

分布式消息服务 Kafka 版

开发指南

文档版本 01
发布日期 2024-11-28



版权所有 © 华为云计算技术有限公司 2024。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HUAWEI和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目录

1 概述	1
2 收集连接信息	2
3 Java	5
3.1 Java 客户端接入示例.....	5
3.2 Java 开发环境搭建.....	12
4 Python	17
5 Go	22
6 获取 Kafka 开源客户端	29
7 spring-kafka 的使用	30

1 概述

Kafka实例完全兼容开源Kafka协议，可以直接使用[kafka开源客户端](#)连接。如果“kafka安全协议”设置为“SASL_SSL”，则在开源客户端基础上使用云服务提供的证书文件。

本指南主要介绍实例连接信息的收集，如获取实例连接地址、Topic名称等，然后提供Java、Python和Go等语言的连接示例。

本指南的示例仅展示Kafka的API调用，生产与消费的API集，请参考[Kafka官网](#)。

客户端网络环境说明

客户端可以通过以下方式访问Kafka实例：

- 如果客户端是云上ECS，与Kafka实例处于同region同VPC，则可以直接访问Kafka实例提供的内网连接地址。
- 如果客户端是云上ECS，与Kafka实例处于相同region但不同VPC，通过以下任意一种方式访问。
 - 创建VPC对等连接，将两个VPC的网络打通，实现跨VPC访问。具体步骤请参考[对等连接](#)。注意修改Kafka实例的安全组，允许端口9092（明文接入）/9093（密文接入）被外部请求访问。
 - 利用VPC终端节点在不同VPC间建立跨VPC的连接通道，实现Kafka客户端通过内网访问Kafka实例，具体步骤请参考[使用VPCEP实现跨VPC访问Kafka](#)。注意修改Kafka实例的安全组，允许端口9011被外部请求访问。
- 如果客户端在其他网络环境，或者与Kafka实例处于不同region，则访问实例的公网地址。

公网访问时，注意修改Kafka实例的安全组，允许端口9094（明文接入）/9095（密文接入）被外部网络访问。

📖 说明

不同网络环境，对于客户端配置来说，只是连接地址的差异，其他都一样。因此，本手册以同一VPC内子网地址的方式，介绍客户端开发环境搭建。

遇到连接超时或失败时，请注意确认网络是否连通。可使用telnet方式，检测实例连接地址与端口。

2 收集连接信息

在连接Kafka实例生产消费消息前，请先获取以下Kafka实例信息。

实例连接地址与端口

实例创建后，从Kafka实例控制台的基本信息页面中获取。

如果Kafka实例为集群部署，至少有3个连接地址，在客户端配置时，建议配置所有的连接地址，提高可靠性。

如果开启公网访问，还可以使用基本信息页面下方的公网连接地址访问Kafka实例。

图 2-1 查看 Kafka 实例 Broker 连接地址与端口



Topic 名称

从Kafka实例控制台的Topic管理页面中获取Topic名称。

📖 说明

未开启“Kafka自动创建Topic”功能时，必须先创建Topic，然后客户端才可以连接Kafka实例生产消费消息。

密文接入信息

如果实例开启密文接入，则需要获得连接实例的用户名与密码、SASL认证机制和Kafka安全协议。Kafka安全协议设置为“SASL_SSL”时，还需要获取SSL证书。

- 连接实例的用户名在Kafka实例控制台的“用户管理”页面中查看，如果忘记密码，可通过[重置密码](#)重新获得。

图 2-2 查看 SASL 用户名

用户名	描述	更新时间	操作
test	--	2024/07/11 09:26:41 GMT+08:00	重置密码 编辑
admin	--	2024/07/11 09:40:12 GMT+08:00	重置密码 编辑 删除

- SASL认证机制在Kafka实例控制台的基本信息页面中获取。
如果SCRAM-SHA-512和PLAIN都开启了，根据实际情况选择其中任何一种配置连接。很久前创建的Kafka实例在详情页如果未显示“开启的SASL认证机制”，默认使用PLAIN机制。

图 2-3 开启的 SASL 认证机制

连接信息

用户名 test [重置密码](#)

内网访问

明文接入

内网明文连接地址 192.168.81.92:9092,192.168.83.7... [复制](#)

密文接入

内网密文连接地址 192.168.81.92:9093,192.168.83.7... [复制](#)

启用的安全协议 SASL_SSL

公网访问 [?](#) 已关闭

开启的SASL认证机制 SCRAM-SHA-512

- Kafka安全协议在Kafka实例控制台的基本信息页面中获取。很久前创建的Kafka实例在详情页如果未显示“启用的安全协议”，默认使用SASL_SSL协议。

图 2-4 启用的安全协议

连接信息

用户名 test [重置密码](#)

内网访问

明文接入

内网明文连接地址 192.168.0.20:9092,192.168.0.124... [复制](#)

密文接入

内网密文连接地址 192.168.0.20:9093,192.168.0.124... [复制](#)

启用的安全协议 SASL_SSL

公网访问 [?](#) 已关闭

- 如果Kafka安全协议设置为“SASL_SSL”，在Kafka实例控制台的基本信息页面中下载证书。

使用Java语言连接实例时，需要用JKS格式的证书。使用Python语言连接实例时，需要用CRT格式的证书。

3 Java

3.1 Java 客户端接入示例

本文介绍Maven方式引入Kafka客户端，并完成Kafka实例连接以及消息生产与消费的相关示例。如果您需要在IDE中查看Demo具体表现，请查看[Java开发环境搭建](#)。

下文所有Kafka的配置信息，如实例连接地址、Topic名称、用户信息等，请参考[收集连接信息](#)获取。

Maven 中引入 Kafka 客户端

```
//Kafka实例基于社区版本1.1.0/2.3.0/2.7/3.x, 推荐客户端保持一致。
<dependency>
  <groupId>org.apache.kafka</groupId>
  <artifactId>kafka-clients</artifactId>
  <version>1.1.0/2.3.0/2.7.2/3.4.0</version>
</dependency>
```

准备 Kafka 配置信息

为了方便，下文分生产与消费两个配置文件介绍。其中涉及SASL认证配置，如果Kafka实例没有开启密文接入，使用的是不加密连接，请注释相关代码；如果Kafka实例开启了密文接入，则必须使用加密方式连接，请设置相关参数。

