

分布式消息服务 Kafka

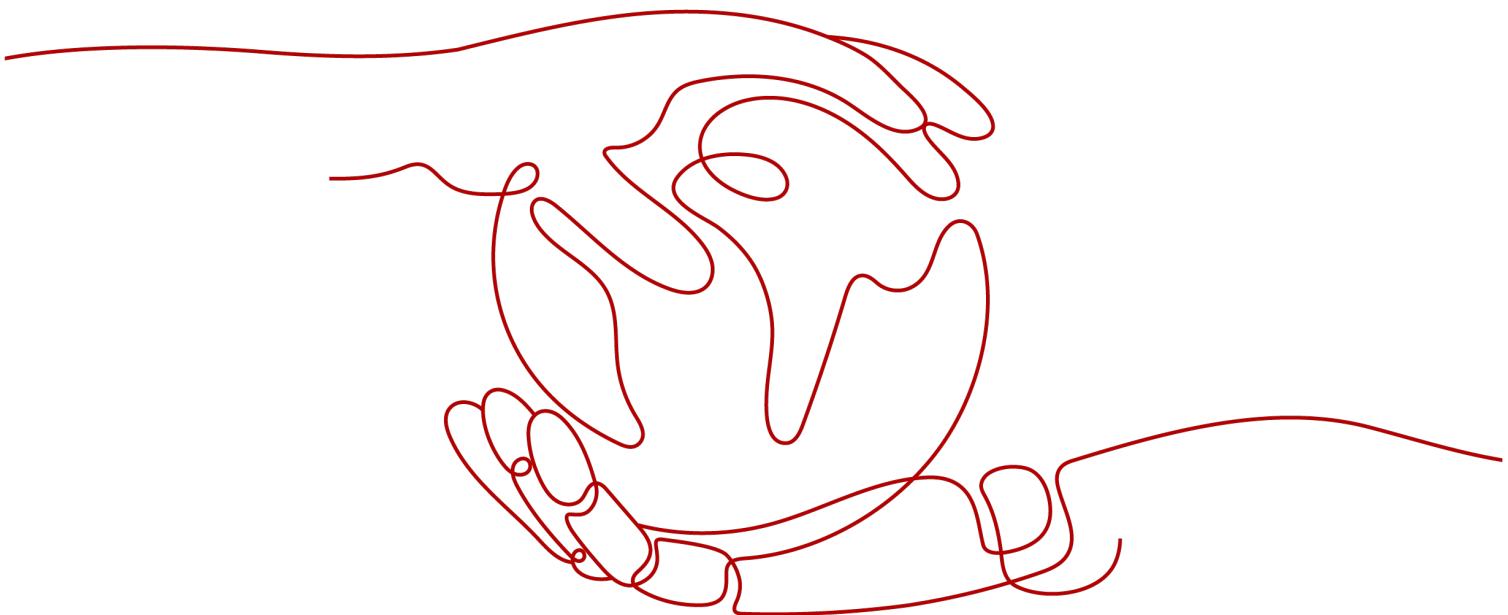
# 开发指南

文档版本

01

发布日期

2020-12-02



**版权所有 © 华为技术有限公司 2021。保留一切权利。**

未经本公司书面许可，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，华为公司对本文档内容不做任何明示或默示的声明或保证。

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

---

<b>1 概述.....</b>	<b>1</b>
<b>2 收集连接信息.....</b>	<b>2</b>
<b>3 Java.....</b>	<b>3</b>
3.1 Java 客户端接入示例.....	3
3.2 Java 开发环境搭建.....	10
<b>4 Python.....</b>	<b>15</b>
<b>5 Kafka 开源客户端获取.....</b>	<b>18</b>
<b>6 客户端使用建议.....</b>	<b>19</b>
<b>A 修订记录.....</b>	<b>20</b>

# 1 概述

专享版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。如果使用SASL认证方式，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、SASL连接使用的证书、公网访问信息等，然后提供Java、Python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

客户端有3种方式访问Kafka实例：

### 1. VPC内子网地址访问

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VPC内子网地址。

### 2. VPC对等连接方式访问

如果客户端是云上ECS，与Kafka实例处于相同region但不同VPC，则可以通过建立VPC对等连接后，访问Kafka实例提供的VPC内子网地址。

### 3. 公网访问

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9095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

## Kafka 实例信息准备

### 1. 实例连接地址与端口

实例创建后，从实例的基本信息页签中获取，在客户端配置时，可将地址都配上。如下所示，规格为100M/s的Kafka实例，会有三个broker地址，其他规格的broker节点个数，请以实际为准。

