阿里云
消息队列 Kafka

开发指南

文档版本：20190920

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
	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。	 **禁止:** 重置操作将丢失用户配置数据。
	该类警示信息可能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，恢复业务所需时间约10分钟。
	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。	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。	单击 确定。
`courier` 字体	命令。	执行 `cd /d C:/windows` 命令，进入Windows系统文件夹。
*## *	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。	`swich` *{stand | slave}*

目录

1 SDK 参考

1.1 接入准备

本文主要介绍不同语言的客户端在接入消息队列 for Apache Kafka 前的通用准备。做好以下接入准备后，您便可开始使用 SDK 来访问消息队列 for Apache Kafka。

获取 Demo

消息队列 for Apache Kafka 提供多种客户端语言的 Demo，请前往消息队列 for Apache Kafka Demo 库查看和获取相应 Demo。使用之前请仔细阅读 *README.md* 文件。

Demo 通常分类为*vpc*和*vpc*目录：

- *vpc*目录下为 VPC 内访问 Demo
- *vpc*目录下为公网访问 Demo

消息队列 for Apache Kafka 目前已提供的 Demo 语言有 Java、C++、Python、Go、NodeJS、Logstash、SpringCloud、Filebeat、Stream 等。更多语言及插件的 Demo 将会不断完善，敬请期待。

准备客户端

消息队列 for Apache Kafka 服务端的版本为 0.10.0.0，建议客户端版本为 0.10.2.2。

创建资源

请提前创建 Topic 和 Consumer Group，并确保资源属于同一个实例。关于创建资源的具体操作，请参见#unique_5。

1.2 Java SDK 接入和使用说明

本文介绍如何通过 Java SDK 接入消息队列 for Apache Kafka 并进行消息收发，您也可以直接参考消息队列 for Apache Kafka Demo 库中的 Demo 和说明。

前提条件

请确保已做好#unique_7。

添加 Maven 依赖

```
//消息队列 for Apache Kafka 服务端版本为 0.10.0.0，建议客户端版本为 0.10.2.2
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
```

```
    <version>0.10.2.2</version>
</dependency>
```

2. 使用 SDK

1. 准备配置文件 *kafka.properties*，可参考消息队列 for Apache Kafka Demo 库中的 Demo 和说明进行修改。

```
## 您在控制台获取的接入点
bootstrap.servers=xxx.xxx.xxx.xxx:9092

## 您在控制台创建的 Topic
topic=alikafka-topic-demo

## 您在控制台创建的 Consumer Group
group.id=CID-consumer-group-demo
```

2. 加载配置文件 `kafka.properties`。

```
public class JavaKafkaConfigurer {

    private static Properties properties;

    public synchronized static Properties getKafkaProperties() {
        if (null != properties) {
            return properties;
        }
        //获取配置文件 kafka.properties 的内容
        Properties kafkaProperties = new Properties();
        try {
            kafkaProperties.load(KafkaProducerDemo.class.getClassLo
ader().getResourceAsStream("kafka.properties"));
        } catch (Exception e) {
            //没加载到文件，程序要考虑退出
            e.printStackTrace();
        }
        properties = kafkaProperties;
        return kafkaProperties;
    }
}
```

3. 发送消息。

```
public class KafkaProducerDemo {

    public static void main(String args[]) {

        //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("bootstrap.servers"));
        //消息队列 for Apache Kafka 消息的序列化方式
```

```
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
 "org.apache.kafka.common.serialization.StringSerializer");
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
 "org.apache.kafka.common.serialization.StringSerializer");
        //请求的最长等待时间
        props.put(ProducerConfig.MAX_BLOCK_MS_CONFIG, 30 * 1000);

        //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，可构造多
个对象，但最好不要超过 5 个
        KafkaProducer<String, String> producer = new
 KafkaProducer<String, String>(props);

        //构造一个消息队列 for Apache Kafka 消息
        String topic = kafkaProperties.getProperty("topic"); //消息所
属的 Topic，请在控制台创建后，填写在这里
        String value = "this is the message's value"; //消息的内容

        ProducerRecord<String, String>  kafkaMessage =  new
 ProducerRecord<String, String>(topic, value);

        try {
            //发送消息，并获得一个 Future 对象
            Future<RecordMetadata> metadataFuture =
 producer.send(kafkaMessage);
            //同步获得 Future 对象的结果
            RecordMetadata recordMetadata = metadataFuture.get();
            System.out.println("Produce ok:" +
 recordMetadata.toString());
        } catch (Exception e) {
            //要考虑重试，参见常见问题：https://help.aliyun.com/
document_detail/124136.html
            System.out.println("error occurred");
            e.printStackTrace();
        }
    }
}
```

4. 消费消息。

```
public class KafkaConsumerDemo {

    public static void main(String args[]) {

        //加载 kafka.properties
        Properties kafkaProperties =  JavaKafkaConfigurer.getKafkaPr
operties();

        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, kafkaPrope
rties.getProperty("bootstrap.servers"));
        //默认值为 30000 ms，可根据自己业务场景调整此值，建议取值不要太小，防止
在超时时间内没有发送心跳导致消费者再均衡
        props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, 25000);
        //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 Consumer Group，请在控制台创建后填写
        //属于同一个 Consumer Group 的消费实例，会负载消费消息
        props.put(ConsumerConfig.GROUP_ID_CONFIG, kafkaProperties.
getProperty("group.id"));
        //构造消息对象，即生成一个消费实例
        KafkaConsumer<String, String> consumer = new org.apache.kafka.
clients.consumer.KafkaConsumer<String, String>(props);
        //设置  Consumer Group 订阅的 Topic，可订阅多个 Topic。如果
GROUP_ID_CONFIG 相同，那建议订阅的 Topic 设置也相同
        List<String> subscribedTopics =  new ArrayList<String>();
        //每个 Topic 需要先在控制台进行创建
        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 的值
                //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", record.partition(), record.offset()));
                }
            } catch (Exception e) {
                try {
                    Thread.sleep(1000);
                } catch (Throwable ignore) {

                }
                //参见常见问题：https://help.aliyun.com/document_detail/
124136.html
                e.printStackTrace();
            }
        }
    }
}
```

2 API 参考

2.1 API 版本说明

本文介绍消息队列 for Apache Kafka 所有的 API 版本发布信息。

详情请见下表。

API 版本	发布日期	说明
V1.0.3	2018 年 12 月 6 日	· 新增根据实例 ID 创建 Topic 的接口: CreateTopic · 新增根据实例 ID 创建 ConsumerGroup 的接口：CreateConsumerGroup
V1.0.1	2018 年 10 月 25 日	GetConsumerProgress 接口的返回值数据结构 `GetConsumerProgressResponse.ConsumerProgress.TopicListItem.OffsetListItem` 增加属性分区字段“partition”
V1.0.0	2018 年 10 月 25 日	消息队列 for Apache Kafka 第一版 API 提供以下查询类接口: · 根据 AccessKeyId/AccessKeySecret 获取的实例列表: GetInstanceList · 根据实例 ID 获取实例下的 Topic 列表：GetTopicList · 根据 Topic 获取 Topic 信息：GetTopicStatus · 根据实例 ID 获取消费 ID 列表: GetConsumerList · 根据 Consumer ID 获取消费进度信息: GetConsumerProgress

2.2 接入指南

消息队列 for Apache Kafka 的管控 API 提供获取实例、Topic 和 Consumer Group 信息的接口。

本文介绍消息队列 for Apache Kafka SDK 的获取、初始参数的设置、使用方法概述等。

获取 SDK

以 Java 应用为例，在配置文件中添加依赖的 SDK 信息：

```
        <dependency>
            <groupId>com.aliyun</groupId>
            <artifactId>aliyun-java-sdk-alikafka</artifactId>
            <version>1.0.5</version>
        </dependency>
        <dependency>
            <groupId>com.aliyun</groupId>
            <artifactId>aliyun-java-sdk-core</artifactId>
            <version>4.3.9</version>
        </dependency>
```

