分布式消息服务 Kafka 版

最佳实践

文档版本 01

发布日期 2025-08-20

华为云计算技术有限公司

版权所有 © 华为云计算技术有限公司 2025。 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

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 Kafka 最佳实践汇总........1
2 提高 Kafka 消息处理效率........2
3 Logstash 对接 Kafka 生产消费消息........4
4 使用 MirrorMaker 跨集群同步数据........12
5 消息堆积处理建议........16
6 业务过载处理建议........18
7 业务数据不均衡处理建议........20
8 Kafka Topic 分区数设置建议........22
9 配置消息堆积数监控........24
10 DMS for Kafka 安全使用建议........28
11 优化消费者轮询（Polling）........30

1 Kafka 最佳实践汇总

本文汇总了基于分布式消息服务Kafka版常见应用场景的操作实践，为每个实践提供详细的方案描述和操作指导，帮助用户轻松使用Kafka。

表 1-1 Kafka 最佳实践一览表

最佳实践	说明
提高Kafka消息处理效率	本章节提供了生产者和消费者对于消息的使用建议，以提高消息发送和消息消费的效率与可靠性。
Logstash对接Kafka生产消费消息	Kafka实例可以作为Logstash输入源，也可以作为Logstash输出源。本章节介绍Logstash如何对接Kafka实例生产消费消息。
使用MirrorMaker跨集群同步数据	使用MirrorMaker可以实现将源集群中的数据镜像复制到目标集群中。本章节介绍两个Kafka实例如何使用MirrorMaker单向或双向同步数据。
消息堆积处理建议	本章节描述了消息堆积的原因，以及处理措施。
业务过载处理建议	本章节描述了CPU使用率高和磁盘写满的原因，以及处理措施。
业务数据不均衡处理建议	本章节描述了业务数据不均衡的原因，以及处理措施。
Kafka Topic分区数设置建议	本章节提供了分区数的设置建议，以及典型问题案例。
配置消息堆积数监控	本章节介绍如何创建消息堆积数超过阈值的告警规则，实现消息堆积数超过阈值时，系统自动发送短信/邮件通知用户，让用户能够实时掌握业务的运行情况。
DMS for Kafka安全使用建议	本章节提供了Kafka使用过程中的安全最佳实践，旨在为提高整体安全能力提供可操作的规范性指导。
优化消费者轮询（Polling）	本章节介绍在对消费消息实时性要求不高的场景中，如何优化消费者Polling，减少消息较少或者没有消息时的资源浪费。

版权所有 © 华为云计算技术有限公司

2 提高 Kafka 消息处理效率

消息发送和消费的可靠性必须由分布式消息服务Kafka版和生产者以及消费者协同工作才能保证。同时开发者需要尽量合理使用分布式消息服务Kafka版的Topic，以提高消息发送和消息消费的效率与准确性。

对使用分布式消息服务Kafka版的生产者和消费者有如下的使用建议：

重视消息生产与消费的确认过程

消息生产

发送消息后，生产者需要根据分布式消息服务Kafka版的返回信息确认消息是否发送成功，如果返回失败需要重新发送。

生产消息时，生产者通过同步等待发送结果或异步回调函数，判断消息是否发送成功。在消息传递过程中，如果发生异常，生产者没有接收到发送成功的信号，生产者自己决策是否需要重复发送消息。如果接收到发送成功的信号，则表明该消息已经被分布式消息服务Kafka版可靠存储。

消息消费

消息消费时，消费者需要确认消息是否已被成功消费。

生产的消息被依次存储在分布式消息服务Kafka版的存储介质中。消费时依次获取分布式消息服务Kafka版中存储的消息。消费者获取消息后，进行消费并记录消费成功或失败的状态，并将消费状态提交到分布式消息服务Kafka版。

在消费过程中，如果出现异常，没有提交消费确认，该批消息会在后续的消费请求中再次被获取。

消息生产与消费的幂等传递

分布式消息服务Kafka版设计了一系列可靠性保障措施，确保消息不丢失。例如使用消息同步存储机制防止系统与服务器层面的异常重启或者掉电，使用消息确认（ACK）机制解决消息传输过程中遇到的异常。