- 生产消息配置文件（对应[生产消息代码](#)中的dms.sdk.producer.properties文件）
以下粗体部分为不同Kafka实例特有的信息，必须修改。客户端其他参数，可以自主添加。

```
#Topic名称在具体的生产与消费代码中。
#####
#Kafka实例的broker信息，ip:port为实例的连接地址和端口，参考“收集连接信息”章节获取。举例：
bootstrap.servers=100.xxx.xxx.87:909x,100.xxx.xxx.69:909x,100.xxx.xxx.155:909x
bootstrap.servers=ip1:port1,ip2:port2,ip3:port3
#发送确认参数
acks=all
#键的序列化方式
key.serializer=org.apache.kafka.common.serialization.StringSerializer
#值的序列化方式
value.serializer=org.apache.kafka.common.serialization.StringSerializer
#producer可以用来缓存数据的内存大小
buffer.memory=33554432
#重试次数
retries=0
```



```
#####  
#如果不使用密文接入，以下参数请注释掉。  
#####  
#设置SASL认证机制、账号和密码。  
#sasl.mechanism为SASL认证机制，username和password为SASL的用户名和密码，参考“收集连接信息”  
#SASL认证机制为“PLAIN”时，配置信息如下。  
sasl.mechanism=PLAIN  
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \  
  username="username" \  
  password="password";  
#SASL认证机制为“SCRAM-SHA-512”时，配置信息如下。  
sasl.mechanism=SCRAM-SHA-512  
sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule required \  
  username="username" \  
  password="password";  
  
#设置Kafka安全协议。security.protocol为安全协议。  
#安全协议为“SASL_SSL”时，配置信息如下。  
security.protocol=SASL_SSL  
#ssl.truststore.location为SSL证书的存放路径，如下代码以Windows系统路径格式举例，您在使用时请根  
#ssl.truststore.location=E:\\temp\\client.jks  
#ssl.truststore.password为服务器证书密码，配置此密码是为了访问Java生成的jks文件。  
ssl.truststore.password=dms@kafka  
#ssl.endpoint.identification.algorithm为证书域名校验开关，为空则表示关闭，这里需要保持关闭状态，  
#ssl.endpoint.identification.algorithm=必须设置为空。  
#安全协议为“SASL_PLAINTEXT”时，配置信息如下。  
security.protocol=SASL_PLAINTEXT
```

- 消费消息配置文件（对应[消费消息代码](#)中的dms.sdk.consumer.properties文件）
以下粗体部分为不同Kafka实例特有的信息，必须修改。客户端其他参数，可以自主添加。

```
#Topic名称在具体的生产与消费代码中。  
#####  
#Kafka实例的broker信息，ip:port为实例的连接地址和端口，参考“收集连接信息”章节获取。举例：  
bootstrap.servers=100.xxx.xxx.87:909x,100.xxx.xxx.69:909x,100.xxx.xxx.155:909x  
bootstrap.servers=ip1:port1,ip2:port2,ip3:port3  
#用来唯一标识consumer进程所在组的字符串，如果设置同样的group id，表示这些processes都是属于同  
group.id=1  
#键的序列化方式  
key.deserializer=org.apache.kafka.common.serialization.StringDeserializer  
#值的序列化方式  
value.deserializer=org.apache.kafka.common.serialization.StringDeserializer  
#偏移量的方式  
auto.offset.reset=earliest  
#####  
#如果不使用密文接入，以下参数请注释掉。  
#####  
#设置SASL认证机制、账号和密码。  
#sasl.mechanism为SASL认证机制，username和password为SASL的用户名和密码，参考“收集连接信息”  
#SASL认证机制为“PLAIN”时，配置信息如下。  
sasl.mechanism=PLAIN  
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \  
  username="username" \  
  password="password";  
#SASL认证机制为“SCRAM-SHA-512”时，配置信息如下。  
sasl.mechanism=SCRAM-SHA-512  
sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule required \  
  username="username" \  
  password="password";  
  
#设置Kafka安全协议。security.protocol为安全协议。  
#安全协议为“SASL_SSL”时，配置信息如下。  
security.protocol=SASL_SSL  
#ssl.truststore.location为SSL证书的存放路径，如下代码以Windows系统路径格式举例，您在使用时请根
```

据实际运行环境修改路径格式。

```
ssl.truststore.location=E:\\temp\\client.jks
#ssl.truststore.password为服务器证书密码，配置此密码是为了访问Java生成的jks文件。
ssl.truststore.password=dms@kafka
#ssl.endpoint.identification.algorithm为证书域名校验开关，为空则表示关闭，这里需要保持关闭状态，必须设置为空。
ssl.endpoint.identification.algorithm=
#安全协议为“SASL_PLAINTEXT”时，配置信息如下。
security.protocol=SASL_PLAINTEXT
```

生产消息

- 测试代码

```
package com.dms.producer;

import org.apache.kafka.clients.producer.Callback;
import org.apache.kafka.clients.producer.RecordMetadata;
import org.junit.Test;

public class DmsProducerTest {
    @Test
    public void testProducer() throws Exception {
        DmsProducer<String, String> producer = new DmsProducer<String, String>();
        int partition = 0;
        try {
            for (int i = 0; i < 10; i++) {
                String key = null;
                String data = "The msg is " + i;
                // 注意填写您创建的topic名称。另外，生产消息的API有多个，具体参见Kafka官网或者下文的生产消息代码。
                producer.produce("topic-0", partition, key, data, new Callback() {
                    public void onCompletion(RecordMetadata metadata,
                        Exception exception) {
                        if (exception != null) {
                            exception.printStackTrace();
                        }
                        return;
                    }
                });
                System.out.println("produce msg completed");
            }
            System.out.println("produce msg:" + data);
        } catch (Exception e) {
            // TODO: 异常处理
            e.printStackTrace();
        } finally {
            producer.close();
        }
    }
}
```

- 生产消息代码

```
package com.dms.producer;

import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.List;
import java.util.Properties;

import org.apache.kafka.clients.producer.Callback;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerRecord;
```

```

public class DmsProducer<K, V> {
    //引入生产消息的配置信息，具体内容参考上文
    public static final String CONFIG_PRODUCER_FILE_NAME = "dms.sdk.producer.properties";

    private Producer<K, V> producer;

    DmsProducer(String path)
    {
        Properties props = new Properties();
        try {
            InputStream in = new BufferedInputStream(new FileInputStream(path));
            props.load(in);
        }catch (IOException e)
        {
            e.printStackTrace();
            return;
        }
        producer = new KafkaProducer<K,V>(props);
    }
    DmsProducer()
    {
        Properties props = new Properties();
        try {
            props = loadFromClasspath(CONFIG_PRODUCER_FILE_NAME);
        }catch (IOException e)
        {
            e.printStackTrace();
            return;
        }
        producer = new KafkaProducer<K,V>(props);
    }

    /**
     * 生产消息
     *
     * @param topic    topic对象
     * @param partition  partition
     * @param key      消息key
     * @param data     消息数据
     */
    public void produce(String topic, Integer partition, K key, V data)
    {
        produce(topic, partition, key, data, null, (Callback)null);
    }

    /**
     * 生产消息
     *
     * @param topic    topic对象
     * @param partition  partition
     * @param key      消息key
     * @param data     消息数据
     * @param timestamp timestamp
     */
    public void produce(String topic, Integer partition, K key, V data, Long timestamp)
    {
        produce(topic, partition, key, data, timestamp, (Callback)null);
    }

    /**
     * 生产消息
     *
     * @param topic    topic对象
     * @param partition  partition
     * @param key      消息key
     * @param data     消息数据
     * @param callback  callback
     */
    public void produce(String topic, Integer partition, K key, V data, Callback callback)
    {

