配置

您已安装.NET。更多信息，请参见安装.NET。

安装C#依赖库

1. 执行以下命令安装C#依赖库。

```
dotnet add package -v 1.5.2 Confluent.Kafka
```

准备配置

1. （可选）下载SSL根证书。如果是SSL接入点，需下载该证书。
2. 配置*producer.cs*和*consumer.cs*文件。

   配置项说明

参数	描述
BootstrapServers	SSL接入点。您可在消息队列Kafka版控制台的**实例详情**页面的**接入点信息**区域获取。
SslCaLocation	下载的SSL根证书的路径。仅SSL接入点需要此配置项。
SaslMechanism	收发消息使用的安全机制。 ○ SSL接入点：取值为SaslMechanism.Plain。 ○ SASL接入点：PLAIN机制时，取值为SaslMechanism.Plain；SCRAM机制时，取值为SaslMechanism.ScramSha256。
SecurityProtocol	收发消息使用的安全协议。 ○ SSL接入点：取值为SecurityProtocol.SaslSsl。 ○ SASL接入点：PLAIN机制时，取值为SecurityProtocol.SaslPlaintext；SCRAM机制时，取值为SecurityProtocol.SaslPlaintext。
SaslUsername	SASL用户名。如果是默认接入点，则无此配置项。 **说明** ○ 如果实例未开启ACL，您可以在消息队列Kafka版控制台的**实例详情**页面的**配置信息**区域获取默认的**用户名**和**密码**。 ○ 如果实例已开启ACL，请确保要使用的SASL用户已被授予向消息队列Kafka版实例收发消息的权限。具体操作，请参见SASL用户授权。
SaslPassword	SASL用户名密码。如果是默认接入点，则无此配置项。
topic	Topic名称。您可在消息队列Kafka版控制台的**Topic 管理**页面获取。

参数	描述
GroupId	Group名称。您可在消息队列Kafka版控制台的Group 管理页面获取。

发送消息

执行以下命令发送消息。

```
dotnet run producer.cs
```

消息程序*producer.cs*示例代码如下：

关于代码中配置项说明，请参见配置项说明。

> **注意** 示例代码为SSL接入点的代码。您需要根据实际接入点类型，删除或者修改配置项代码。

```
using System;
using Confluent.Kafka;
class Producer
{
    public static void Main(string[] args)
    {
        var conf = new ProducerConfig {
            BootstrapServers = "XXX,XXX,XXX",
            SslCaLocation = "XXX/ca-cert.pem",
            SaslMechanism = SaslMechanism.Plain,
            SecurityProtocol = SecurityProtocol.SaslSsl,
            SaslUsername = "XXX",
            SaslPassword = "XXX",
            };
        Action<DeliveryReport<Null, string>> handler = r =>
            Console.WriteLine(!r.Error.IsError
                ? $"Delivered message to {r.TopicPartitionOffset}"
                : $"Delivery Error: {r.Error.Reason}");
        string topic ="XXX";
        using (var p = new ProducerBuilder<Null, string>(conf).Build())
        {
            for (int i=0; i<100; ++i)
            {
                p.Produce(topic, new Message<Null, string> { Value = i.ToString() }, handle
r);
            }
            p.Flush(TimeSpan.FromSeconds(10));
        }
    }
}
```