设置公共参数

构建并启动客户端时需要设置一系列参数信息，具体示例如下：

```
public static void main(String[] args) {
    //构建 Client
    IAcsClient iAcsClient = buildAcsClient();
 }
 private static IAcsClient buildAcsClient() {
        //鉴权使用的 AccessKeyId，由阿里云官网控制台获取
        String accessKey = "XXXXXX";
        //鉴权使用的 AccessKeySecret，由阿里云官网控制台获取
        String secretKey = "XXXXXX";

        //产品 Code，消息队列 for Apache Kafka 产品常量值为 “alikafka”
        String productName = "alikafka";

        //API 的网关所在地域，目前支持的有 cn-beijing 和 cn-hangzhou 等
        String regionId = "cn-beijing";
        //接入点名称同 regionId 一致
        String endPointName = "cn-beijing";
        //对应 endPoint 的域名
        String domain = "alikafka.cn-beijing.aliyuncs.com";

        try {
            DefaultProfile.addEndpoint(endPointName, regionId,
 productName, domain);
        } catch (ClientException e) {
            //log error
        }
        //构造 Client
        IClientProfile profile = DefaultProfile.getProfile(regionId,
 accessKey, secretKey);
        return new DefaultAcsClient(profile);
}
```

接入点列表

地域名称	RegionId	Domain
华东1（杭州）	cn-hangzhou	alikafka.cn-hangzhou.aliyuncs.com
华东2（上海）	cn-shanghai	alikafka.cn-shanghai.aliyuncs.com
华北1（青岛）	cn-qingdao	alikafka.cn-qingdao.aliyuncs.com
华北2（北京）	cn-beijing	alikafka.cn-beijing.aliyuncs.com
华北3（张家口）	cn-zhangjiakou	alikafka.cn-zhangjiakou.aliyuncs.com
华北5（呼和浩特）	cn-huhehaote	alikafka.cn-huhehaote.aliyuncs.com
华南1（深圳）	cn-shenzhen	alikafka.cn-shenzhen.aliyuncs.com
中国（香港）	cn-hongkong	alikafka.cn-hongkong.aliyuncs.com
新加坡	ap-southeast-1	alikafka.ap-southeast-1.aliyuncs.com

接口概览

以下是消息队列 for Apache Kafka 目前支持的管控 API 的概览说明。

- 实例管理接口
  - #unique_11：查询实例信息列表
- Topic 管理接口
  - #unique_12：创建 Topic
  - #unique_13：获取 Topic 列表
  - #unique_14：查询 Topic 消息收发信息

 说明：

更多接口将陆续开放。

- Consumer Group 接口
  - #unique_15：创建 Consumer Group
  - #unique_16：查询 Consumer Group 列表
  - #unique_17：查询 Consumer Group 消费进度

> 说明:
> 更多接口将陆续开放。

使用限制

单用户单接口的请求频率的限制为 3 QPS。更新和创建类接口的使用限制，请参见相应接口文档中的使用限制内容。

2.3 实例管理接口

2.3.1 GetInstanceList

调用 GetInstanceList 获取您在某地域下所购买的实例的信息。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	GetInstanceList	要执行的动作。取值：GetInstanceList
RegionId	String	是	cn-hangzhou	实例所属的地域 ID。

返回数据

名称	类型	示例值	描述
Code	Integer	200	返回码，返回 “200” 代表成功。
InstanceList			实例列表信息。
CreateTime	Long	1566215995000	创建时间。

名称	类型	示例值	描述
DeployType	Integer	4	部署类型。
EndPoint	String	192.168.0.212:9092,192.168.0.210:9092,192.168.0.211:****	默认接入点。
ExpiredTime	Long	1568908800000	过期时间。
InstanceId	String	alikafka_pre-cn-mp919o4v****	实例 ID。
Name	String	alikafka_pre-cn-mp919o4v****	实例名称。
RegionId	String	cn-hangzhou	实例所属的地域 ID。
ServiceStatus	Integer	5	服务状态。
SslEndPoint	String	47.111.110.11:9093,121.40.96.141:9093,47.111.118.133:****	SSL 接入点。
UpgradeServiceDetailInfo			升级服务详细信息。
Current2OpenSourceVersion	String	0.10	开源版本。
VSwitchId	String	vsw-bp13rg6bcpkxofr78****	VSwitch ID。
VpcId	String	vpc-bp1l6hrlykj3405r7****	VPC ID。
Message	String	operation success.	返回信息。
RequestId	String	ABA4A7FD-E10F-45C7-9774-A5236015****	请求 ID。
Success	Boolean	true	调用是否成功。

示例

请求示例

```
http(s)://[Endpoint]/?Action=GetInstanceList
&RegionId=cn-hangzhou
&<公共请求参数>
```

正常返回示例

XML 格式

```
<GetInstanceListResponse>
      <Message>operation success.</Message>
      <RequestId>0220FACD-4D57-4F46-BA77-AD333498****</RequestId>
      <Success>true</Success>
      <Code>200</Code>
      <InstanceList>
            <InstanceVO>
                  <Name>alikafka_pre-cn-mp919o4v****</Name>
                  <DeployType>4</DeployType>
                  <CreateTime>1566215995000</CreateTime>
                  <RegionId>cn-hangzhou</RegionId>
                  <InstanceId>alikafka_pre-cn-mp919o4v****</InstanceId
>
                  <SslEndPoint>47.111.110.11:9093,121.40.96.141:9093,
47.111.118.133:****</SslEndPoint>
                  <EndPoint>192.168.0.212:9092,192.168.0.210:9092,192.
168.0.211:****</EndPoint>
                  <ExpiredTime>1568908800000</ExpiredTime>
                  <VSwitchId>vsw-bp13rg6bcpkxofr78****</VSwitchId>
                  <VpcId>vpc-bp1l6hrlykj3405r7****</VpcId>
                  <UpgradeServiceDetailInfo>
                        <Current2OpenSourceVersion>0.10</Current2Op
enSourceVersion>
                  </UpgradeServiceDetailInfo>
                  <ServiceStatus>5</ServiceStatus>
            </InstanceVO>
      </InstanceList>
</GetInstanceListResponse>
```

JSON 格式

```
{
 "Message":"operation success.",
 "RequestId":"0220FACD-4D57-4F46-BA77-AD333498****",
 "Success":true,
 "Code":200,
 "InstanceList":{
  "InstanceVO":[
   {
    "Name":"alikafka_pre-cn-mp919o4v****",
    "DeployType":4,
    "InstanceId":"alikafka_pre-cn-mp919o4v****",
    "RegionId":"cn-hangzhou",
    "CreateTime":1566215995000,
    "SslEndPoint":"47.111.110.11:9093,121.40.96.141:9093,47.111.118.
133:****",
```

```
    "EndPoint":"192.168.0.212:9092,192.168.0.210:9092,192.168.0.211
:****",
    "VSwitchId":"vsw-bp13rg6bcpkxofr78****",
    "ExpiredTime":1568908800000,
    "VpcId":"vpc-bp1l6hrlykj3405r7****",
    "ServiceStatus":5,
    "UpgradeServiceDetailInfo":{
     "Current2OpenSourceVersion":"0.10"
    }
   }
  ]
 }
}
```

错误码

访问错误中心查看更多错误码。

2.4 Topic 管理接口

2.4.1 CreateTopic

调用 CreateTopic 创建 Topic。

调用该接口创建 Topic 时，请注意：

- 单用户请求频率限制为 1 QPS。
- 每个实例下最多可创建的 Topic 数量与您所购买的实例版本相关，详情请参见计费说明。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	CreateTopic	系统规定参数。 取值：CreateTopic
InstanceId	String	是	alikafka_pre-cn-mp919o4v****	实例 ID。 可调用 GetInstanceList 获取。
RegionId	String	是	cn-hangzhou	实例所属的地域 ID。

名称	类型	是否必选	示例值	描述
Remark	String	是	alikafka_topic_test	Topic 的备注。 限制 64 个字符。
Topic	String	是	alikafka_topic_test	Topic 的名称。 · 只能包含字母、数字、下划线（_）和短横线（-）。 · 长度为 3-64 个字符，多于 64 个字符将被自动截取。 · 一旦创建后不能再修改。

返回数据

名称	类型	示例值	描述
Code	Integer	200	返回码。 返回 200 为成功。
Message	String	operation success	返回信息。
RequestId	String	9C0F207C-77A6-43E5-991C-9D98510A****	请求的唯一标识 ID。
Success	Boolean	true	调用是否成功。

示例

请求示例

```
http(s)://[Endpoint]/?Action=CreateTopic
&InstanceId=alikafka_pre-cn-mp919o4v****
&RegionId=cn-hangzhou
&Remark=alikafka_topic_test
&Topic=alikafka_topic_test
&<公共请求参数>
```