```

```

        produce(topic, partition, key, data, null, callback);
    }

    public void produce(String topic, V data)
    {
        produce(topic, null, null, data, null, (Callback)null);
    }

    /**
     * 生产消息
     *
     * @param topic    topic对象
     * @param partition partition
     * @param key      消息key
     * @param data     消息数据
     * @param timestamp timestamp
     * @param callback callback
     */
    public void produce(String topic, Integer partition, K key, V data, Long timestamp, Callback
callback)
    {
        ProducerRecord<K, V> kafkaRecord =
            timestamp == null ? new ProducerRecord<K, V>(topic, partition, key, data)
                : new ProducerRecord<K, V>(topic, partition, timestamp, key, data);
        produce(kafkaRecord, callback);
    }

    public void produce(ProducerRecord<K, V> kafkaRecord)
    {
        produce(kafkaRecord, (Callback)null);
    }

    public void produce(ProducerRecord<K, V> kafkaRecord, Callback callback)
    {
        producer.send(kafkaRecord, callback);
    }

    public void close()
    {
        producer.close();
    }

    /**
     * get classloader from thread context if no classloader found in thread
     * context return the classloader which has loaded this class
     *
     * @return classloader
     */
    public static ClassLoader getCurrentClassLoader()
    {
        ClassLoader classLoader = Thread.currentThread()
            .getContextClassLoader();
        if (classLoader == null)
        {
            classLoader = DmsProducer.class.getClassLoader();
        }
        return classLoader;
    }

    /**
     * 从classpath 加载配置信息
     *
     * @param configFileName 配置文件名称
     * @return 配置信息
     * @throws IOException
     */
    public static Properties loadFromClasspath(String configFileName) throws IOException
    {
        ClassLoader classLoader = getCurrentClassLoader();
    }

```

```
Properties config = new Properties();

List<URL> properties = new ArrayList<URL>();
Enumeration<URL> propertyResources = classLoader
    .getResources(configFileName);
while (propertyResources.hasMoreElements())
{
    properties.add(propertyResources.nextElement());
}

for (URL url : properties)
{
    InputStream is = null;
    try
    {
        is = url.openStream();
        config.load(is);
    }
    finally
    {
        if (is != null)
        {
            is.close();
            is = null;
        }
    }
}

return config;
}
```

消费消息

- 测试代码

```
package com.dms.consumer;

import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.junit.Test;
import java.util.Arrays;

public class DmsConsumerTest {
    @Test
    public void testConsumer() throws Exception {
        DmsConsumer consumer = new DmsConsumer();
        consumer.consume(Arrays.asList("topic-0"));
        try {
            for (int i = 0; i < 10; i++){
                ConsumerRecords<Object, Object> records = consumer.poll(1000);
                System.out.println("the numbers of topic:" + records.count());
                for (ConsumerRecord<Object, Object> record : records)
                {
                    System.out.println(record.toString());
                }
            }
        }catch (Exception e)
        {
            // TODO: 异常处理
            e.printStackTrace();
        }finally {
            consumer.close();
        }
    }
}
```

- 消费消息代码

```
package com.dms.consumer;
```

```

import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.util.*;

public class DmsConsumer {

    public static final String CONFIG_CONSUMER_FILE_NAME = "dms.sdk.consumer.properties";

    private KafkaConsumer<Object, Object> consumer;

    DmsConsumer(String path)
    {
        Properties props = new Properties();
        try {
            InputStream in = new BufferedInputStream(new FileInputStream(path));
            props.load(in);
        }catch (IOException e)
        {
            e.printStackTrace();
            return;
        }
        consumer = new KafkaConsumer<Object, Object>(props);
    }

    DmsConsumer()
    {
        Properties props = new Properties();
        try {
            props = loadFromClasspath(CONFIG_CONSUMER_FILE_NAME);
        }catch (IOException e)
        {
            e.printStackTrace();
            return;
        }
        consumer = new KafkaConsumer<Object, Object>(props);
    }

    public void consume(List topics)
    {
        consumer.subscribe(topics);
    }

    public ConsumerRecords<Object, Object> poll(long timeout)
    {
        return consumer.poll(timeout);
    }

    public void close()
    {
        consumer.close();
    }

    /**
     * get classloader from thread context if no classloader found in thread
     * context return the classloader which has loaded this class
     *
     * @return classloader
     */
    public static ClassLoader getCurrentClassLoader()
    {
        ClassLoader classLoader = Thread.currentThread()
            .getContextClassLoader();
        if (classLoader == null)
        {
            classLoader = DmsConsumer.class.getClassLoader();
        }
    }
}

```

```
    }
    return classLoader;
}

/**
 * 从classpath 加载配置信息
 *
 * @param configFileName 配置文件名称
 * @return 配置信息
 * @throws IOException
 */
public static Properties loadFromClasspath(String configFileName) throws IOException
{
    ClassLoader classLoader = getCurrentClassLoader();
    Properties config = new Properties();

    List<URL> properties = new ArrayList<URL>();
    Enumeration<URL> propertyResources = classLoader
        .getResources(configFileName);
    while (propertyResources.hasMoreElements())
    {
        properties.add(propertyResources.nextElement());
    }

    for (URL url : properties)
    {
        InputStream is = null;
        try
        {
            is = url.openStream();
            config.load(is);
        }
        finally
        {
            if (is != null)
            {
                is.close();
                is = null;
            }
        }
    }

    return config;
}
}
```

3.2 Java 开发环境搭建

基于[收集连接信息](#)的介绍，假设您已经获取了实例连接相关的信息，以及配置好客户端的网络环境。本章节以生产与发送消息的Demo为例，介绍Kafka客户端的环境配置。

开发环境

- Maven
Apache Maven 3.0.3及以上版本，可至[Maven官方下载页面](#)下载。
- JDK
Java Development Kit 1.8.111及以上版本，可至[Oracle官方下载页面](#)下载。
安装后注意配置JAVA的环境变量。
- IntelliJ IDEA
获取并安装IntelliJ IDEA，可至[IntelliJ IDEA官方网站](#)下载。

操作步骤

步骤1 下载Demo包。

下载后解压，有如下文件：

表 3-1 Kafka Demo 文件清单

文件名	路径	说明
DmsConsumer.java	.\src\main\java\com\dms\consumer	消费消息的API。
DmsProducer.java	.\src\main\java\com\dms\producer	生产消息的API。
dms.sdk.consumer.properties	.\src\main\resources	消费消息的配置信息。
dms.sdk.producer.properties	.\src\main\resources	生产消息的配置信息。
client.jks	.\src\main\resources	SSL证书，用于SASL_SSL方式连接。
DmsConsumerTest.java	.\src\test\java\com\dms\consumer	消费消息的测试代码。
DmsProducerTest.java	.\src\test\java\com\dms\producer	生产消息的测试代码。
pom.xml	.\	maven配置文件，包含Kafka客户端引用。

步骤2 打开IntelliJ IDEA，导入Demo。

Demo是一个Maven构建的Java工程，因此需要配置JDK环境，以及IDEA的Maven插件。

图 3-1 选择“导入工程”

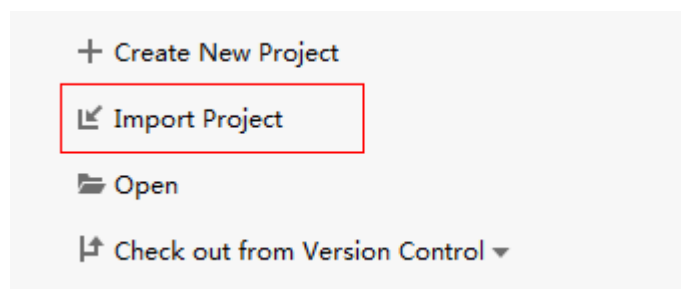


图 3-2 选择 “Maven”