订阅消息

执行以下命令消费消息。

```
dotnet run consumer.cs
```

消息程序*consumer.cs*示例代码如下：

关于代码中配置项说明，请参见配置项说明。

> **注意** 示例代码为SSL接入点的代码。您需要根据实际接入点类型，删除或者修改配置项代码。

```
using System;
using System.Threading;
using Confluent.Kafka;
class Consumer
{
    public static void Main(string[] args)
    {
        var conf = new ConsumerConfig {
            GroupId = "XXX",
            BootstrapServers = "XXX,XXX,XXX",
            SslCaLocation = "XXX/ca-cert.pem",
            SaslMechanism = SaslMechanism.Plain,
            SecurityProtocol = SecurityProtocol.SaslSsl,
            SaslUsername = "XXX",
            SaslPassword = "XXX",
            AutoOffsetReset = AutoOffsetReset.Earliest
        };
        string topic = "XXX";
        using (var c = new ConsumerBuilder<Ignore, string>(conf).Build())
        {
            c.Subscribe(topic);
            CancellationTokenSource cts = new CancellationTokenSource();
            Console.CancelKeyPress += (_, e) => {
                e.Cancel = true;
                cts.Cancel();
            };
            try
            {
                while (true)
                {
                    try
                    {
                        var cr = c.Consume(cts.Token);
                        Console.WriteLine($"Consumed message '{cr.Value}' at: '{cr.TopicPar
titionOffset}'.");
                    }
                    catch (ConsumeException e)
                    {
                        Console.WriteLine($"Error occured: {e.Error.Reason}");
                    }
                }
            }
            catch (OperationCanceledException)
            {
                c.Close();
            }
        }
    }
}
```

11.客户端发送问题

11.1. 哪里可以找到发送最佳实践？

发布者最佳实践。

请参见发布者最佳实践。

11.2. 如何进行发送消息的测试？

您可以直接在消息队列Kafka版控制台进行发送消息的测试。

您可以在控制台的**Topic管理**页面，单击目标Topic右侧的**发送消息**，进行消息发送测试，以验证集群是否运转正常。

11.3. 使用客户端发送消息后，如何确定是否发送成功？

如果回调成功则说明消息发送成功。

大部分客户端在发送之后，会返回Callback或者Future，如果回调成功，则说明消息发送成功。

此外，您还可以在控制台通过以下方式确认消息发送是否正常：

- 查看Topic的分区状态，可以实时看到各个分区的消息数量。
- 查看Topic的流量监控，可以看到分钟级别的流量曲线。

11.4. 生产者会建立多少个到Broker的连接？

每个生产者通常会建立2个到Broker的TCP连接。

每个生产者通常会建立2个到Broker的TCP连接，一个TCP连接用于更新元数据，一个TCP连接用于发送消息。

更多信息，请参见How are TCP Connections managed by kafka-clients scala library。

11.5. Java客户端设置回调是否会影响消息发送的速度？

取决于回调里的处理耗时以及max.in.flight.requests.per.connection的设置。

Java客户端设置回调是否会影响消息发送的速度取决于：

- 回调里的处理耗时：为减少回调里的处理耗时，不要过于频繁地在回调里面做耗时的处理。您可以积累一定量Ack后再做批量的回调处理，或者在另一个异步的线程去处理，从而不阻塞回调的完成。
- max.in.flight.requests.per.connection的设置：在阻塞结束之前，最多能发的消息数由max.in.flight.requests.per.connection决定。

11.6. 为什么PHP发送延时比较长？

PHP发送延时比较长是PHP的语言特性导致的。

PHP每次发送时，都会重新初始化一个KafkaProducer对象，这个初始化会进行各种操作，包括连接各个Broker、更新Metadata等，在VPC内耗时100ms以上，在公网可能耗时500ms以上。相比之下，Java会复用KafkaProducer，所以发送延迟会很低。

12.客户端消费问题

12.1. 客户端首次接入如何排查问题？

您可以从网络连通、客户端版本、配置这三个方向进行排查。

您的客户端首次接入消息队列Kafka版，通常会被以下三个问题困扰：

- 网络连通
- 客户端版本
- 配置

您可以从上述三个方向，一步一步排查，基本上可以解决99%的问题。

12.2. 消息堆积了怎么办？

消息堆积一般是消费速度过慢或者消费线程阻塞造成的，建议查看堆栈信息进行排查。

消息队列Kafka版的消息是客户端主动去服务端拉取的，一般来说，因为是批量拉取机制，服务端拉取都不会是消费的瓶颈。

消息堆积一般是消费速度过慢或者消费线程阻塞造成的。

建议打印消费消息的耗时，或者查看堆栈信息以查看线程执行情况。

> **注意** Java进程可以用Jstack打印消费者进程的堆栈信息。

12.3. 为什么消费客户端频繁出现Rebalance？

可能是消息队列Kafka版开源版本过低或者Consumer没有独立线程维持心跳。

Condition

消费客户端频繁出现Rebalance。

Cause

可能导致故障的原因包括：

- 开源版本为0.10的消息队列Kafka版有一定的概率会触发频繁Rebalance。
- 消息队列Kafka版的Consumer没有独立线程维持心跳，而是把心跳维持与poll接口耦合在一起。其结果就是，如果用户消费出现卡顿，就会导致Consumer心跳超时，引发rebalance。
  - `session.timeout.ms`：心跳超时时间（可以由客户端自行设置）。
  - `max.poll.records`：每次poll返回的最大消息数量。