考虑到网络异常等极端情况，用户除了做好消息生产与消费的确认，还需要配合分布式消息服务Kafka版完成消息发送与消费的重复传输设计。

- 当无法确认消息是否已发送成功，生产者需要将消息重复发送给分布式消息服务Kafka版。

- 当重复收到已处理过的消息，消费者需要告诉分布式消息服务Kafka版消费成功且保证不重复处理。

消息可以批量生产和消费

为提高消息发送和消息消费效率，推荐使用批量消息发送和消费。

图 2-1 消息批量生产与消费

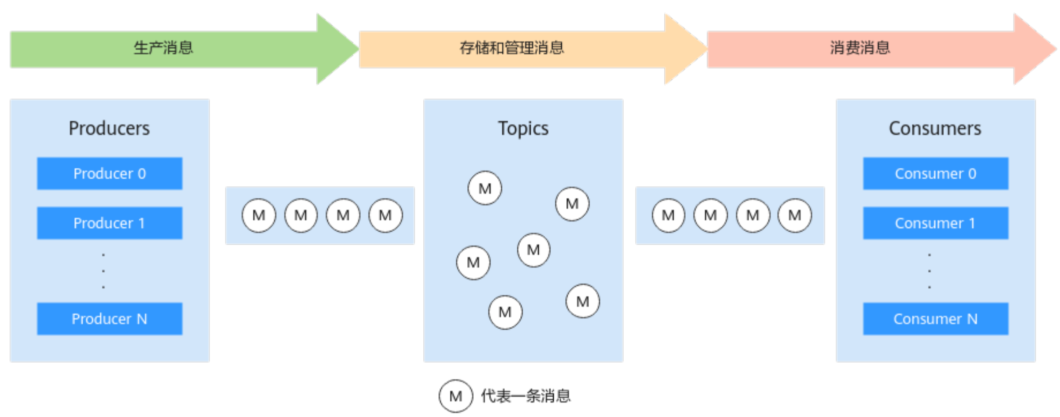
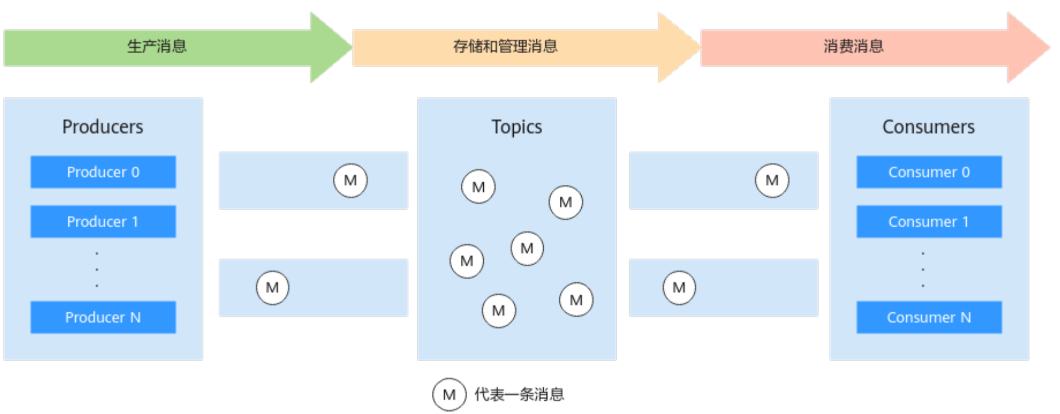


图 2-2 消息逐条生产与消费



此外，批量消费消息时，消费者应按照接收的顺序对消息进行处理、确认，当对某一条消息处理失败时，不再需要继续处理本批消息中的后续消息，直接对已正确处理的消息进行确认即可。

巧用消费组协助运维

用户使用分布式消息服务Kafka版作为消息中间件，查看Topic的消息内容对于定位问题与调试服务是至关重要的。

当消息的生产和消费过程中遇到疑难问题时，通过创建不同消费组可以帮助定位分析问题或调试服务对接。用户可以创建一个新的消费组，对Topic中的消息进行消费并分析消费过程，这样不会影响其他服务对消息的处理。