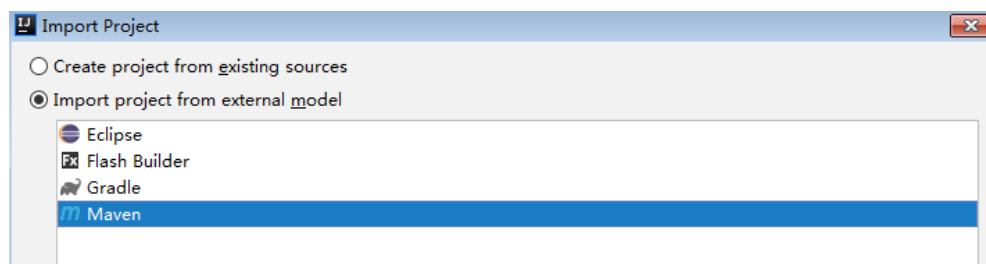
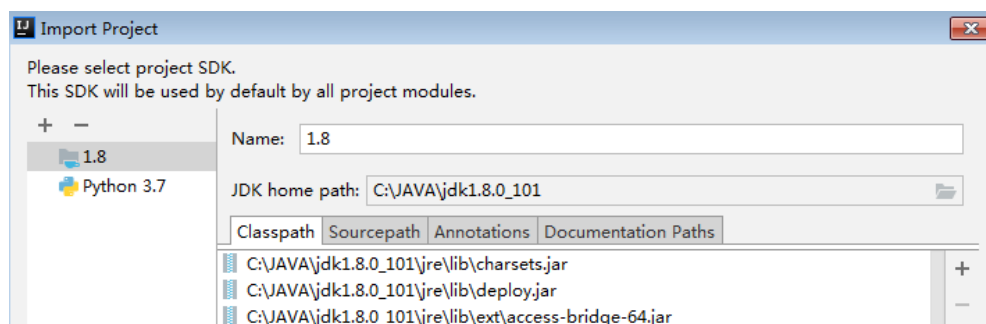
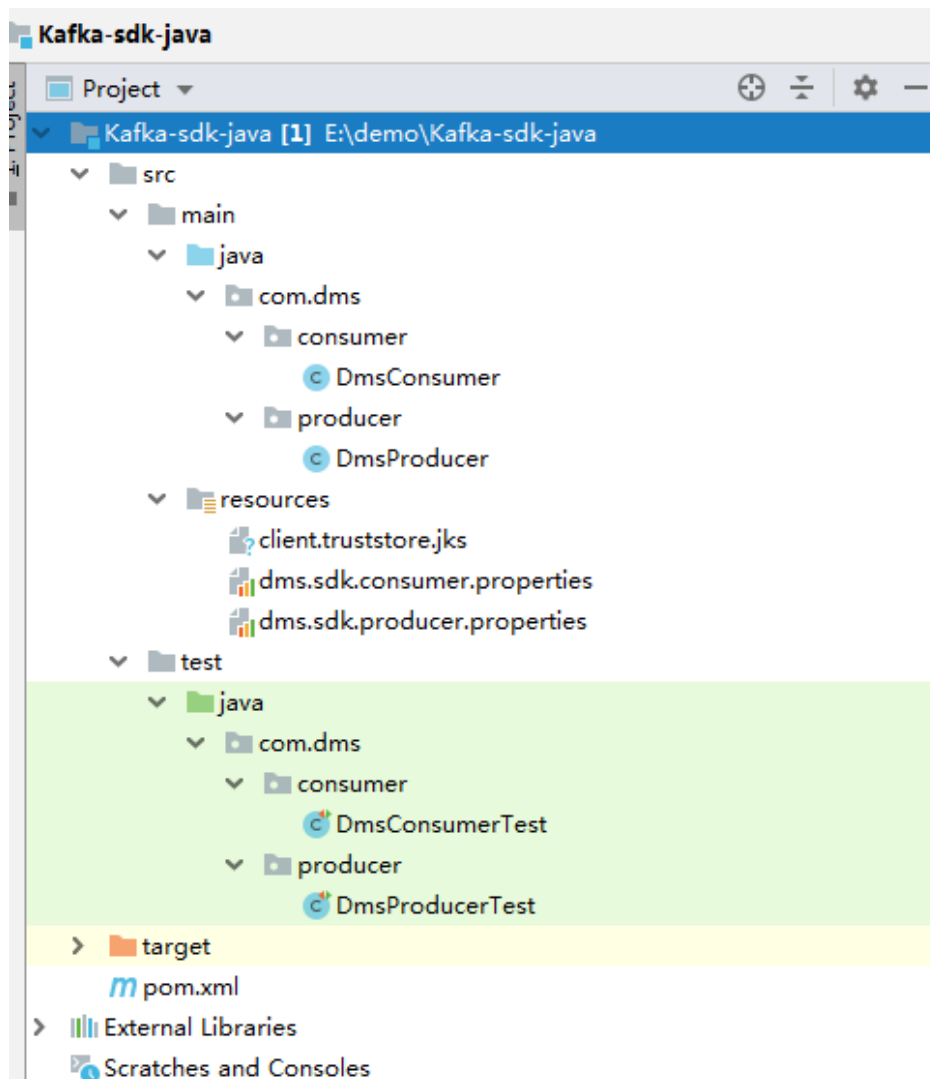


图 3-3 选择 Java 环境



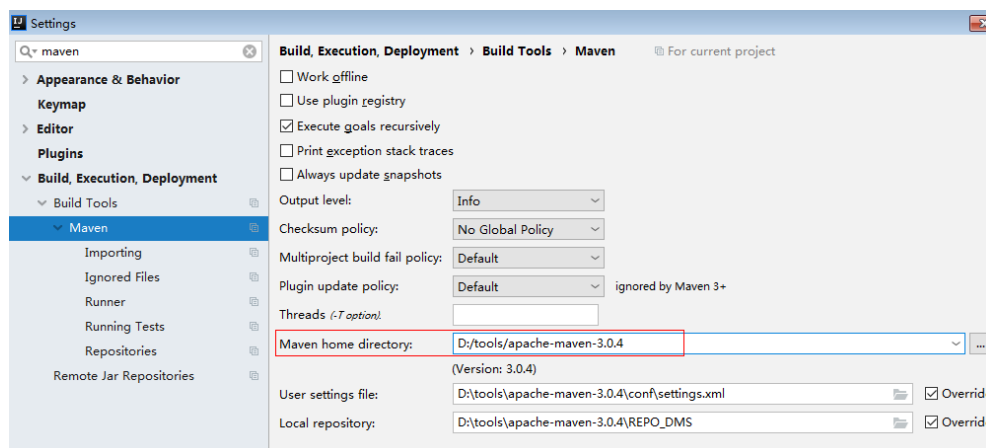
其他选项可默认或自主选择。然后单击Finish，完成Demo导入。

导入后Demo工程如下：



步骤3 配置Maven路径。

打开“File > Settings”，找到“Maven home directory”信息项，选择正确的Maven路径，以及Maven所需的settings.xml文件。