如果开启公网访问，还可以使用基本信息页签下方得公网访问地址连接Kafka的Topic。

图 2-1 查看 Kafka 实例 Broker 连接地址与端口

连接地址

IPV4

192.168.1.5:9092,192.168.1.143:9092,192.168.1.48:9092

### 2. Topic名称

从实例的Topic管理页签中获取Topic名称。

### 3. SASL信息

如果实例创建时开启SASL方式访问，则需要获得在开启SASL\_SSL时输入的用户名与密码，以及[下载SSL证书](#)。

SASL用户名在实例基本页签中查看，如果忘记初始设置的密码，可通过“重置 Kafka密码”操作重新获得。

# 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专享实例基于社区版本2.3.0，推荐客户端保持一致。  
<dependency>  
    <groupId>org.apache.kafka</groupId>  
    <artifactId>kafka-clients</artifactId>  
    <version>2.3.0</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SASL，使用的是不加密连接，请注释相关代码；如果Kafka实例开启了SASL，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SASL认证，以下参数请注释掉。
#####
#设置jaas帐号和密码，username和password为创建Kafka实例过程中开启SASL_SSL时填入的用户名和密码。
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
    username="username" \
    password="password"; 
#SASL鉴权方式
sasl.mechanism=PLAIN
#加密协议，目前支持SASL_SSL协议
security.protocol=SASL_SSL
#ssl truststore文件的位置
ssl.truststore.location=E:\\temp\\client.truststore.jks
#ssl truststore文件的密码
ssl.truststore.password=dms@kafka
#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=
```

- 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都是属于同一个consumer group
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SASL认证，以下参数请注释掉。
#####
#设置jaas帐号和密码，username和password为创建Kafka实例过程中开启SASL_SSL时填入的用户名和密码。
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
    username="username" \
    password="password"; 
#SASL鉴权方式
sasl.mechanism=PLAIN
#加密协议，目前支持SASL_SSL协议
security.protocol=SASL_SSL
#ssl truststore文件的位置
ssl.truststore.location=E:\\temp\\client.truststore.jks
#ssl truststore文件的密码
ssl.truststore.password=dms@kafka
#关闭证书域名校验
ssl.endpoint.identification.algorithm=
```

## 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.BufferedReader;
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 BufferedReader(new InputStreamReader(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.BufferedReader;
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。

文件名	路径	说明
dms.sdk.producer.properties	.\src\main\resources	生产消息的配置信息。
client.truststore.jks	.\src\main\resources	SSL证书，用于SASL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”

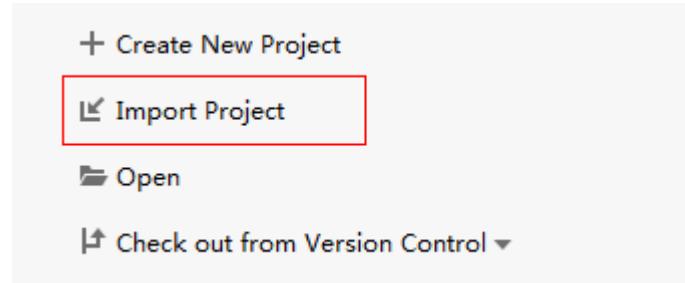


图 3-2 选择“Maven”