版权所有 © 华为云计算技术有限公司

3 Logstash 对接 Kafka 生产消费消息

方案概述

应用场景

Logstash是免费且开放的服务器端数据处理管道，能够从多个来源采集数据，转换数据，然后将数据发送到指定的存储中。Kafka是一种高吞吐量的分布式发布订阅消息系统，也是Logstash支持的众多输入输出源之一。本章节主要介绍Logstash如何对接Kafka实例。

方案架构

- Kafka实例作为Logstash输出源的示意图如下。

图 3-1 Kafka 实例作为 Logstash 输出源

Logstash从数据库采集数据，然后发送到Kafka实例中进行存储。Kafka实例作为Logstash输出源时，由于Kafka的高吞吐量，可以存储大量数据。

- Kafka实例作为Logstash输入源的示意图如下。

图 3-2 Kafka 实例作为 Logstash 输入源

日志采集客户端将数据发送到Kafka实例中，Logstash根据自身性能从Kafka实例中拉取数据。Kafka实例作为Logstash输入源时，可以防止突发流量对于Logstash的影响，以及解耦日志采集客户端和Logstash，保证系统的稳定性。

约束与限制

Logstash从7.5版本开始支持Kafka Integration Plugin插件，Kafka Integration Plugin插件包含Kafka input Plugin和Kafka output Plugin。Kafka input Plugin用于从Kafka实例的Topic中读取数据，Kafka output Plugin把数据写入到Kafka实例的Topic。Logstash、Kafka Integration Plugin与Kafka客户端的版本对应关系如表3-1所示。**请确保Kafka客户端版本大于或等于Kafka实例的版本。**

表 3-1 版本对应关系

Logstash版本	Kafka Integration Plugin版本	Kafka客户端版本
8.3~8.8	10.12.0	2.8.1
8.0~8.2	10.9.0~10.10.0	2.5.1
7.12~7.17	10.7.4~10.9.0	2.5.1
7.8~7.11	10.2.0~10.7.1	2.4
7.6~7.7	10.0.1	2.3.0
7.5	10.0.0	2.1.0

前提条件

执行实施步骤前，请确保已完成以下操作：

- **下载Logstash**。
- 准备一台Windows系统的主机，在主机中安装**Java Development Kit 1.8.111或以上版本**和Git Bash。
- **创建Kafka实例**和**Topic**，并获取Kafka实例信息。

  Kafka实例未开启公网访问和SASL认证时，获取**表3-2**所示信息。

  表 3-2 Kafka 实例信息（未开启公网访问和 SASL 认证）

参数名	获取途径
内网连接地址	在Kafka实例详情页的“连接信息”区域，获取“内网连接地址”。
Topic名称	在Kafka实例控制台，单击实例名称，进入实例详情页。在左侧导航栏单击“Topic管理”，进入Topic列表页面，获取Topic名称。 下文以topic-logstash为例介绍。

  Kafka实例未开启公网访问、已开启SASL认证时，获取**表3-3**所示信息。

  表 3-3 Kafka 实例信息（未开启公网访问、已开启 SASL 认证）

参数名	获取途径
内网连接地址	在Kafka实例详情页的“连接信息”区域，获取“内网连接地址”。
开启的SASL认证机制	在Kafka实例详情页的“连接信息”区域，获取“开启的SASL认证机制”。

参数名	获取途径
启用的安全协议	在Kafka实例详情页的“连接信息”区域，获取“启用的安全协议”。
证书	在Kafka实例详情页的“连接信息”区域，在“SSL证书”所在行，单击“下载”。下载压缩包后解压，获取压缩包中的客户端证书文件：client.jks。
SASL用户名和密码	在Kafka实例控制台，单击实例名称，进入实例详情页。在左侧导航栏单击“用户管理”，进入用户列表页面，获取用户名。如果忘记了密码，单击“重置密码”，重新设置密码。
Topic名称	在Kafka实例控制台，单击实例名称，进入实例详情页。在左侧导航栏单击“Topic管理”，进入Topic列表页面，获取Topic名称。 下文以topic-logstash为例介绍。