Remedy

操作步骤

1. 检查消息队列Kafka版实例的开源版本。

如果您的实例的开源版本是0.10，建议您将实例版本升级到稳定的0.10.2。详情请参见升级实例版本。

2. 调整参数。
   - `session.timeout.ms`：建议设置成25s，不超过30s。
   - `max.poll.records`：要小于（最好是远小于）“单个线程每秒消费的条数”x“消费线程的个数”x“`session.timeout`的秒数”。
3. 提高客户端消费速度。

12.4. 消费端从服务端拉取不到消息或拉取消息缓慢

可能的原因包括消息流量达到网络带宽、单个消息大小超过网络带宽或者Consumer每次拉取的消息量超过网络带宽。

Condition

Topic中有消息并且Consumer未消费到最新的位置，出现消费端从服务端拉取不到消息或拉取消息缓慢的情况（特别是公网消费时）。

Cause

可能的原因包括：

- 消费流量达到网络带宽。
- 单个消息大小超过网络带宽。
- Consumer每次拉取的消息量超过网络带宽。

> **说明** Consumer每次消息的拉取量受以下参数影响：
> - max.poll.records：每次拉取的最多消息数。
> - fetch.max.bytes：每次拉取的最大总byte数。
> - max.partition.fetch.bytes：每个Partition每次拉取的最大总byte数。

Remedy

操作步骤

1. 登录消息队列Kafka版控制台查询消息。

   如果能查询到消息，请继续尝试以下步骤。
2. 在**实例详情**页面，单击左侧导航栏的**监控报警**，查看消费流量是否已达到网络带宽。

   如果消费流量已经达到网络带宽，您需要扩充网络带宽。
3. 检查Topic中是否存在单个消息的大小超过网络带宽。

   如果存在单个消息的大小超过网络带宽，请提高网络带宽，或者减小单个消息的大小。
4. 检查Consumer每次拉取的消息量是否超过网络带宽。

   如果每次拉取的消息量超过网络带宽，您需要调整以下参数。
   - 网络带宽>fetch.max.bytes
   - 网络带宽>max.partition.fetch.bytes*总订阅Partition数

12.5. 为什么不推荐使用Sarama Go客户端收发消息？

问题现象

所有Sarama Go版本客户端存在以下已知问题：

- 当Topic新增分区时，Sarama Go客户端无法感知并消费新增分区，需要客户端重启后，才能消费到新增分区。
- 当Sarama Go客户端同时订阅两个以上的Topic时，有可能会导致部分分区无法正常消费消息。
- 当Sarama Go客户端的消费位点重置策略设置为 `Oldest(earliest)` 时，如果客户端宕机或服务端版本升级，由于Sarama Go客户端自行实现OutOfRange机制，有可能会导致客户端从最小位点开始重新消费所有消息。

操作步骤

1. 建议尽早将Sarama Go客户端替换为Confluent Go客户端。

   Confluent Go客户端的Demo地址，请访问kafka-confluent-go-demo。

   > **注意** 如果无法在短期内替换客户端，请注意以下事项：
   >
   > - 针对生产环境，请将位点重置策略设置为 `Newest(latest)` ；针对测试环境，或者其他明确可以接收大量重复消息的场景，设置为 `Oldest(earliest)` 。
   > - 如果发生了位点重置，产生大量堆积，您可以使用消息队列Kafka版控制台提供的重置消费位点功能，手动重置消费位点到某一时间点，无需改代码或换Consumer Group。具体操作，请参见重置消费位点。