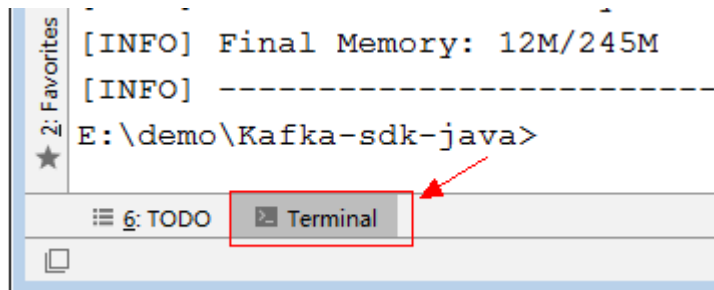


步骤4 修改Kafka配置信息。

以生产消息为例，具体修改请参考[生产消息配置文件](#)。

步骤5 在IDEA工具的左下角，打开Terminal窗口，执行**mvn test**命令体验demo。

图 3-4 IDEA 的 Terminal 窗口位置



生产消息会得到以下回显信息：

```
-----
T E S T S
-----
Running com.dms.producer.DmsProducerTest
produce msg:The msg is 0
produce msg:The msg is 1
produce msg:The msg is 2
produce msg:The msg is 3
produce msg:The msg is 4
produce msg:The msg is 5
produce msg:The msg is 6
produce msg:The msg is 7
produce msg:The msg is 8
produce msg:The msg is 9
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 138.877 sec
```

消费消息会得到以下回显信息：

```
-----
T E S T S
-----
Running com.dms.consumer.DmsConsumerTest
the numbers of topic:0
the numbers of topic:0
the numbers of topic:6
ConsumerRecord(topic = topic-0, partition = 2, offset = 0, CreateTime = 1557059377179, serialized key size = -1, serialized value size = 12, headers = RecordHeaders(headers = [], isReadOnly = false), key = null, value = The msg is 2)
ConsumerRecord(topic = topic-0, partition = 2, offset = 1, CreateTime = 1557059377195, serialized key size = -1, serialized value size = 12, headers = RecordHeaders(headers = [], isReadOnly = false), key = null, value = The msg is 5)
```

----**结束**

4 Python

本文以Linux CentOS环境为例，介绍Python版本的Kafka客户端连接指导，包括Kafka客户端安装，以及生产、消费消息。

使用前请参考[收集连接信息](#)收集Kafka所需的连接信息。

准备环境

- Python
一般系统预装了Python，您可以在命令行输入**python**或者**python3**，查看Python是否已经安装。**python**命令只能查询Python 2.x版本，**python3**命令只能查询Python 3.x版本，如果无法确认Python版本，请分别输入两个命令查看结果。

以Python 3.x为例，得到如下回显，说明Python已安装。

```
[root@ecs-test python-kafka]# python3
Python 3.7.1 (default, Jul 5 2020, 14:37:24)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

如果未安装Python，请使用以下命令安装：

```
yum install python
```

- Python版的Kafka客户端
执行以下命令，安装推荐版本的kafka-python：
 - Python 2.x版本：**pip install kafka-python==2.0.1**
 - Python 3.x版本：**pip3 install kafka-python==2.0.1**

生产消息

步骤1 在客户端创建一个文件，用于存放生产消息的代码示例。

```
touch producer.py
```

producer.py表示文件名，您可以自定义文件名。

步骤2 执行以下命令，编辑文件。

```
vim producer.py
```

步骤3 将以下生产消息的代码示例写入文件中，并保存。

- SASL认证方式

```
from kafka import KafkaProducer
import ssl
```

```
##连接信息
conf = {
    'bootstrap_servers': ["ip1:port1","ip2:port2","ip3:port3"],
    'topic_name': 'topic_name',
    'sasl_username': 'username',
    'sasl_password': 'password'
}

context = ssl.create_default_context()
context = ssl.SSLContext(ssl.PROTOCOL_SSLv23)
##如果Kafka安全协议设置为“SASL_PLAINTEXT”，请注释以下参数。
context.verify_mode = ssl.CERT_REQUIRED
##证书文件，SSL证书参考“收集连接信息”章节获取。如果Kafka安全协议设置为
“SASL_PLAINTEXT”，请注释以下参数。
context.load_verify_locations("phy_ca.crt")

print('start producer')
producer = KafkaProducer(bootstrap_servers=conf['bootstrap_servers'],
                        sasl_mechanism="PLAIN",
                        ssl_context=context,
                        security_protocol='SASL_SSL',
                        sasl_plain_username=conf['sasl_username'],
                        sasl_plain_password=conf['sasl_password'])

data = bytes("hello kafka!", encoding="utf-8")
producer.send(conf['topic_name'], data)
producer.close()
print('end producer')
```

示例代码中的参数说明如下，请参考[收集连接信息](#)获取参数值。

- bootstrap_servers: 实例连接地址与端口。
- topic_name: Topic名称。
- sasl_plain_username/sasl_plain_password: 首次开启密文接入时设置的用户名与密码，或者创建用户时设置的用户名和密码。为了确保用户名和密码的安全性，建议对用户名和密码进行加密处理，使用时解密。
- context.load_verify_locations: 证书文件。如果Kafka安全协议设置为“SASL_SSL”，需要设置此参数。使用Python语言连接实例时，需要用CRT格式的证书。
- sasl_mechanism: SASL认证机制。在Kafka实例控制台的基本信息页面中获取。如果SCRAM-SHA-512和PLAIN都开启了，根据实际情况选择其中任何一种配置连接。很久前创建的Kafka实例在详情页如果未显示“开启的SASL认证机制”，默认使用PLAIN机制。
- security_protocol: Kafka的安全协议。在Kafka实例控制台的基本信息页面中获取。很久前创建的Kafka实例在详情页如果未显示“启用的安全协议”，默认使用SASL_SSL协议。
 - 安全协议设置为“SASL_SSL”时，采用SASL方式进行认证，数据通过SSL证书进行加密传输，安全性更高。此时需要配置连接实例的用户名和密码，以及证书文件。
 - 安全协议设置为“SASL_PLAINTEXT”时，采用SASL方式进行认证，数据通过明文传输，性能更好。此时需要配置连接实例的用户名和密码，无需配置证书文件。

- 非SASL认证方式

```
from kafka import KafkaProducer

conf = {
    'bootstrap_servers': ["ip1:port1","ip2:port2","ip3:port3"],
    'topic_name': 'topic_name',
}
```

```
print('start producer')
producer = KafkaProducer(bootstrap_servers=conf['bootstrap_servers'])

data = bytes("hello kafka!", encoding="utf-8")
producer.send(conf['topic_name'], data)
producer.close()
print('end producer')
```

示例代码中的参数说明如下，请参考[收集连接信息](#)获取参数值。

- bootstrap_servers: 实例连接地址与端口。
- topic_name: Topic名称。

步骤4 执行以下命令，运行生产消息的代码示例。

```
# Python 2.x版本
python producer.py
```

```
# Python 3.x版本
python3 producer.py
```

运行成功后，返回如下回显。

```
[root@ecs-test ~]# python3 producer.py
start producer
end producer
[root@ecs-test ~]#
```

----结束

消费消息

步骤1 在客户端创建一个文件，用于存放消费消息的代码示例。