正常返回示例

XML 格式

```
<CreateTopicResponse>
   <Message>operation success</Message>
   <RequestId>9C0F207C-77A6-43E5-991C-9D98510A****</RequestId>
```

```
    <Success>true</Success>
    <Code>200</Code>
</CreateTopicResponse>
```

JSON 格式

```
{
 "Message":"operation success",
 "RequestId":"9C0F207C-77A6-43E5-991C-9D98510A****",
 "Success":true,
 "Code":200
}
```

错误码

访问错误中心查看更多错误码。

2.4.2 GetTopicList

调用 GetTopicList 获取 Topic 的列表。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	GetTopicList	要执行的动作。取值：GetTopicList
InstanceId	String	是	alikafka_pre-cn-0pp1954n2003	实例 ID。
RegionId	String	是	cn-hangzhou	地域 ID。

返回数据

名称	类型	示例值	描述
Code	Integer	200	返回码。返回 200 为成功。
CurrentPage	Integer	1	当前页面。
Message	String	operation success.	返回信息。

名称	类型	示例值	描述
PageSize	Integer	10000	页面大小。
RequestId	String	82BD585C-17A1-486E-B3E8-AABCE8EE****	请求 ID。
Success	Boolean	true	调用是否成功。
TopicList			Topic 列表。
CompactTopic	Boolean	false	存储引擎是否为 Compact。
CreateTime	Long	1566804394000	创建时间。
InstanceId	String	alikafka_pre-cn-0pp1954n****	实例 ID。
LocalTopic	Boolean	false	Topic 的存储引擎是否为 Local。
PartitionNum	Integer	12	分区数。
RegionId	String	cn-hangzhou	地域 ID。
Remark	String	kafka_test_topic	Topic 备注。
Status	Integer	0	状态码。 · 0：服务中。 · 1：冻结。 · 2：暂停。
StatusName	String	服务中	状态描述。
Topic	String	poptest	Topic 名称。
Total	Integer	12	Topic 总数。

示例

请求示例

```
http(s)://[Endpoint]/?Action=GetTopicList
```

```
&InstanceId=alikafka_pre-cn-0pp1954n2003
&<公共请求参数>
```

正常返回示例

XML 格式

```
<TopicListResponse>
   <TopicList>
      <TopicVO>
         <PartitionNum>12</PartitionNum>
         <Status>0</Status>
         <CompactTopic>false</CompactTopic>
         <RegionId>cn-hangzhou</RegionId>
         <InstanceId>alikafka_pre-cn-0pp1954n****</InstanceId>
         <CreateTime>1567146954000</CreateTime>
         <Topic>poptest</Topic>
         <StatusName>服务中</StatusName>
         <LocalTopic>false</LocalTopic>
         <Remark>test</Remark>
      </TopicVO>
      <TopicVO>
         <PartitionNum>12</PartitionNum>
         <Status>0</Status>
         <CompactTopic>false</CompactTopic>
         <RegionId>cn-hangzhou</RegionId>
         <InstanceId>alikafka_pre-cn-0pp1954n****</InstanceId>
         <CreateTime>1566804394000</CreateTime>
         <Topic>kafka_test_topic</Topic>
         <StatusName>服务中</StatusName>
         <LocalTopic>false</LocalTopic>
         <Remark>kafka_test_topic1111111111111111111122222222222222</
Remark>
      </TopicVO>
      <TopicVO>
         <PartitionNum>12</PartitionNum>
         <Status>0</Status>
         <CompactTopic>false</CompactTopic>
         <RegionId>cn-hangzhou</RegionId>
         <InstanceId>alikafka_pre-cn-0pp1954n****</InstanceId>
         <CreateTime>1566804290000</CreateTime>
         <Topic>topic_test</Topic>
         <StatusName>服务中</StatusName>
         <LocalTopic>false</LocalTopic>
         <Remark>topic_test</Remark>
      </TopicVO>
      <TopicVO>
         <PartitionNum>6</PartitionNum>
         <Status>0</Status>
         <CompactTopic>false</CompactTopic>
         <RegionId>cn-hangzhou</RegionId>
         <InstanceId>alikafka_pre-cn-0pp1954n****</InstanceId>
         <CreateTime>1565589319000</CreateTime>
         <Topic>testkafka</Topic>
         <StatusName>服务中</StatusName>
         <LocalTopic>false</LocalTopic>
         <Remark>test-data-migration</Remark>
      </TopicVO>
      <TopicVO>
         <PartitionNum>45</PartitionNum>
         <Status>0</Status>
         <CompactTopic>false</CompactTopic>
```

```
            <RegionId>cn-hangzhou</RegionId>
            <InstanceId>alikafka_pre-cn-0pp1954n****</InstanceId>
            <CreateTime>1565331243000</CreateTime>
            <Topic>normal_topic_4fec2e97daa6f089f1a48b36b6bea34e_6</Topic
>
            <StatusName>服务中</StatusName>
            <LocalTopic>false</LocalTopic>
            <Remark>topic1</Remark>
         </TopicVO>
         <TopicVO>
            <PartitionNum>24</PartitionNum>
            <Status>0</Status>
            <CompactTopic>false</CompactTopic>
            <RegionId>cn-hangzhou</RegionId>
            <InstanceId>alikafka_pre-cn-0pp1954n****</InstanceId>
            <CreateTime>1565331241000</CreateTime>
            <Topic>normal_topic_4fec2e97daa6f089f1a48b36b6bea34e_5</Topic
>
            <StatusName>服务中</StatusName>
            <LocalTopic>false</LocalTopic>
            <Remark>topic1</Remark>
         </TopicVO>
      </TopicList>
      <Message>operation success.</Message>
      <PageSize>10000</PageSize>
      <RequestId>BF67619B-17C5-4FE6-BFB4-83B8C9F7****</RequestId>
      <Success>true</Success>
      <CurrentPage>1</CurrentPage>
      <Code>200</Code>
      <Total>6</Total>
</TopicListResponse>
```

JSON 格式

```
{
 "TopicList":{
  "TopicVO":[
   {
    "PartitionNum":12,
    "Status":0,
    "CompactTopic":false,
    "CreateTime":1567146954000,
    "InstanceId":"alikafka_pre-cn-0pp1954n****",
    "RegionId":"cn-hangzhou",
    "Topic":"poptest",
    "StatusName":"服务中",
    "LocalTopic":false,
    "Remark":"test"
   },
   {
    "PartitionNum":12,
    "Status":0,
    "CompactTopic":false,
    "CreateTime":1566804394000,
    "InstanceId":"alikafka_pre-cn-0pp1954n****",
    "RegionId":"cn-hangzhou",
    "Topic":"kafka_test_topic",
    "StatusName":"服务中",
    "LocalTopic":false,
    "Remark":"kafka_test_topic1111111111111111111122222222222222"
   },
   {
    "PartitionNum":12,
```

```
     "Status":0,
     "CompactTopic":false,
     "CreateTime":1566804290000,
     "InstanceId":"alikafka_pre-cn-0pp1954n****",
     "RegionId":"cn-hangzhou",
     "Topic":"topic_test",
     "StatusName":"服务中",
     "LocalTopic":false,
     "Remark":"topic_test"
    },
    {
     "PartitionNum":6,
     "Status":0,
     "CompactTopic":false,
     "CreateTime":1565589319000,
     "InstanceId":"alikafka_pre-cn-0pp1954n****",
     "RegionId":"cn-hangzhou",
     "Topic":"testkafka",
     "StatusName":"服务中",
     "LocalTopic":false,
     "Remark":"test-data-migration"
    },
    {
     "PartitionNum":45,
     "Status":0,
     "CompactTopic":false,
     "CreateTime":1565331243000,
     "InstanceId":"alikafka_pre-cn-0pp1954n****",
     "RegionId":"cn-hangzhou",
     "Topic":"normal_topic_4fec2e97daa6f089f1a48b36b6bea34e_6",
     "StatusName":"服务中",
     "LocalTopic":false,
     "Remark":"topic1"
    },
    {
     "PartitionNum":24,
     "Status":0,
     "CompactTopic":false,
     "CreateTime":1565331241000,
     "InstanceId":"alikafka_pre-cn-0pp1954n****",
     "RegionId":"cn-hangzhou",
     "Topic":"normal_topic_4fec2e97daa6f089f1a48b36b6bea34e_5",
     "StatusName":"服务中",
     "LocalTopic":false,
     "Remark":"topic1"
    }
   ]
  },
  "Message":"operation success.",
  "PageSize":10000,
  "RequestId":"BF67619B-17C5-4FE6-BFB4-83B8C9F7****",
  "CurrentPage":1,
  "Success":true,
  "Code":200,
  "Total":6
}
```