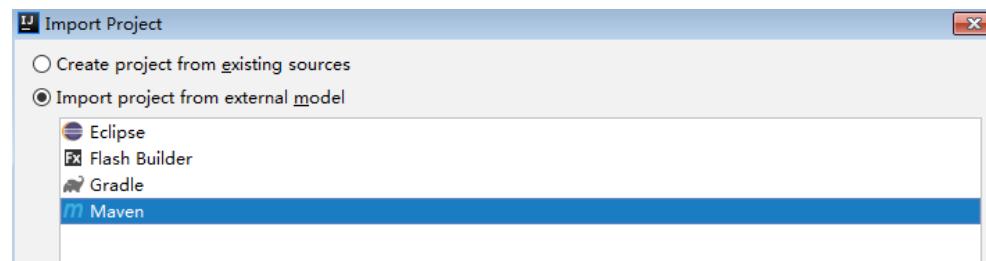
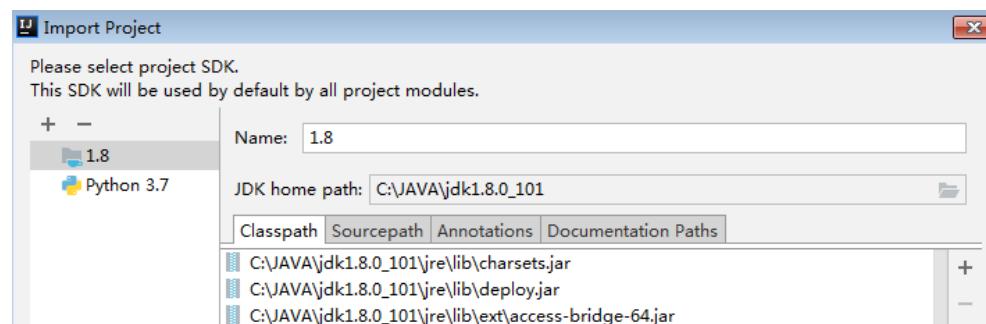
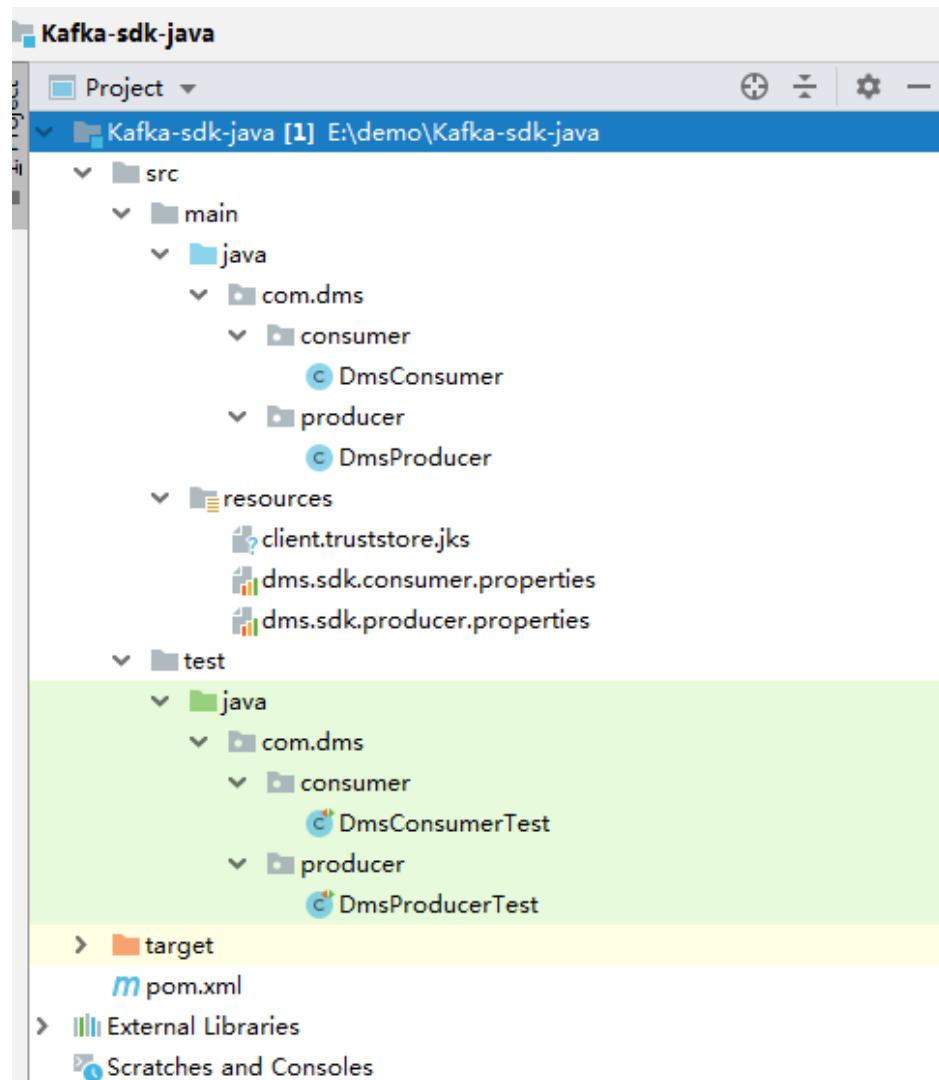