Kafka实例已开启公网访问、未开启SASL认证时，获取**表3-4**所示信息。

表 3-4 Kafka 实例信息（已开启公网访问、未开启 SASL 认证）

参数名	获取途径
公网连接地址	在Kafka实例详情页的“连接信息”区域，获取“公网连接地址”。
Topic名称	在Kafka实例控制台，单击实例名称，进入实例详情页。在左侧导航栏单击“Topic管理”，进入Topic列表页面，获取Topic名称。 下文以topic-logstash为例介绍。

Kafka实例已开启公网访问和SASL认证时，获取**表3-5**所示信息。

表 3-5 Kafka 实例信息（已开启公网访问和 SASL 认证）

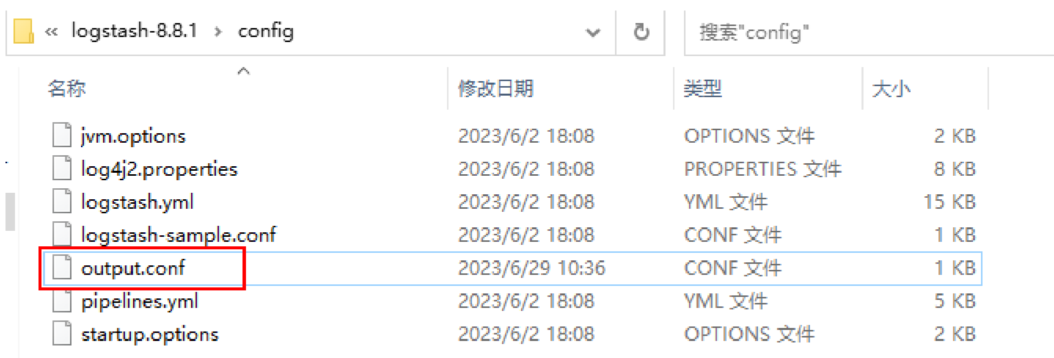
参数名	获取途径
公网连接地址	在实例详情页的“连接信息”区域，获取“公网连接地址”
开启的SASL认证机制	在Kafka实例详情页的“连接信息”区域，获取“开启的SASL认证机制”。
启用的安全协议	在Kafka实例详情页的“连接信息”区域，获取“启用的安全协议”。
证书	在Kafka实例详情页的“连接信息”区域，在“SSL证书”所在行，单击“下载”。下载压缩包后解压，获取压缩包中的客户端证书文件：client.jks。

参数名	获取途径
SASL用户名和密码	在Kafka实例控制台，单击实例名称，进入实例详情页。在左侧导航栏单击“用户管理”，进入用户列表页面，获取用户名。如果忘记了密码，单击“重置密码”，重新设置密码。
Topic名称	在Kafka实例控制台，单击实例名称，进入实例详情页。在左侧导航栏单击“Topic管理”，进入Topic列表页面，获取Topic名称。 下文以topic-logstash为例介绍。

实施步骤（Kafka 实例作为 Logstash 输出源）

步骤1 在Windows主机中，解压Logstash压缩包，进入“config”文件夹，创建“output.conf”配置文件。

图 3-3 创建“output.conf”配置文件



步骤2 在“output.conf”配置文件中，增加如下内容，连接Kafka实例。

```
input {
    stdin {}
}
output {
 kafka {
     bootstrap_servers => "ip1:port1,ip2:port2,ip3:port3"
     topic_id => "topic-logstash"

     #如果不使用SASL认证，以下参数请注释掉。
     #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';"

     #安全协议为“SASL_SSL”时，配置信息如下。
     security_protocol => "SASL_SSL"
     ssl_truststore_location => "C:\\Users\\Desktop\\logstash-8.8.1\\config\\client.jks"
     ssl_truststore_password => "dms@kafka"
     ssl_endpoint_identification_algorithm => ""

     #安全协议为“SASL_PLAINTEXT”时，配置信息如下。
     security_protocol => "SASL_PLAINTEXT"
     }
}
```