```
touch consumer.py
```

consumer.py表示文件名，您可以自定义文件名。

步骤2 执行以下命令，编辑文件。

```
vim consumer.py
```

步骤3 将以下消费消息的代码示例写入文件中，并保存。

- SASL认证方式

```
from kafka import KafkaConsumer
import ssl
##连接信息
conf = {
    'bootstrap_servers': ["ip1:port1","ip2:port2","ip3:port3"],
    'topic_name': 'topic_name',
    'sasl_username': 'username',
    'sasl_password': 'password',
    'consumer_id': 'consumer_id'
}

context = ssl.create_default_context()
context = ssl.SSLContext(ssl.PROTOCOL_SSLv23)
##如果Kafka安全协议设置为“SASL_PLAINTEXT”，请注释以下参数。
context.verify_mode = ssl.CERT_REQUIRED
##证书文件，SSL证书参考“收集连接信息”章节获取。如果Kafka安全协议设置为
“SASL_PLAINTEXT”，请注释以下参数。
context.load_verify_locations("phy_ca.crt")

print('start consumer')
consumer = KafkaConsumer(conf['topic_name'],
                          bootstrap_servers=conf['bootstrap_servers'],
                          group_id=conf['consumer_id'],
                          sasl_mechanism="PLAIN",
```

```
ssl_context=context,
security_protocol='SASL_SSL',
sasl_plain_username=conf['sasl_username'],
sasl_plain_password=conf['sasl_password'])

for message in consumer:
    print("%s:%d:%d: key=%s value=%s" % (message.topic, message.partition,message.offset,
message.key,message.value))

print('end consumer')
```

示例代码中的参数说明如下，请参考[收集连接信息](#)获取参数值。

- bootstrap_servers: 实例连接地址与端口。
- topic_name: Topic名称。
- sasl_plain_username/sasl_plain_password: 首次开启密文接入时设置的用户名与密码，或者创建用户时设置的用户名和密码。为了确保用户名和密码的安全性，建议对用户名和密码进行加密处理，使用时解密。
- consumer_id: 消费组名称。根据业务需求，自定义消费组名称，如果设置的消费组不存在，Kafka会自动创建。
- context.load_verify_locations: 证书文件。如果Kafka安全协议设置为“SASL_SSL”，需要设置此参数。使用Python语言连接实例时，需要用CRT格式的证书。
- sasl_mechanism: SASL认证机制。在Kafka实例控制台的基本信息页面中获取。如果SCRAM-SHA-512和PLAIN都开启了，根据实际情况选择其中任何一种配置连接。很久前创建的Kafka实例在详情页如果未显示“开启的SASL认证机制”，默认使用PLAIN机制。
- security_protocol: Kafka的安全协议。在Kafka实例控制台的基本信息页面中获取。很久前创建的Kafka实例在详情页如果未显示“启用的安全协议”，默认使用SASL_SSL协议。
 - 安全协议设置为“SASL_SSL”时，采用SASL方式进行认证，数据通过SSL证书进行加密传输，安全性更高。此时需要配置连接实例的用户名和密码，以及证书文件。
 - 安全协议设置为“SASL_PLAINTEXT”时，采用SASL方式进行认证，数据通过明文传输，性能更好。此时需要配置连接实例的用户名和密码，无需配置证书文件。

- 非SASL认证方式

```
from kafka import KafkaConsumer

conf = {
    'bootstrap_servers': ["ip1:port1","ip2:port2","ip3:port3"],
    'topic_name': 'topic-name',
    'consumer_id': 'consumer-id'
}

print('start consumer')
consumer = KafkaConsumer(conf['topic_name'],
    bootstrap_servers=conf['bootstrap_servers'],
    group_id=conf['consumer_id'])

for message in consumer:
    print("%s:%d:%d: key=%s value=%s" % (message.topic, message.partition,message.offset,
message.key,message.value))

print('end consumer')
```

示例代码中的参数说明如下，请参考[收集连接信息](#)获取参数值。

- bootstrap_servers: 实例连接地址与端口。
- topic_name: Topic名称。
- consumer_id: 消费组名称。根据业务需求，自定义消费组名称，如果设置的消费组不存在，Kafka会自动创建。

步骤4 执行以下命令，运行消费消息的代码示例。

```
# Python 2.x版本  
python consumer.py
```

```
# Python 3.x版本  
python3 consumer.py
```

运行成功后，返回如下回显。

```
[root@ecs-test ~]# python3 consumer.py  
start consumer
```

如需停止消费使用**Ctrl+C**命令退出。

----**结束**

5 Go

本文以Linux CentOS环境为例，介绍Go 1.16.5版本的Kafka客户端连接指导，包括demo代码库的获取，以及生产、消费消息。

使用前请参考[收集连接信息](#)收集Kafka所需的连接信息。

准备环境

- 执行以下命令，检查是否已安装Go。

```
go version
```

返回如下回显时，说明Go已经安装。

```
[root@ecs-test confluent-kafka-go]# go version  
go version go1.16.5 linux/amd64
```

如果未安装Go，参考如下步骤安装。

```
#下载Go安装包。
```

```
wget https://go.dev/dl/go1.16.5.linux-amd64.tar.gz
```

```
#解压安装包到“/usr/local”目录，“/usr/local”目录可以根据实际情况修改。  
sudo tar -C /usr/local -xzf go1.16.5.linux-amd64.tar.gz
```

```
#设置环境变量。
```

```
echo 'export PATH=$PATH:/usr/local/go/bin' >> ~/.profile  
source ~/.profile
```

- 执行以下命令，获取demo需要的代码库。

```
go get github.com/confluentinc/confluent-kafka-go/kafka
```

生产消息

- SASL认证方式

```
package main
```

```
import (  
    "bufio"  
    "fmt"  
    "github.com/confluentinc/confluent-kafka-go/kafka"  
    "log"  
    "os"  
    "os/signal"  
    "syscall"  
)
```

```
var (  
    brokers = "ip1:port1,ip2:port2,ip3:port3"  
    topics  = "topic_name"  
    user    = "username"
```

```
password = "password"
caFile = "phy_ca.crt" //SSL证书参考“收集连接信息”章节获取。如果Kafka安全协议设置为
“SASL_PLAINTEXT”，请删除此参数。
)

func main() {
    log.Println("Starting a new kafka producer")

    config := &kafka.ConfigMap{
        "bootstrap.servers": brokers,
        "security.protocol": "SASL_SSL",
        "sasl.mechanism": "PLAIN",
        "sasl.username": user,
        "sasl.password": password,
        "ssl.ca.location": caFile, //如果Kafka安全协议设置为“SASL_PLAINTEXT”，请删除此参
数。
        "ssl.endpoint.identification.algorithm": "none",
    }
    producer, err := kafka.NewProducer(config)
    if err != nil {
        log.Panicf("producer error, err: %v", err)
        return
    }

    go func() {
        for e := range producer.Events() {
            switch ev := e.(type) {
            case *kafka.Message:
                if ev.TopicPartition.Error != nil {
                    log.Printf("Delivery failed: %v\n", ev.TopicPartition)
                } else {
                    log.Printf("Delivered message to %v\n", ev.TopicPartition)
                }
            }
        }
    }()