错误码

访问错误中心查看更多错误码。

2.4.3 GetTopicStatus

调用 GetTopicStatus 获取 Topic 的消息收发数据。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	GetTopicStatus	要执行的动作。取值：GetTopicStatus
InstanceId	String	是	alikafka_pre-cn-v0h15tjmo003	实例 ID。 可调用 GetInstanceList获取。
Topic	String	是	normal_topic_9d034262835916103455551be06cc2dc_6	Topic 名称。 可调用 GetTopicList 获取。

返回数据

名称	类型	示例值	描述
Code	Integer	200	状态吗。返回 200 代表成功。
Message	String	operation success.	返回信息。
RequestId	String	E475C7E2-8C35-46EF-BE7D-5D2A9F5D****	请求 ID。
Success	Boolean	true	调用是否成功。
TopicStatus			Topic 状态。
LastTimeStamp	Long	1566470063575	最后一条被消费的消息的产生时间。
OffsetTable			偏移列表。

名称	类型	示例值	描述
LastUpdate Timestamp	Long	1566470063547	最后被消费的消息的产生时间。
MaxOffset	Long	76	消息最大位点。
MinOffset	Long	0	消息最小位点。
Partition	Integer	0	分区 ID。
Topic	String	testkafka	Topic 名称。
TotalCount	Long	423	消息总数。

示例

请求示例

```
http(s)://[Endpoint]/?Action=GetTopicStatus
&InstanceId=alikafka_pre-cn-v0h15tjmo003
&Topic=normal_topic_9d0342628359161034555551be06cc2dc_6
&<公共请求参数>
```

正常返回示例

`XML` **格式**

```
<GetTopicStatusResponse>
   <Message>operation success.</Message>
   <RequestId>E475C7E2-8C35-46EF-BE7D-5D2A9F5D****</RequestId>
   <TopicStatus>
      <TotalCount>423</TotalCount>
      <LastTimeStamp>1566470063575</LastTimeStamp>
      <OffsetTable>
         <OffsetTable>
            <Partition>0</Partition>
            <LastUpdateTimestamp>1566470063547</LastUpdateTimestamp>
            <Topic>testkafka</Topic>
            <MaxOffset>76</MaxOffset>
            <MinOffset>0</MinOffset>
         </OffsetTable>
         <OffsetTable>
            <Partition>1</Partition>
            <LastUpdateTimestamp>1566470063575</LastUpdateTimestamp>
            <Topic>testkafka</Topic>
            <MaxOffset>69</MaxOffset>
            <MinOffset>0</MinOffset>
         </OffsetTable>
         <OffsetTable>
            <Partition>2</Partition>
            <LastUpdateTimestamp>1566470063554</LastUpdateTimestamp>
            <Topic>testkafka</Topic>
```

```
            <MaxOffset>70</MaxOffset>
            <MinOffset>0</MinOffset>
         </OffsetTable>
         <OffsetTable>
            <Partition>3</Partition>
            <LastUpdateTimestamp>1566470063538</LastUpdateTimestamp>
            <Topic>testkafka</Topic>
            <MaxOffset>67</MaxOffset>
            <MinOffset>0</MinOffset>
         </OffsetTable>
         <OffsetTable>
            <Partition>4</Partition>
            <LastUpdateTimestamp>1566470063568</LastUpdateTimestamp>
            <Topic>testkafka</Topic>
            <MaxOffset>67</MaxOffset>
            <MinOffset>0</MinOffset>
         </OffsetTable>
         <OffsetTable>
            <Partition>5</Partition>
            <LastUpdateTimestamp>1566470063561</LastUpdateTimestamp>
            <Topic>testkafka</Topic>
            <MaxOffset>74</MaxOffset>
            <MinOffset>0</MinOffset>
         </OffsetTable>
      </OffsetTable>
   </TopicStatus>
   <Success>true</Success>
   <Code>200</Code>
</GetTopicStatusResponse>
```

JSON 格式

```
{
 "Message":"operation success.",
 "RequestId":"E475C7E2-8C35-46EF-BE7D-5D2A9F5D****",
 "TopicStatus":{
  "TotalCount":423,
  "OffsetTable":{
   "OffsetTable":[
    {
     "Partition":0,
     "LastUpdateTimestamp":1566470063547,
     "Topic":"testkafka",
     "MaxOffset":76,
     "MinOffset":0
    },
    {
     "Partition":1,
     "LastUpdateTimestamp":1566470063575,
     "Topic":"testkafka",
     "MaxOffset":69,
     "MinOffset":0
    },
    {
     "Partition":2,
     "LastUpdateTimestamp":1566470063554,
     "Topic":"testkafka",
     "MaxOffset":70,
     "MinOffset":0
    },
    {
     "Partition":3,
     "LastUpdateTimestamp":1566470063538,
```

```
      "Topic":"testkafka",
      "MaxOffset":67,
      "MinOffset":0
     },
     {
      "Partition":4,
      "LastUpdateTimestamp":1566470063568,
      "Topic":"testkafka",
      "MaxOffset":67,
      "MinOffset":0
     },
     {
      "Partition":5,
      "LastUpdateTimestamp":1566470063561,
      "Topic":"testkafka",
      "MaxOffset":74,
      "MinOffset":0
     }
    ]
   },
   "LastTimeStamp":1566470063575
  },
  "Success":true,
  "Code":200
}
```


错误码

访问错误中心查看更多错误码。

2.4.4 DeleteTopic

调用 DeleteTopic 删除 Topic。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DeleteTopic	要执行的动作。取值： DeleteTopic
InstanceId	String	是	alikafka_post-cn-mp91a44k3002	实例 ID。 可调用 GetInstanceList 获取。
RegionId	String	是	cn-hangzhou	实例所属的地域 ID。

名称	类型	是否必选	示例值	描述
Topic	String	是	Kafkatest	Topic 的名称。 · 只能包含字母、数字、下划线（_）和短横线（-）。 · 长度为 3-64 个字符，多于 64 个字符将被自动截取。 · 一旦创建后不能再修改。

返回数据

示例

请求示例

```
http(s)://[Endpoint]/?Action=DeleteTopic
&InstanceId=alikafka_post-cn-mp91a44k3002
&RegionId=cn-hangzhou
&Topic=Kafkatest
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<DeleteTopicResponse>
    <Message>operation success</Message>
    <RequestId>9B618B3F-9506-4661-A211-D00C4556F928</RequestId>
    <Success>true</Success>
    <Code>200</Code>
</DeleteTopicResponse>
```

`JSON` 格式

```
{
 "Message":"operation success",
 "RequestId":"9B618B3F-9506-4661-A211-D00C4556F928",
 "Success":true,
 "Code":200
}
```

错误码

HttpCode	错误码	错误信息	描述
500	InternalError	An internal error occurred; please try again later.	系统内部错误，请稍后重试

访问错误中心查看更多错误码。

2.5 Consumer Group 管理接口

2.5.1 CreateConsumerGroup

调用 CreateConsumerGroup 创建 Consumer Group。

您在调用 CreateConsumerGroup 创建 Consumer Group 时，请注意：单用户请求频率限制为 1 QPS。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	CreateConsumerGroup	要执行的动作。取值：CreateConsumerGroup
ConsumerId	String	是	consumer_group_test	Consumer Group 名称。 · 只能包含字母、数字、短横线（-）、下划线（_）。 · 长度限制在 3-64 个字符，多于 64 个字符将被自动截取。 · 一旦创建后不能再修改。
InstanceId	String	是	alikafka_pre-cn-0pp1954n****	实例 ID。可调用 GetInstanceList 获取。
RegionId	String	是	cn-hangzhou	地域 ID。

返回数据

名称	类型	示例值	描述
Code	Integer	200	返回码。返回 200 为成功。
Message	String	operation success	返回信息。

名称	类型	示例值	描述
RequestId	String	B191CC4D-B067-4508-987A-ACDA8D89****	请求 ID。
Success	Boolean	true	调用是否成功。

示例

请求示例

```
http(s)://[Endpoint]/?Action=CreateConsumerGroup
&ConsumerId=consumer_group_test
&InstanceId=alikafka_pre-cn-0pp1954n****
&RegionId=cn-hangzhou
&<公共请求参数>
```