版权所有 © 华为云计算技术有限公司

参数说明如下：

- bootstrap_servers：前提条件中获取的Kafka实例“内网连接地址”/“公网连接地址”。
- topic_id：前提条件中获取的Topic名称。
- sasl_mechanism：SASL认证机制。
- sasl_jaas_config：SASL jaas的配置文件，根据实际情况修改为前提条件中获取的SASL用户名和密码。
- security_protocol：Kafka实例的安全协议。
- ssl_truststore_location：SSL证书的存放位置。
- ssl_truststore_password：服务器证书密码，**不可更改，需要保持为dms@kafka**。
- ssl_endpoint_identification_algorithm：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**。

如果需要了解Kafka output Plugin的其他参数，请参见**Kafka output Plugin**。

步骤3 在Logstash文件夹根目录打开Git Bash，执行以下命令启动Logstash。

```
./bin/logstash -f ./config/output.conf
```

返回“Successfully started Logstash API endpoint”时，表示启动成功。

图 3-4 启动 Logstash

步骤4 在Logstash中，生产消息，如下图所示。

图 3-5 生产消息

步骤5 切换到Kafka控制台，单击实例名称，进入实例详情页。

步骤6 在左侧导航栏单击“消息查询”，进入消息查询页面。

步骤7 在“Topic名称”中选择“topic-logstash”，单击“搜索”，查询消息。

图 3-6 查询消息

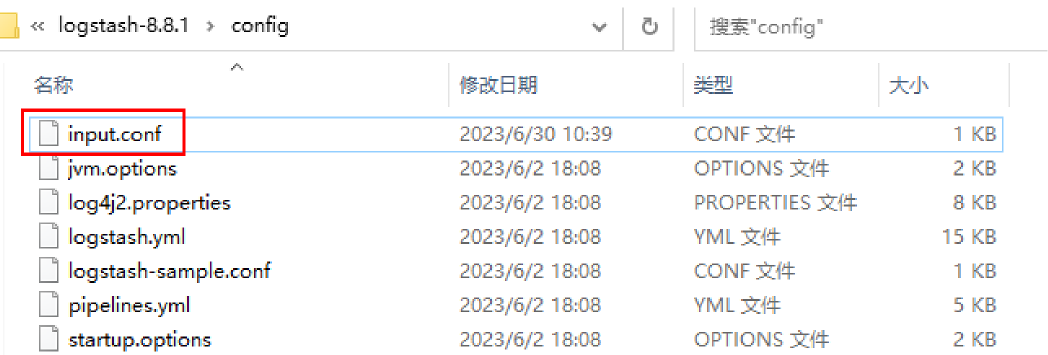
从图3-6可以看出，Logstash的Kafka output Plugin已经把数据写入到Kafka实例的topic-logstash中。

----结束

实施步骤（Kafka 实例作为 Logstash 输入源）

步骤1 在Windows主机中，解压Logstash压缩包，进入“config”文件夹，创建“input.conf”配置文件。

图 3-7 创建“input.conf”配置文件



步骤2 在“input.conf”配置文件中，增加如下内容，连接Kafka实例。

```
input {
 kafka {
    bootstrap_servers => "ip1:port1,ip2:port2,ip3:port3"
    group_id => "logstash_group"
    topic_id => "topic-logstash"
    auto_offset_reset => "earliest"

    #如果不使用SASL认证，以下参数请注释掉。
    #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';"

    #安全协议为“SASL_SSL”时，配置信息如下。
    security_protocol => "SASL_SSL"
    ssl_truststore_location => "C:\\Users\\Desktop\\logstash-8.8.1\\config\\client.jks"
    ssl_truststore_password => "dms@kafka"
    ssl_endpoint_identification_algorithm => ""

    #安全协议为“SASL_PLAINTEXT”时，配置信息如下。
    security_protocol => "SASL_PLAINTEXT"
```

```
    }
}
output {
 stdout{codec=>rubydebug}
}
```

参数说明如下：

- bootstrap_servers：前提条件中获取的Kafka实例“内网连接地址”/“公网连接地址”。
- group_id：消费组的名称。
- topic_id：前提条件中获取的Topic名称。
- auto_offset_reset：指定消费者的消费策略。latest表示偏移量自动被重置到最晚偏移量，earliest表示偏移量自动被重置到最早偏移量，none表示向消费者抛出异常。本文以earliest为例。
- sasl_mechanism：SASL认证机制。
- sasl_jaas_config：SASL jaas的配置文件，根据实际情况修改为前提条件中获取的SASL用户名和密码。
- security_protocol：Kafka实例的安全协议。
- ssl_truststore_location：SSL证书的存放位置。
- ssl_truststore_password：服务器证书密码，**不可更改，需要保持为dms@kafka**。
- ssl_endpoint_identification_algorithm：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**。