图 3-3 选择 Java 环境



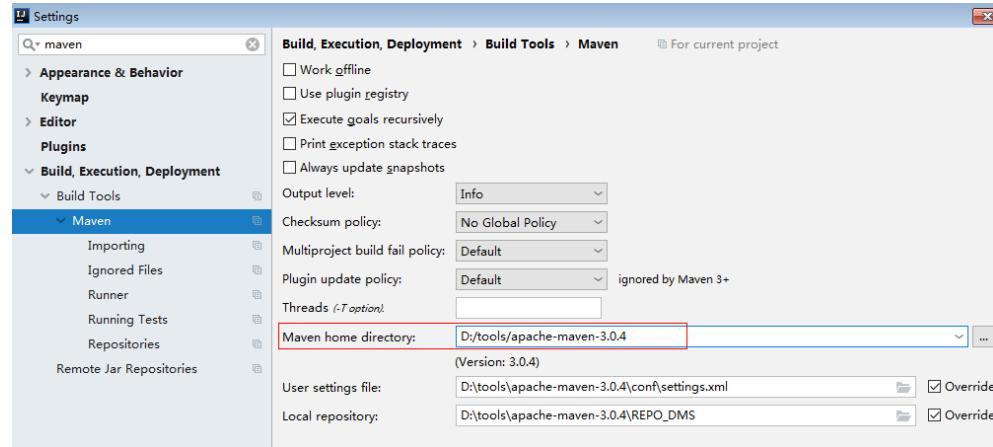
其他选项可默认或自主选择。然后单击Finish，完成Demo导入。

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，需配置以下信息，其中加粗内容必须修改。

```
#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SASL认证，以下参数请注释掉。
#####
#设置jaas帐号和密码，username和password为创建Kafka实例过程中开启SASL_SSL时填入的用户名和密码。
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
    username="username" \
    password="password";
```

#SASL鉴权方式

sasl.mechanism=PLAIN

#加密协议，目前支持SASL\_SSL协议

security.protocol=SASL\_SSL

#ssl truststore文件的位置

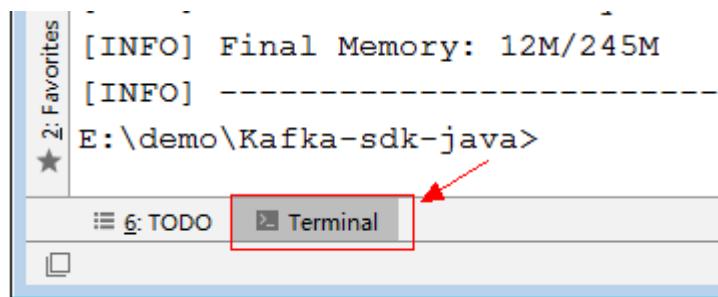
ssl.truststore.location=E:\\temp\\\\client.truststore.jks

#ssl truststore文件的密码

ssl.truststore.password=dms@kafka

#### 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。在命令行输入**python**，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：

**pip install kafka-python==2.0.1**

## 生产消息

### 说明

以下加粗内容需要替换为实例自有信息，请根据实际情况替换。

- 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_plain_username': 'username',
    'sasl_plain_password': 'password'
}

context = ssl.create_default_context()
context = ssl.SSLContext(ssl.PROTOCOL_SSLv23)
context.verify_mode = ssl.CERT_REQUIRED
```

```
##证书文件
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_plain_username'],
                        sasl_plain_password=conf['sasl_plain_password'])

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

- 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')
```

## 消费消息

- 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_plain_username': 'username',
    'sasl_plain_password': 'password',
    'consumer_id': 'consumer_id'
}

context = ssl.create_default_context()
context = ssl.SSLContext(ssl.PROTOCOL_SSLv23)
context.verify_mode = ssl.CERT_REQUIRED
##证书文件
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_plain_username'],
                        sasl_plain_password=conf['sasl_plain_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')
```

- 非SASL认证方式

注意，加粗内容需要替换为实例自有信息。

```
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')
```

# 5 Kafka 开源客户端获取

Kafka专享实例完全兼容开源客户端，如果您使用其他语言，也可以[从Kafka官网获取客户端](#)，按照Kafka官网提供的连接说明，与Kafka专享实例对接。

# 6 客户端使用建议

- 生产消息
  - 消息发送失败需要有重试机制。  
建议重试3次，通过参数：retries=3 配置。
  - 生产的callback函数不能阻塞，否则会阻塞客户端消息的发送  
对于时延敏感消息，设置发送优化：linger.ms=0。  
生产端的JVM内存要足够，避免内存不足导致发送阻塞。
- 消费消息
  - consumer的owner线程需确保不会异常退出，否则会导致客户端没有发起消费请求，阻塞消费。
  - 使用长连接poll模式消费消息，不要消费结束就关闭consumer通道，这样会导致频繁rebalance，阻塞消费。
  - consumer需周期性poll（建议间隔为200毫秒），维持和server端的心跳，避免因为心跳超时导致consumer频繁加入和退出，阻塞消费。
  - 消费线程退出要调用consumer的close方法，避免同一个组的其他消费者阻塞session.timeout.ms的时间。
  - consumer的session根据业务情况设置一个合理值，如30秒：  
session.timeout.ms=30000。
  - consumer数量不能超过topic的分区数，否则会有consumer拉取不到消息。
  - 确保处理完消息后再做消息commit，避免业务消息处理失败，无法重新拉取处理失败的消息。
  - consumer拉取的消息本地缓存应有大小限制，避免OOM（内存溢出）。
  - Kafka不能保证消费重复的消息，业务侧需保证消息处理的幂等性。

# A 修订记录

---

发布日期	修订记录
2020-12-02	第一次正式发布。