正常返回示例

`XML` **格式**

```
<CreateConsumerGroupResponse>
      <Message>operation success</Message>
      <RequestId>B191CC4D-B067-4508-987A-ACDA8D89****</RequestId>
      <Success>true</Success>
      <Code>200</Code>
</CreateConsumerGroupResponse>
```

`JSON` **格式**

```
{
 "Message":"operation success",
 "RequestId":"B191CC4D-B067-4508-987A-ACDA8D89****",
 "Success":true,
 "Code":200
}
```

错误码

访问错误中心查看更多错误码。

2.5.2 GetConsumerList

调用 GetConsumerList 获取 Consumer Group 列表。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	GetConsumerList	要执行的动作。取值：GetConsumerList
InstanceId	String	是	alikafka_post-cn-v0h18sav****	实例 ID。
RegionId	String	是	cn-hangzhou	地域 ID。

返回数据

名称	类型	示例值	描述
Code	Integer	200	返回码。返回 200 代表成功。
ConsumerList			Consumer Group 列表。
ConsumerId	String	CID_c34a6f44915f80d70cb42c4b14ee40c3_4	Consumer Group 名称。
InstanceId	String	alikafka_post-cn-v0h18sav0001	实例 ID。
RegionId	String	cn-hangzhou	地域 ID。
Message	String	operation success.	返回信息。
RequestId	String	808F042B-CB9A-4FBC-9009-00E7DDB6****	请求 ID。
Success	Boolean	true	调用是否成功。

示例

请求示例

```
http(s)://[Endpoint]/?Action=GetConsumerList
&InstanceId=alikafka_post-cn-v0h18sav****
&RegionId=cn-hangzhou
```

```
&<公共请求参数>
```

正常返回示例

XML 格式

```
<GetConsumerListResponse>
      <ConsumerList>
            <ConsumerVO>
                  <ConsumerId>CID_c34a6f44915f80d70cb42c4b14ee40c3_4</
ConsumerId>
                  <InstanceId>alikafka_post-cn-v0h18sav****</
InstanceId>
                  <RegionId>cn-hangzhou</RegionId>
            </ConsumerVO>
            <ConsumerVO>
                  <ConsumerId>CID_c34a6f44915f80d70cb42c4b14ee40c3_3</
ConsumerId>
                  <InstanceId>alikafka_post-cn-v0h18sav****</
InstanceId>
                  <RegionId>cn-hangzhou</RegionId>
            </ConsumerVO>
      </ConsumerList>
      <Message>operation success.</Message>
      <RequestId>808F042B-CB9A-4FBC-9009-00E7DDB6****</RequestId>
      <Success>true</Success>
      <Code>200</Code>
</GetConsumerListResponse>
```

JSON 格式

```
{
 "ConsumerList":{
  "ConsumerVO":[
   {
    "ConsumerId":"CID_c34a6f44915f80d70cb42c4b14ee40c3_4",
    "RegionId":"cn-hangzhou",
    "InstanceId":"alikafka_post-cn-v0h18sav****"
   },
   {
    "ConsumerId":"CID_c34a6f44915f80d70cb42c4b14ee40c3_3",
    "RegionId":"cn-hangzhou",
    "InstanceId":"alikafka_post-cn-v0h18sav****"
   }
  ]
 },
 "Message":"operation success.",
 "RequestId":"808F042B-CB9A-4FBC-9009-00E7DDB6****",
 "Success":true,
 "Code":200
}
```

错误码

访问错误中心查看更多错误码。

2.5.3 GetConsumerProgress

调用 GetConsumerProgress 查询 Consumer Group 的消费状态。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	GetConsumerProgress	要执行的动作。取值：GetConsumerProgress
ConsumerId	String	是	kafka-test	Consumer Group ID。
InstanceId	String	是	alikafka_pre-cn-mp919o4v****	实例 ID。
RegionId	String	是	cn-hangzhou	地域 ID。

返回数据

名称	类型	示例值	描述
Code	Integer	200	返回码。返回 200 代表成功。
ConsumerProgress			消费状态。
LastTimestamp	Long	1566874931671	此 Consumer Group 最后被消费的消息的产生时间。
TopicList			此 Consumer Group 对应的每个 Topic 的消费进度列表。
LastTimestamp	Long	1566874931649	该分区最后被消费的消息的产生时间。
OffsetList			偏移列表。
BrokerOffset	Long	9	最大位点。

名称	类型	示例值	描述
ConsumerOffset	Long	9	消费位点。
LastTimestamp	Long	1566874931649	该分区最后被消费的消息的产生时间。
Partition	Integer	0	分区 ID。
Topic	String	kafka-test	Topic 名称。
TotalDiff	Long	0	该 Topic 的未消费消息总量，即堆积量。
TotalDiff	Long	0	此 Consumer Group 未消费的消息总量，即堆积量。
Message	String	operation success.	返回信息。
RequestId	String	252820E1-A2E6-45F2-B4C9-1056B8CE****	请求 ID。
Success	Boolean	true	调用是否成功。

示例

请求示例

```
http(s)://[Endpoint]/?Action=GetConsumerProgress
&ConsumerId=kafka-test
&InstanceId=alikafka_pre-cn-mp919o4v****
&RegionId=cn-hangzhou
&<公共请求参数>
```

正常返回示例

`XML` 格式

```
<GetConsumerProgressResponse>
      <Message>operation success.</Message>
      <RequestId>252820E1-A2E6-45F2-B4C9-1056B8CE****</RequestId>
      <ConsumerProgress>
            <TotalDiff>0</TotalDiff>
            <LastTimestamp>1566874931671</LastTimestamp>
            <TopicList>
                  <TopicList>
                        <TotalDiff>0</TotalDiff>
```

```
                        <OffsetList>
                              <OffsetList>
                                    <Partition>0</Partition>
                                    <LastTimestamp>1566874931649</
LastTimestamp>
                                    <BrokerOffset>9</BrokerOffset>
                                    <ConsumerOffset>9</ConsumerOffset>
                              </OffsetList>
                              <OffsetList>
                                    <Partition>1</Partition>
                                    <LastTimestamp>1566874931605</
LastTimestamp>
                                    <BrokerOffset>9</BrokerOffset>
                                    <ConsumerOffset>9</ConsumerOffset>
                              </OffsetList>
                              <OffsetList>
                                    <Partition>2</Partition>
                                    <LastTimestamp>1566874931561</
LastTimestamp>
                                    <BrokerOffset>10</BrokerOffset>
                                    <ConsumerOffset>10</ConsumerOffset
>
                              </OffsetList>
                              <OffsetList>
                                    <Partition>3</Partition>
                                    <LastTimestamp>1566874931628</
LastTimestamp>
                                    <BrokerOffset>8</BrokerOffset>
                                    <ConsumerOffset>8</ConsumerOffset>
                              </OffsetList>
                              <OffsetList>
                                    <Partition>4</Partition>
                                    <LastTimestamp>1566874931579</
LastTimestamp>
                                    <BrokerOffset>8</BrokerOffset>
                                    <ConsumerOffset>8</ConsumerOffset>
                              </OffsetList>
                              <OffsetList>
                                    <Partition>5</Partition>
                                    <LastTimestamp>1566874931570</
LastTimestamp>
                                    <BrokerOffset>10</BrokerOffset>
                                    <ConsumerOffset>10</ConsumerOffset
>
                              </OffsetList>
                              <OffsetList>
                                    <Partition>6</Partition>
                                    <LastTimestamp>1566874931639</
LastTimestamp>
                                    <BrokerOffset>9</BrokerOffset>
                                    <ConsumerOffset>9</ConsumerOffset>
                              </OffsetList>
                              <OffsetList>
                                    <Partition>7</Partition>
                                    <LastTimestamp>1566874931586</
LastTimestamp>
                                    <BrokerOffset>8</BrokerOffset>
                                    <ConsumerOffset>8</ConsumerOffset>
                              </OffsetList>
                              <OffsetList>
                                    <Partition>8</Partition>
                                    <LastTimestamp>1566874931661</
LastTimestamp>
                                    <BrokerOffset>9</BrokerOffset>
```

```
                                    <ConsumerOffset>9</ConsumerOffset>
                                </OffsetList>
                                <OffsetList>
                                    <Partition>9</Partition>
                                    <LastTimestamp>1566874931616</
LastTimestamp>
                                    <BrokerOffset>8</BrokerOffset>
                                    <ConsumerOffset>8</ConsumerOffset>
                                </OffsetList>
                                <OffsetList>
                                    <Partition>10</Partition>
                                    <LastTimestamp>1566874931596</
LastTimestamp>
                                    <BrokerOffset>8</BrokerOffset>
                                    <ConsumerOffset>8</ConsumerOffset>
                                </OffsetList>
                                <OffsetList>
                                    <Partition>11</Partition>
                                    <LastTimestamp>1566874931671</
LastTimestamp>
                                    <BrokerOffset>9</BrokerOffset>
                                    <ConsumerOffset>9</ConsumerOffset>
                                </OffsetList>
                            </OffsetList>
                            <LastTimestamp>1566874931671</LastTimestamp>
                            <Topic>kafka-test</Topic>
                        </TopicList>
                    </TopicList>
            </ConsumerProgress>
            <Success>true</Success>
            <Code>200</Code>
</GetConsumerProgressResponse>
```