如果需要了解Kafka input Plugin的其他参数，请参见**Kafka input Plugin**。

步骤3 在Logstash文件夹根目录打开Git Bash，执行以下命令启动Logstash。

```
./bin/logstash -f ./config/input.conf
```

Logstash启动成功后，Kafka input Plugin会自动从Kafka实例的topic-logstash中读取数据，如下图所示。

版权所有 © 华为云计算技术有限公司

图 **3-8** Logstash 从 topic-logstash 中读取的数据

```
[2023-06-29T18:04:42,600][INFO ][org.apache.kafka.clients.consumer.internals.SubscriptionState][main
to position FetchPosition{offset=0, offsetEpoch=Optional.empty, currentLeader=LeaderAndEpoch{leader=
[2023-06-29T18:04:42,604][INFO ][org.apache.kafka.clients.consumer.internals.SubscriptionState][main
to position FetchPosition{offset=0, offsetEpoch=Optional.empty, currentLeader=LeaderAndEpoch{leader=
[2023-06-29T18:04:42,605][INFO ][org.apache.kafka.clients.consumer.internals.SubscriptionState][main
to position FetchPosition{offset=0, offsetEpoch=Optional.empty, currentLeader=LeaderAndEpoch{leader=
{
       "message" => "2023-06-29T09:57:20.511653600Z {hostname=l[illegible]} hello",
    "@timestamp" => 2023-06-29T10:04:42.667403300Z,
      "@version" => "1",
         "event" => {
        "original" => "2023-06-29T09:57:20.511653600Z {hostname=lw[illegible]} hello"
    }
}
{
       "message" => "2023-06-29T09:59:40.461979100Z {hostname=lw[illegible]} ^C",
    "@timestamp" => 2023-06-29T10:04:42.671392700Z,
      "@version" => "1",
         "event" => {
        "original" => "2023-06-29T09:59:40.461979100Z {hostname=lw[illegible]} ^C"
    }
}
{
       "message" => "2023-06-29T09:57:23.415122800Z {hostname=lw[illegible]} 123!",
    "@timestamp" => 2023-06-29T10:04:42.671392700Z,
      "@version" => "1",
         "event" => {
        "original" => "2023-06-29T09:57:23.415122800Z {hostname=l[illegible]} 123!"
    }
}
{
       "message" => "2023-06-29T09:57:21.622637600Z {hostname=lw[illegible]} dms",
    "@timestamp" => 2023-06-29T10:04:42.671392700Z,
      "@version" => "1",
         "event" => {
        "original" => "2023-06-29T09:57:21.622637600Z {hostname=lw[illegible]} dms"
    }
}
```

----结束

版权所有 © 华为云计算技术有限公司

4 使用 MirrorMaker 跨集群同步数据

方案概述

应用场景

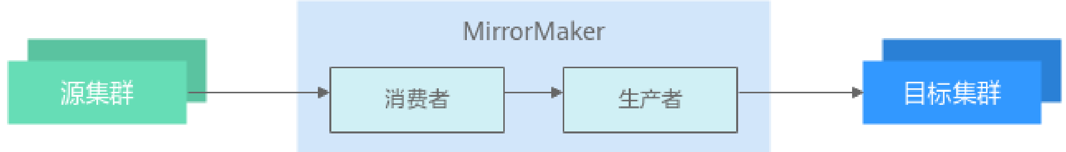
在以下场景，使用MirrorMaker进行不同集群间的数据同步，可以确保Kafka集群的可用性和可靠性。

- 备份和容灾：企业存在多个数据中心，为了防止其中一个数据中心出现问题，导致业务不可用，会将集群数据同步备份在多个不同的数据中心。
- 集群迁移：当今很多企业将业务迁移上云，迁移过程中需要确保线下集群和云上集群的数据同步，保证业务的连续性。