    // Produce messages to topic (asynchronously)
    fmt.Println("please enter message:")
    go func() {
        for {
            err := producer.Produce(&kafka.Message{
                TopicPartition: kafka.TopicPartition{Topic: &topics, Partition: kafka.PartitionAny},
                Value:            GetInput(),
            }, nil)
            if err != nil {
                log.Panicf("send message fail, err: %v", err)
                return
            }
        }
    }()

    sigterm := make(chan os.Signal, 1)
    signal.Notify(sigterm, syscall.SIGINT, syscall.SIGTERM)
    select {
    case <-sigterm:
        log.Println("terminating: via signal")
    }
    // Wait for message deliveries before shutting down
    producer.Flush(15 * 1000)
    producer.Close()
}

func GetInput() []byte {
    reader := bufio.NewReader(os.Stdin)
    data, _, _ := reader.ReadLine()
    return data
}
```

示例代码中的参数说明如下，请参考[收集连接信息](#)获取参数值。

- brokers: 实例连接地址与端口。
- topics: Topic名称。
- user/password: 首次开启密文接入时设置的用户名与密码, 或者创建用户时设置的用户名和密码。为了确保用户名和密码的安全性, 建议对用户名和密码进行加密处理, 使用时解密。
- caFile: 证书文件。如果Kafka安全协议设置为“SASL_SSL”, 需要设置此参数。
- security.protocol: Kafka的安全协议。在Kafka实例控制台的基本信息页面中获取。很久前创建的Kafka实例在详情页如果未显示“启用的安全协议”, 默认使用SASL_SSL协议。
 - 安全协议设置为“SASL_SSL”时, 采用SASL方式进行认证, 数据通过SSL证书进行加密传输, 安全性更高。此时需要配置连接实例的用户名和密码, 以及证书文件。
 - 安全协议设置为“SASL_PLAINTEXT”时, 采用SASL方式进行认证, 数据通过明文传输, 性能更好。此时需要配置连接实例的用户名和密码, 无需配置证书文件。
- sasl.mechanism: SASL认证机制。在Kafka实例控制台的基本信息页面中获取。如果SCRAM-SHA-512和PLAIN都开启了, 根据实际情况选择其中任何一种配置连接。很久前创建的Kafka实例在详情页如果未显示“开启的SASL认证机制”, 默认使用PLAIN机制。

- 非SASL认证方式

```
package main

import (
    "bufio"
    "fmt"
    "github.com/confluentinc/confluent-kafka-go/kafka"
    "log"
    "os"
    "os/signal"
    "syscall"
)

var (
    brokers = "ip1:port1,ip2:port2,ip3:port3"
    topics = "topic_name"
)

func main() {
    log.Println("Starting a new kafka producer")

    config := &kafka.ConfigMap{
        "bootstrap.servers": brokers,
    }
    producer, err := kafka.NewProducer(config)
    if err != nil {
        log.Panicf("producer error, err: %v", err)
        return
    }

    go func() {
        for e := range producer.Events() {
            switch ev := e.(type) {
            case *kafka.Message:
                if ev.TopicPartition.Error != nil {
                    log.Printf("Delivery failed: %v\n", ev.TopicPartition)
                } else {
                    log.Printf("Delivered message to %v\n", ev.TopicPartition)
                }
            }
        }
    }
}
```

```
    }
  }
}()

// Produce messages to topic (asynchronously)
fmt.Println("please enter message:")
go func() {
  for {
    err := producer.Produce(&kafka.Message{
      TopicPartition: kafka.TopicPartition{Topic: &topics, Partition: kafka.PartitionAny},
      Value:          GetInput(),
    }, nil)
    if err != nil {
      log.Panicf("send message fail, err: %v", err)
      return
    }
  }
}()

sigterm := make(chan os.Signal, 1)
signal.Notify(sigterm, syscall.SIGINT, syscall.SIGTERM)
select {
case <-sigterm:
  log.Println("terminating: via signal")
}
// Wait for message deliveries before shutting down
producer.Flush(15 * 1000)
producer.Close()
}

func GetInput() []byte {
  reader := bufio.NewReader(os.Stdin)
  data, _, _ := reader.ReadLine()
  return data
}
}
```

示例代码中的参数说明如下，请参考[收集连接信息](#)获取参数值。

- brokers: 实例连接地址与端口。
- topics: Topic名称。

消费消息

- SASL认证方式

```
package main

import (
  "fmt"
  "github.com/confluentinc/confluent-kafka-go/kafka"
  "log"
  "os"
  "os/signal"
  "syscall"
)

var (
  brokers = "ip1:port1,ip2:port2,ip3:port3"
  group   = "group-id"
  topics  = "topic_name"
  user    = "username"
  password = "password"
  caFile  = "phy_ca.crt" //SSL证书参考“收集连接信息”章节获取。如果Kafka安全协议设置为
                        “SASL_PLAINTEXT”，请删除此参数。
)

func main() {
  log.Println("Starting a new kafka consumer")

  config := &kafka.ConfigMap{
```

```
"bootstrap.servers": brokers,
"group.id":      group,
"auto.offset.reset": "earliest",
"security.protocol": "SASL_SSL",
"sasl.mechanism":  "PLAIN",
"sasl.username":  user,
"sasl.password":  password,
"ssl.ca.location": caFile, //如果Kafka安全协议设置为“SASL_PLAINTEXT”，请删除此参
数。
"ssl.endpoint.identification.algorithm": "none",
}

consumer, err := kafka.NewConsumer(config)
if err != nil {
    log.Panicf("Error creating consumer: %v", err)
    return
}

err = consumer.SubscribeTopics([]string{topics}, nil)
if err != nil {
    log.Panicf("Error subscribe consumer: %v", err)
    return
}

go func() {
    for {
        msg, err := consumer.ReadMessage(-1)
        if err != nil {
            log.Printf("Consumer error: %v (%v)", err, msg)
        } else {
            fmt.Printf("Message on %s: %s\n", msg.TopicPartition, string(msg.Value))
        }
    }
}()

sigterm := make(chan os.Signal, 1)
signal.Notify(sigterm, syscall.SIGINT, syscall.SIGTERM)
select {
case <-sigterm:
    log.Println("terminating: via signal")
}
if err = consumer.Close(); err != nil {
    log.Panicf("Error closing consumer: %v", err)
}
}
```

示例代码中的参数说明如下，请参考[收集连接信息](#)获取参数值。

- **brokers**: 实例连接地址与端口。
- **group**: 消费组名称。根据业务需求，自定义消费组名称，如果设置的消费组不存在，Kafka会自动创建。
- **topics**: Topic名称。
- **user/password**: 首次开启密文接入时设置的用户名与密码，或者创建用户时设置的用户名和密码。为了确保用户名和密码的安全性，建议对用户名和密码进行加密处理，使用时解密。
- **caFile**: 证书文件。如果Kafka安全协议设置为“SASL_SSL”，需要设置此参数。
- **security.protocol**: Kafka的安全协议。在Kafka实例控制台的基本信息页面中获取。很久前创建的Kafka实例在详情页如果未显示“启用的安全协议”，默认使用SASL_SSL协议。
 - 安全协议设置为“SASL_SSL”时，采用SASL方式进行认证，数据通过SSL证书进行加密传输，安全性更高。此时需要配置连接实例的用户名和密码，以及证书文件。

- 安全协议设置为“SASL_PLAINTEXT”时，采用SASL方式进行认证，数据通过明文传输，性能更好。此时需要配置连接实例的用户名和密码，无需配置证书文件。
 - sasl.mechanism: SASL认证机制。在Kafka实例控制台的基本信息页面中获取。如果SCRAM-SHA-512和PLAIN都开启了，根据实际情况选择其中任意一种配置连接。很久前创建的Kafka实例在详情页如果未显示“开启的SASL认证机制”，默认使用PLAIN机制。
- 非SASL认证方式