JSON 格式

```
{
 "Message":"operation success.",
 "RequestId":"252820E1-A2E6-45F2-B4C9-1056B8CE****",
 "ConsumerProgress":{
  "TotalDiff":0,
  "LastTimestamp":1566874931671,
  "TopicList":{
   "TopicList":[
    {
     "TotalDiff":0,
     "LastTimestamp":1566874931671,
     "OffsetList":{
      "OffsetList":[
       {
        "Partition":0,
        "LastTimestamp":1566874931649,
        "BrokerOffset":9,
        "ConsumerOffset":9
       },
       {
        "Partition":1,
        "LastTimestamp":1566874931605,
        "BrokerOffset":9,
        "ConsumerOffset":9
       },
       {
        "Partition":2,
        "LastTimestamp":1566874931561,
```

```
        "BrokerOffset":10,
        "ConsumerOffset":10
       },
       {
        "Partition":3,
        "LastTimestamp":1566874931628,
        "BrokerOffset":8,
        "ConsumerOffset":8
       },
       {
        "Partition":4,
        "LastTimestamp":1566874931579,
        "BrokerOffset":8,
        "ConsumerOffset":8
       },
       {
        "Partition":5,
        "LastTimestamp":1566874931570,
        "BrokerOffset":10,
        "ConsumerOffset":10
       },
       {
        "Partition":6,
        "LastTimestamp":1566874931639,
        "BrokerOffset":9,
        "ConsumerOffset":9
       },
       {
        "Partition":7,
        "LastTimestamp":1566874931586,
        "BrokerOffset":8,
        "ConsumerOffset":8
       },
       {
        "Partition":8,
        "LastTimestamp":1566874931661,
        "BrokerOffset":9,
        "ConsumerOffset":9
       },
       {
        "Partition":9,
        "LastTimestamp":1566874931616,
        "BrokerOffset":8,
        "ConsumerOffset":8
       },
       {
        "Partition":10,
        "LastTimestamp":1566874931596,
        "BrokerOffset":8,
        "ConsumerOffset":8
       },
       {
        "Partition":11,
        "LastTimestamp":1566874931671,
        "BrokerOffset":9,
        "ConsumerOffset":9
       }
      ]
     },
     "Topic":"kafka-test"
    }
   ]
  }
 },
```

```
  "Success":true,
  "Code":200
}
```

错误码

访问错误中心查看更多错误码。

2.5.4 DeleteConsumerGroup

调用 DeleteConsumerGroup 删除 ConsumerGroup。

调试

您可以在OpenAPI Explorer中直接运行该接口，免去您计算签名的困扰。运行成功后，OpenAPI Explorer可以自动生成SDK代码示例。

请求参数

名称	类型	是否必选	示例值	描述
Action	String	是	DeleteConsumerGroup	要执行的动作。取值： DeleteConsumerGroup
ConsumerId	String	是	testconsumer	Consumer Group 名称。 · 只能包含字母、数字、短横线（-）、下划线（_）。 · 长度限制在 3-64 个字符，多于 64 个字符将被自动截取。 · 一旦创建后不能再修改。
InstanceId	String	是	alikafka_post-cn-mp91a44k****	实例 ID。 可调用 GetInstanceList 获取。
RegionId	String	是	cn-hangzhou	地域 ID。

返回数据

名称	类型	示例值	描述
Code	Integer	200	返回码。返回 200 代表成功。
Message	String	operation success	返回信息。

名称	类型	示例值	描述
RequestId	String	1AA2A2AD-2727-4573-B1C7-A0388BCD****	请求 ID。
Success	Boolean	true	调用是否成功。

示例

请求示例

```
http(s)://[Endpoint]/?Action=DeleteConsumerGroup
&ConsumerId=testconsumer
&InstanceId=alikafka_post-cn-mp91a44k****
&RegionId=cn-hangzhou
&<公共请求参数>
```

正常返回示例

`XML` **格式**

```
<DeleteConsumerGroupResponse>
    <Message>operation success</Message>
    <RequestId>1AA2A2AD-2727-4573-B1C7-A0388BCD61CC</RequestId>
    <Success>true</Success>
    <Code>200</Code>
</DeleteConsumerGroupResponse>
```

`JSON` **格式**

```
{
 "Message":"operation success",
 "RequestId":"1AA2A2AD-2727-4573-B1C7-A0388BCD61CC",
 "Success":true,
 "Code":200
}
```

错误码

HttpCode	错误码	错误信息	描述
500	InternalError	An internal error occurred; please try again later.	系统内部错误，请稍后重试

访问错误中心查看更多错误码。

2.6 HTTP 调用方式

2.6.1 概述

本文适用基于 API URL 发起 HTTP/HTTPS POST 请求的用户，如果您使用的是 SDK、阿里云 CLI 或者 API Explorer，可以跳过此环节。

发起 API 请求的 URL 由不同参数拼凑而成，有固定的#unique_32。URL 中包含#unique_33、您的#unique_34和某个 API 的具体参数。每篇 API 文档均给出了 URL 请求示例供您参考，但是为了方便显示，我们并没有编码这些 URL 示例，您需要在发起请求前自行编码。我们根据您的签名验证了请求后，会返回结果给您。接口调用成功会显示返回参数，调用失败则显示相应报错，您可以根据公共错误码和具体 API 错误码分析排查。返回参数的详情请参见 SDK 接口文档。

说明:

推荐您使用 SDK，以免除您手动签名验证环节，方便调用接口以及管理资源。

2.6.2 公共参数

公共参数分为公告请求参数和公共返回参数和。

公共请求参数

以下公共请求参数适用于通过 URL 发送 POST 请求调用消息队列 for Apache Kafka 的 API。

名称	类型	是否必需	描述
Action	String	是	API 的名称。取值请参见#unique_36。
AccessKeyId	String	是	访问密钥 ID。AccessKey（包含 AccessKeyId 和 AccessKeySecret）用于调用 API，而用户密码用于登录消息队列 for Apache Kafka 控制台。
Signature	String	是	您的签名。取值请参见#unique_34。
SignatureMethod	String	是	签名方式。取值范围：HMAC-SHA1。
SignatureVersion	String	是	签名算法版本。取值范围：1.0。

名称	类型	是否必需	描述
SignatureNonce	String	是	签名唯一随机数。用于防止网络重放攻击，建议您每一次请求都使用不同的随机数。
Timestamp	String	是	请求的时间戳。按照 ISO8601 标准表示，并需要使用 UTC 时间，格式为 `yyyy-MM-ddTHH:mm:ssZ`。示例：`2018-01-01T12:00:00Z` 表示北京时间 2018 年 1 月 1 日 20 点 00 分 00 秒。
Version	String	是	API 的版本号，格式为 `YYYY-MM-DD`。取值范围：2018-10-15。
Format	String	否	返回参数的语言类型。取值范围：json 或 xml。默认值：xml。

请求示例

```
https:/alikafka.cn-hangzhou.aliyuncs.com/?Action=XXXXXX
&Format=xml
&Version=2014-05-26
&Signature=Pc5WB8gokVn0xfeu%2FZV%2BiNM1dgI%3D
&SignatureMethod=HMAC-SHA1
&SignatureNonce=15215528852396
&SignatureVersion=1.0
&AccessKeyId=key-test
&Timestamp=2018-01-01T12:00:00Z
...
```