方案架构

使用MirrorMaker可以实现将源集群中的数据镜像复制到目标集群中。其原理如**图4-1**所示，MirrorMaker本质上也是生产消费消息，首先从源集群中消费数据，然后将消费的数据生产到目标集群。如果您需要了解更多关于MirrorMaker的信息，请参见**Mirroring data between clusters**。

图 4-1 MirrorMaker 原理图



约束与限制

- 源集群中节点的IP地址和端口号不能和目标集群中节点的IP地址和端口号相同，否则会导致数据在Topic内无限循环复制。
- 使用MirrorMaker同步数据，至少需要有两个或以上集群，不可在单个集群内部使用MirrorMaker，否则会导致数据在Topic内无限循环复制。

实施步骤

步骤1 购买一台弹性云服务器，确保弹性云服务器与源集群、目标集群网络互通。具体购买操作，请参考**购买弹性云服务器**。

步骤2 登录弹性云服务器，安装Java JDK，并配置JAVA_HOME与PATH环境变量，使用执行用户在用户家目录下修改“.bash_profile”，添加如下行。其中“/opt/java/jdk1.8.0_151”为JDK的安装路径，请根据实际情况修改。

```
export JAVA_HOME=/opt/java/jdk1.8.0_151
export PATH=$JAVA_HOME/bin:$PATH
```

执行**source .bash_profile**命令使修改生效。

说明

弹性云服务器默认自带的JDK可能不符合要求，例如OpenJDK，需要配置为Oracle的JDK，可至Oracle官方下载页面**下载Java Development Kit 1.8.111及以上版本**。

步骤3 下载**Kafka 3.3.1版本的二进制软件包**。

```
wget https://archive.apache.org/dist/kafka/3.3.1/kafka_2.12-3.3.1.tgz
```

步骤4 解压二进制软件包。

```
tar -zxvf kafka_2.12-3.3.1.tgz
```

步骤5 进入二进制软件包目录，修改“config”目录下的“connect-mirror-maker.properties”的配置文件，在配置文件中指定源集群和目标集群的IP地址和端口以及其他配置。

```
# 指定两个集群
clusters = A, B
A.bootstrap.servers = A_host1:A_port, A_host2:A_port, A_host3:A_port
B.bootstrap.servers = B_host1:B_port, B_host2:B_port, B_host3:B_port

# 指定数据同步方向，可以单向同步也可互相同步
A->B.enabled = true

# 指定同步的Topic，支持正则匹配，默认复制全部Topic，如："foo-.*"
A->B.topics = .*

# 取消以下两个配置的注释表示A、B两个集群互相复制同步
#B->A.enabled = true
#B->A.topics = .*

# 设置副本个数，如果需要同步多个Topic且副本数各不相同，建议先创建同名同副本数的Topic再启动
MirrorMaker
replication.factor=3

# 设置消费进度同步方向，可以单向同步也可互相同步
A->B.sync.group.offsets.enabled=true

############################# Internal Topic Settings  #############################
# The replication factor for mm2 internal topics "heartbeats", "B.checkpoints.internal" and
# "mm2-offset-syncs.B.internal"
# 测试环境可以为1，生产环境建议以下配置大于1，比如设为3
checkpoints.topic.replication.factor=3
heartbeats.topic.replication.factor=3
offset-syncs.topic.replication.factor=3

# The replication factor for connect internal topics "mm2-configs.B.internal", "mm2-offsets.B.internal" and
# "mm2-status.B.internal"
# 测试环境可以为1，生产环境建议以下配置大于1，比如设为3
offset.storage.replication.factor=3
status.storage.replication.factor=3
config.storage.replication.factor=3

# customize as needed
# replication.policy.separator = _
# sync.topic.acls.enabled = false
# emit.heartbeats.interval.seconds = 5
# 设置目标集群中的Topic名称和源集群相同
# replication.policy.class = org.apache.kafka.connect.mirror.IdentityReplicationPolicy
```

 版权所有 © 华为云计算技术有限公司

步骤6 在二进制软件包目录下，启动MirrorMaker，进行数据同步。

```
./bin/connect-mirror-maker.sh config/connect-mirror-maker.properties
```