```
package main

import (
    "fmt"
    "github.com/confluentinc/confluent-kafka-go/kafka"
    "log"
    "os"
    "os/signal"
    "syscall"
)

var (
    brokers = "ip1:port1,ip2:port2,ip3:port3"
    group   = "group-id"
    topics  = "topic_name"
)

func main() {
    log.Println("Starting a new kafka consumer")

    config := &kafka.ConfigMap{
        "bootstrap.servers": brokers,
        "group.id":          group,
        "auto.offset.reset": "earliest",
    }

    consumer, err := kafka.NewConsumer(config)
    if err != nil {
        log.Panicf("Error creating consumer: %v", err)
        return
    }

    err = consumer.SubscribeTopics([]string{topics}, nil)
    if err != nil {
        log.Panicf("Error subscribe consumer: %v", err)
        return
    }

    go func() {
        for {
            msg, err := consumer.ReadMessage(-1)
            if err != nil {
                log.Printf("Consumer error: %v (%v)", err, msg)
            } else {
                fmt.Printf("Message on %s: %s\n", msg.TopicPartition, string(msg.Value))
            }
        }
    }()

    sigterm := make(chan os.Signal, 1)
    signal.Notify(sigterm, syscall.SIGINT, syscall.SIGTERM)
    select {
    case <-sigterm:
        log.Println("terminating: via signal")
    }
    if err = consumer.Close(); err != nil {
        log.Panicf("Error closing consumer: %v", err)
    }
}
```

```
}  
}
```

示例代码中的参数说明如下，请参考[收集连接信息](#)获取参数值。

- brokers: 实例连接地址与端口。
- group: 消费组名称。根据业务需求，自定义消费组名称，如果设置的消费组不存在，Kafka会自动创建。
- topics: Topic名称。

6 获取 Kafka 开源客户端

Kafka实例支持多种语言的客户端，如果您想要使用其他语言连接Kafka实例，可以从[Kafka官网获取客户端](#)。

Kafka官网支持的客户端语言包括：C/C++、Python、Go、PHP、Node.js等等。

Kafka实例完全兼容开源客户端，按照Kafka官网提供的连接说明，与Kafka实例对接。

7 spring-kafka 的使用

本文介绍如何使用spring-kafka连接华为云Kafka实例进行消息的生产和消费。相关代码您可以从[kafka-springboot-demo](#)中获取。

下文所有Kafka的配置信息，如实例连接地址、Topic名称、用户信息等，请参考[收集连接信息](#)获取。

在 pom.xml 文件中引入 spring-kafka 依赖

```
<dependency>
  <groupId>org.springframework.kafka</groupId>
  <artifactId>spring-kafka</artifactId>
</dependency>
```

在 application.properties 文件中填写配置

```
##### Kafka #####
## Kafka实例的broker信息, ip:port为实例的连接地址和端口
spring.kafka.bootstrap-servers=ip1:port1,ip2:port2,ip3:port3
##### 生产者配置 #####
spring.kafka.producer.retries=0
spring.kafka.producer.batch-size=16384
spring.kafka.producer.buffer-memory=33554432
spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer=org.apache.kafka.common.serialization.StringSerializer
##### 消费者配置 #####
spring.kafka.consumer.group-id=test-consumer-group
spring.kafka.consumer.auto-offset-reset=earliest
spring.kafka.consumer.enable-auto-commit=true
spring.kafka.consumer.auto-commit-interval=100
spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer=org.apache.kafka.common.serialization.StringDeserializer
##### SASL配置(不开启SASL时将以下配置删除) #####
## 设置SASL认证机制、账号和密码。
## spring.kafka.properties.sasl.mechanism为SASL认证机制, username和密码为SASL的用户名和密码, 参考“收集连接信息”章节获取。为了确保用户名和密码的安全性, 建议对用户名和密码进行加密处理, 使用时解密。
## SASL认证机制为“PLAIN”时, 配置信息如下。
spring.kafka.properties.sasl.mechanism=PLAIN
spring.kafka.properties.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
  username="username" \
  password="password";
## SASL认证机制为“SCRAM-SHA-512”时, 配置信息如下。
spring.kafka.properties.sasl.mechanism=SCRAM-SHA-512
spring.kafka.properties.sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule
required \
  username="username" \
  password="password";
```

```
## 设置Kafka安全协议。spring.kafka.security.protocol为安全协议。  
## 安全协议为“SASL_SSL”时，配置信息如下。  
spring.kafka.security.protocol=SASL_SSL  
## spring.kafka.ssl.trust-store-location为SSL证书的存放路径，如下代码以Windows系统路径格式举例，您在使用时请根据实际运行环境修改路径格式。  
spring.kafka.ssl.trust-store-location=file:D:\\temp\\client.jks  
## spring.kafka.ssl.trust-store-password为服务器证书密码，无需修改，配置此密码是为了访问Java生成的jks文件。  
spring.kafka.ssl.trust-store-password=dms@kafka  
## spring.kafka.properties.ssl.endpoint.identification.algorithm为证书域名校验开关，为空则表示关闭，这里需要保持关闭状态，必须设置为空。  
spring.kafka.properties.ssl.endpoint.identification.algorithm=  
## 安全协议为“SASL_PLAINTEXT”时，配置信息如下。  
spring.kafka.security.protocol=SASL_PLAINTEXT
```

生产消息

```
package com.huaweicloud.dms.example.producer;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.kafka.core.KafkaTemplate;  
import org.springframework.scheduling.annotation.Scheduled;  
import org.springframework.stereotype.Component;  
  
import java.util.UUID;  
  
/**  
 * @author huaweicloud DMS  
 */  
@Component  
public class DmsKafkaProducer {  
    /**  
     * Topic名称,根据实际情况修改  
     */  
    public static final String TOPIC = "test_topic";  
  
    @Autowired  
    private KafkaTemplate<String, String> kafkaTemplate;  
  
    /**  
     * 定时任务每5秒钟生产一条消息  
     */  
    @Scheduled(cron = "*/*5 * * * * ?")  
    public void send() {  
        String message = String.format("{id:%s,timestamp:%s}", UUID.randomUUID().toString(),  
System.currentTimeMillis());  
        kafkaTemplate.send(TOPIC, message);  
        System.out.println("send finished, message = " + message);  
    }  
}
```

消费消息

```
package com.huaweicloud.dms.example.consumer;  
  
import org.apache.kafka.clients.consumer.ConsumerRecord;  
import org.springframework.kafka.annotation.KafkaListener;  
import org.springframework.stereotype.Component;  
  
import java.util.Optional;  
  
/**  
 * @author huaweicloud DMS  
 */  
@Component  
public class DmsKafkaConsumer {  
    /**  
     * Topic名称, 根据实际情况修改  
     */  
}
```

```
*/  
private static final String TOPIC = "test_topic";  
  
@KafkaListener(topics = {TOPIC})  
public void listen(ConsumerRecord<String, String> record) {  
    Optional<String> message = Optional.ofNullable(record.value());  
    if (message.isPresent()) {  
        System.out.println("consume finished, message = " + message.get());  
    }  
}  
}
```