公共返回参数

名称	类型	描述
RequestId	String	请求 ID。无论调用接口成功与否，我们都会返回请求 ID。

2.6.3 请求示例

本文提供使用 Java 语言构造参数和签名并 POST 发送请求的示例，其他语言流程类似。

使用示例

```
  private static final String ISO8601_DATE_FORMAT = "yyyy-MM-dd'T'HH:
mm:ss'Z'";
    private static final String ENCODING = "UTF-8";
    private static final String HTTP_METHOD = "POST";
    //购买的实例所在地域的 Region ID
    private static final String REGION_ID = "cn-xxxxxx";
    private static final String ACCESS_KEY = "xxxxxx";
    private static final String ACCESS_KEY_SECRET = "xxxxxx";


    public static void main(String[] args) {
        try {
            // 1. 设置参数
            Map<String, String> parameters = buildCommonParams();
            // 2. 排序请求参数：根据字母排序
            String[] sortedKeys = sortParamsToArray(parameters);
            // 3. 构造 stringToSign 字符串
            String stringToSign = buildSignString(parameters,
sortedKeys);
            // 4. 计算签名
            String signature = calculateSignature(stringToSign);
            // 5. 将签名设置到参数中
            parameters.put("Signature", signature);
            // 6. 发送 POST 请求
            String result = post(String.format("http://alikafka.%s.
aliyuncs.com/", REGION_ID), parameters);
            // 7. 验证结果
            System.out.println(result);
        } catch (Throwable throwable) {
            throwable.printStackTrace();
        }
    }

    private static Map<String, String> buildCommonParams() {
        Map<String, String> parameters = Maps.newHashMap();
        // Action 为请求方法
        // GetInstanceList：获取实例信息；GetConsumerList：获取消费组列表；
GetTopicList：获取 Topic 列表；GetTopicStatus：获取 Topic 状态
        // GetConsumerProgress：获取消费状态；CreateTopic：创建 Topic；
CreateConsumerGroup：创建 Consumer Group
        parameters.put("Action", "GetInstanceList");
        // 接口版本："2018-10-15"
        parameters.put("Version", "2018-10-15");
        // 请求参数：RegionId 为必填参数
        // 如果接口有其他参数请都设置：比如 InstanceId/Topic/ConsumerId
        // parameters.put("InstanceId", "cn-huhehaote");
        // parameters.put("Topic", "cn-huhehaote");
        // parameters.put("Remark", "cn-huhehaote");
        // parameters.put("ConsumerId", "cn-huhehaote");
        parameters.put("RegionId", REGION_ID);
        parameters.put("AccessKeyId", ACCESS_KEY);
        // 时间戳，注意格式:yyyy-MM-dd'T'HH:mm:ss'Z'
        parameters.put("Timestamp", formatIso8601Date(new Date()));
        parameters.put("SignatureMethod", "HMAC-SHA1");
        parameters.put("SignatureVersion", "1.0");
        parameters.put("SignatureNonce", UUID.randomUUID().toString
());
```

```
        parameters.put("Format", "json");
        return parameters;
    }

    private static String[] sortParamsToArray(Map<String, String>
parameters) {
        String[] sortedKeys = parameters.keySet().toArray(new String[]
{});
        Arrays.sort(sortedKeys);
        return sortedKeys;
    }

    private static String buildSignString(Map<String, String>
parameters,
                                          String[] sortedKeys) throws
UnsupportedEncodingException {
        StringBuilder stringToSign = new StringBuilder();
        String SEPARATOR = "&";
        stringToSign.append(HTTP_METHOD).append(SEPARATOR);
        stringToSign.append(percentEncode("/")).append(SEPARATOR);
        StringBuilder canonicalizedQueryString = new StringBuilder();
        for(String key : sortedKeys) {
            // 这里注意编码 key 和 value
            canonicalizedQueryString.append("&")
                .append(percentEncode(key)).append("=")
                .append(percentEncode(parameters.get(key)));
        }

        // 这里注意编码 canonicalizedQueryString
        stringToSign.append(percentEncode(canonicalizedQueryString.
toString().substring(1)));
        return stringToSign.toString();
    }

    private static String calculateSignature(String stringToSign)
        throws NoSuchAlgorithmException, InvalidKeyException,
UnsupportedEncodingException {
        String ALGORITHM = "HmacSHA1";
        String ENCODING = "UTF-8";
        //对应账号的 AccessKeySecret
        String accessKeySecret = ACCESS_KEY_SECRET + "&";
        Mac mac = Mac.getInstance(ALGORITHM);
        mac.init(new SecretKeySpec(accessKeySecret.getBytes(ENCODING
), ALGORITHM));
        byte[] signData = mac.doFinal(stringToSign.getBytes(ENCODING
));
        return new String(Base64.getEncoder().encode(signData));
    }

    private static String percentEncode(String value) throws
UnsupportedEncodingException {
        return value != null ? URLEncoder.encode(value, ENCODING).
replace("+", "%20").replace("*", "%2A").replace("%7E", "~") : null;
    }

    private static String formatIso8601Date(Date date) {
        SimpleDateFormat df = new SimpleDateFormat(ISO8601_DATE_FORMAT
);
        df.setTimeZone(new SimpleTimeZone(0, "GMT"));
        return df.format(date);
    }

    private static String post(String url, Map<String, String>
paramMap) throws IOException {
```

```
        Form form = Form.form();
        for (String key : paramMap.keySet()) {
            form.add(key, paramMap.get(key));
        }

        return Request.Post(url).bodyForm(form.build()).connectTimeout
(10000).execute().returnContent().asString();
    }
```

2.6.4 请求结构

消息队列 for Apache Kafka 的 API 支持基于 URL 发起 HTTP/HTTPS POST 请求。请求参数需要包含在 URL 中。本文列举了 POST 请求中的结构解释，并提供了消息队列 for Apache Kafka 的服务接入地址（Endpoint）。

结构示例

以下为 GetInstanceList 一条未编码的 URL 请求示例：

```
http://alikafka.cn-hangzhou.aliyuncs.com/?Action=GetInstanceList
&RegionId=cn-hangzhou
&<公共请求参数>
```

- `http` 指定了请求通信协议。
- `alikafka.cn-hangzhou.aliyuncs.com` 指定了消息队列 for Apache Kafka 的服务接入地址（Endpoint）。
- `Action=GetInstanceList` 指定了要调用的 API。
- `<公共请求参数>`是系统规定的公共参数。

通信协议

支持 HTTP 或 HTTPS协议请求通信。为了获得更高的安全性，推荐您使用 HTTPS 协议发送请求。

接入地址

地域名称	RegionId	Domain
华东1（杭州）	cn-hangzhou	alikafka.cn-hangzhou.aliyuncs.com
华东2（上海）	cn-shanghai	alikafka.cn-shanghai.aliyuncs.com
华北1（青岛）	cn-qingdao	alikafka.cn-qingdao.aliyuncs.com

地域名称	RegionId	Domain
华北2（北京）	cn-beijing	alikafka.cn-beijing.aliyuncs.com
华北3（张家口）	cn-zhangjiakou	alikafka.cn-zhangjiakou.aliyuncs.com
华北5（呼和浩特）	cn-huhehaote	alikafka.cn-huhehaote.aliyuncs.com
华南1（深圳）	cn-shenzhen	alikafka.cn-shenzhen.aliyuncs.com
中国（香港）	cn-hongkong	alikafka.cn-hongkong.aliyuncs.com
新加坡	ap-southeast-1	alikafka.ap-southeast-1.aliyuncs.com

请求参数

您需要通过 `Action` 参数指定目标操作，例如 `Action=GetInstanceList`，还需要指定接口的其他参数以及#unique_33。`Action` 参数的可取值如下：

- GetInstanceList：获取实例信息；
- GetConsumerList：获取消费组列表；
- GetTopicList：获取 Topic 列表；
- GetTopicStatus：获取 Topic 状态；
- GetConsumerProgress：获取消费状态；
- CreateTopic：创建 Topic；
- CreateConsumerGroup：创建消费组。

字符编码

请求及返回结果都使用 UTF-8 字符集编码。

2.6.5 签名机制

对于每一次 HTTP 或者 HTTPS 协议请求，我们会根据访问中的签名信息验证访问请求者身份。具体使用 AccessKey 的 AccessKeyId 和 AccessKeySecret 对称加密验证实现。

您的用户名/密码是用于登录消息队列 for Apache Kafka 控制台的身份凭据，与此类似，AccessKey 是用于调用 API 的身份凭据。AccessKey 中的 AccessKeyId 是访问者身份，AccessKeySecret 是加密签名字符串和服务器端验证签名字符串的密钥，必须严格保密谨防泄露。

> **说明:**
> 消息队列 for Apache Kafka 提供了多种编程语言的 SDK，可以免去您签名的烦恼。更多详情，请下载 SDK。

步骤一：构造规范化请求字符串

1. 排序参数。排序规则以首字母顺序排序，排序参数包括#unique_33/unique_33_Connect_42_section_02z_sq3_nwo和接口自定义参数，不包括公共请求参数中的 `Signature` 参数。
2. 编码参数。使用 UTF-8 字符集按照 RFC3986 规则编码请求参数和参数取值，编码规则如下：
   - 字符 A~Z、a~z、0~9 以及字符 “-”、“_”、“.”、“~” 不编码。
   - 其它字符编码成 `%XY` 的格式，其中 `XY` 是字符对应 ASCII 码的 16 进制。示例：半角双引号（"）对应 `%22`。
   - 扩展的 UTF-8 字符，编码成 `%XY%ZA...` 的格式。
   - 空格（ ）编码成 `%20`，而不是加号（+）。

   该编码方式与 `application/x-www-form-urlencoded MIME` 格式编码算法相似，但又有所不同。

   如果您使用的是 Java 标准库中的 `java.net.URLEncoder`，可以先用标准库中 `percentEncode` 编码，随后将编码后的字符中加号（+）替换为 `%20`、星号（*）替换为 `%2A`、`%7E` 替换为波浪号（~），即可得到上述规则描述的编码字符串。示例如下：

   ```
   private static final String ENCODING = "UTF-8";
   private static String percentEncode(String value) throws
   UnsupportedEncodingException {
   return value != null ? URLEncoder.encode(value, ENCODING).replace
   ("+", "%20").replace("*", "%2A").replace("%7E", "~") : null;
   }
   ```
3. 使用等号（=）连接编码后的请求参数和参数取值。
4. 使用与号（&）连接编码后的请求参数，注意参数排序与步骤 1 一致。

现在，您得到了规范化请求字符串（`CanonicalizedQueryString`），其结构遵循#unique_32。

步骤二：构造签名字符串

1. 构造待签名字符串 `StringToSign`。您可以同样使用 `percentEncode` 处理上一步构造的规范化请求字符串，规则如下：

   ```
   StringToSign=
   ```

```
HTTPMethod + "&" + //HTTPMethod: 发送请求的 HTTP 方法，例如 POST。
percentEncode("/") + "&" + //percentEncode("/"): 字符 (/) UTF-8 编码得到
的值，即 %2F。
percentEncode(CanonicalizedQueryString) //您的规范化请求字符串。
```

2. 按照 RFC2104 的定义，计算待签名字符串 `StringToSign` 的 HMAC-SHA1 值。示例使用的是 Java Base64 编码方法。

```
Signature = Base64( HMAC-SHA1( AccessSecret, UTF-8-Encoding-Of(
StringToSign) ) )
```

> **说明:**
> 计算签名时，RFC2104 规定的 Key 值是您的 AccessKeySecret 并加上与号（&），其 ASCII 值为 38。

3. 添加根据 RFC3986 规则编码后的参数 `Signature` 到规范化请求字符串URL中。

示例一：参数拼接法

以调用 `GetInstanceList` 获取实例列表为例。假设您获得了 `AccessKeyID=testid` 以及 `AccessKeySecret=testsecret`，签名流程如下所示：

1. 构造规范化请求字符串。

```
http://alikafka.%s.aliyuncs.com/?Timestamp=2016-02-23T12:46:24Z
&Format=XML&AccessKeyId=testid&Action=GetInstanceList&SignatureM
ethod=HMAC-SHA1&SignatureNonce=3ee8c1b8-83d3-44af-a94f-4e0ad82fd6cf&
Version=2014-05-26&SignatureVersion=1.0
```

2. 构造待签名字符串 StringtoSign。

```
POST&%2F&AccessKeyId%3Dtestid&Action%3DGetInstanceList&Format%3DXML
&SignatureMethod%3DHMAC-SHA1&SignatureNonce%3D3ee8c1b8-83d3-44af-
a94f-4e0ad82fd6cf&SignatureVersion%3D1.0&Timestamp%3D2016-02-23T12%
253A46%253A24Z&Version%3D2014-05-26
```

3. 计算签名值。因为 `AccessKeySecret=testsecret`，用于计算的 Key 为 `testsecret&`，计算得到的签名值为 `OLeaidS1JvxuMvnyHOwuJ+uX5qY=`。示例使用的是 Java Base64 编码方法。

```
Signature = Base64( HMAC-SHA1( AccessSecret, UTF-8-Encoding-Of(
StringToSign) ) )
```

4. 添加 RFC3986 规则编码后的 `Signature=OLeaidS1JvxuMvnyHOwuJ%2BuX5qY%3D` 到步骤 1 的 URL 中。

```
http://alikafka.%s.aliyuncs.com/?SignatureVersion=1.0&Action=
GetInstanceList&Format=JSON&SignatureNonce=3ee8c1b8-83d3-44af-
a94f-4e0ad82fd6cf&Version=2014-05-26&AccessKeyId=testid&Signature=
OLeaidS1JvxuMvnyHOwuJ%2BuX5qY%3D&SignatureMethod=HMAC-SHA1&Timestamp
=2016-02-23T12%253A46%253A24Z
```

通过以上 URL，您可以使用浏览器、curl 或者 wget 等工具发起 HTTP 请求调用 `GetInstanceList`，查看阿里云的地域列表。

示例二：编程语言法

依然以调用 `GetInstanceList` 获取实例列表为例。假设您获得了 `AccessKeyID=testid` 以及 `AccessKeySecret=testsecret`，并且假定所有请求参数放在一个 `Java Map<String, String>` 对象里。

1. 预定义编码方法。

```
private static final String ENCODING = "UTF-8";
private static String percentEncode(String value) throws Unsupporte
dEncodingException {
return value != null ? URLEncoder.encode(value, ENCODING).replace
("+", "%20").replace("*", "%2A").replace("%7E", "~") : null;
}
```

2. 预定义编码时间格式 `Timestamp`。参数 `Timestamp` 必须符合 ISO8601 规范，并需要使用 UTC 时间，时区为 +0。

```
private static final String ISO8601_DATE_FORMAT = "yyyy-MM-dd'T'HH:
mm:ss'Z'";
private static String formatIso8601Date(Date date) {
SimpleDateFormat df = new SimpleDateFormat(ISO8601_DATE_FORMAT);
df.setTimeZone(new SimpleTimeZone(0, "GMT"));
return df.format(date);
}
```

3. 构造请求字符串。

```
final String HTTP_METHOD = "POST";
Map parameters = new HashMap();
// 输入请求参数
parameters.put("Action", "GetInstanceList");
parameters.put("Version", "2014-05-26");
parameters.put("AccessKeyId", "testid");
parameters.put("Timestamp", formatIso8601Date(new Date()));
parameters.put("SignatureMethod", "HMAC-SHA1");
parameters.put("SignatureVersion", "1.0");
parameters.put("SignatureNonce", UUID.randomUUID().toString());
```

```
parameters.put("Format", "JSON");
// 排序请求参数
String[] sortedKeys = parameters.keySet().toArray(new String[]{});
Arrays.sort(sortedKeys);
final String SEPARATOR = "&";
// 构造 stringToSign 字符串
StringBuilder stringToSign = new StringBuilder();
stringToSign.append(HTTP_METHOD).append(SEPARATOR);
stringToSign.append(percentEncode("/")).append(SEPARATOR);
StringBuilder canonicalizedQueryString = new StringBuilder();
for(String key : sortedKeys) {
// 这里注意编码 key 和 value
canonicalizedQueryString.append("&")
.append(percentEncode(key)).append("=")
.append(percentEncode(parameters.get(key)));
}
// 这里注意编码 canonicalizedQueryString
stringToSign.append(percentEncode(
canonicalizedQueryString.toString().substring(1)));
```

4. 签名。因为 `AccessKeySecret=testsecret`，所以用于计算 HMAC 的 Key 为 `testsecret&`，计算得到的签名值为 `OLeaidS1JvxuMvnyHOwuJ%2BuX5qY%3D`。

```
// 以下是一段计算签名的示例代码
final String ALGORITHM = "HmacSHA1";
final String ENCODING = "UTF-8";
key = "testsecret&";
Mac mac = Mac.getInstance(ALGORITHM);
mac.init(new SecretKeySpec(key.getBytes(ENCODING), ALGORITHM));
byte[] signData = mac.doFinal(stringToSign.getBytes(ENCODING));
String signature = new String(Base64.encodeBase64(signData));
```