步骤7 （可选）MirrorMaker开启后，如果在源集群上新建了Topic，如需对此Topic进行数据同步，则需重启MirrorMaker，重启步骤参考**步骤6**。也可配置自动同步新增Topic，按需增加如**表4-1**所示配置后，无需重启MirrorMaker，即可周期性同步新增Topic。其中，“refresh.topics.interval.seconds”为必选，其他参数根据实际情况选择。

表 4-1 MirrorMaker 配置参数

参数名	默认值	说明
sync.topic.configs.enabled	true	是否监控源集群的配置更改
sync.topic.acls.enabled	true	是否监控源集群ACL的更改
emit.heartbeats.enabled	true	连接器应定期发出心跳
emit.heartbeats.interval.seconds	5秒	心跳频率
emit.checkpoints.enabled	true	连接器应定期发出消费端偏移量信息
emit.checkpoints.interval.seconds	5秒	检查点的频率
refresh.topics.enabled	true	连接器应定期检查新主题
refresh.topics.interval.seconds	5秒	检查源集群中是否有新主题的频率
refresh.groups.enabled	true	连接器应定期检查新的消费组
refresh.groups.interval.seconds	5秒	检查源集群新的消费组频率
replication.policy.class	org.apache.kafka.connect.mirror.DefaultReplicationPolicy	使用LegacyReplicationPolicy模仿旧版MirrorMaker
heartbeats.topic.retention.ms	1天	首次创建心跳主题时使用
checkpoints.topic.retention.ms	1天	首次创建检查点主题时使用
offset.syncs.topic.retention.ms	max long	首次创建偏移同步主题时使用

----结束

验证数据是否同步

步骤1 在目标集群中查看Topic列表，确认是否有源集群Topic。

说明

“replication.policy.class”的默认值为“org.apache.kafka.connect.mirror.DefaultReplicationPolicy”，此时目标集群中的Topic名称和源集群相比，多了前缀（如A.），这是MirrorMaker为了防止Topic循环备份进行的设置。如果想要Topic名称保持一致，请将“replication.policy.class”设置为“org.apache.kafka.connect.mirror.IdentityReplicationPolicy”。

步骤2 在源集群生产并消费消息，在目标集群查看消费进度，确认数据是否已从源集群同步到了目标集群。

如果目标集群为华为云Kafka实例的话，在分布式消息服务Kafka版控制台的“实例管理 > 消费组管理 > 消费进度”中，查看消费进度。

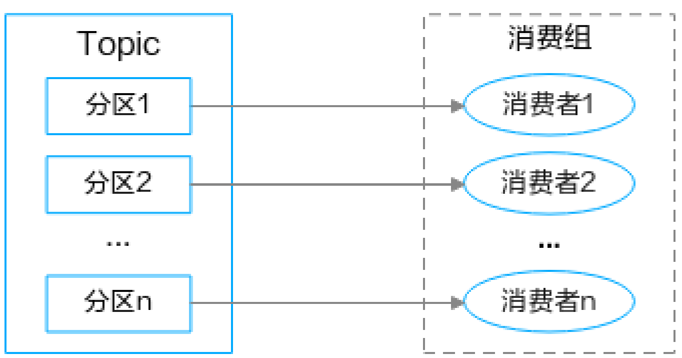
----结束

版权所有 © 华为云计算技术有限公司

5 消息堆积处理建议

方案概述

Kafka将Topic划分为多个分区，消息被分布式存储在分区中。同一个消费组内，一个消费者可同时消费多个分区，但一个分区在同一时刻只能被一个消费者消费。



在消息处理过程中，如果客户端的消费速度跟不上服务端的发送速度，未处理的消息会越来越多，这部分消息就被称为堆积消息。消息没有被及时消费就会产生消息堆积，从而会造成消息消费延迟。

消息堆积原因

导致消息堆积的常见原因如下：

- 生产者短时间内生产大量消息到Topic，消费者无法及时消费。
- 消费者的消费能力不足（消费者并发低、消息处理时间长），导致消费效率低于生产效率。
- 消费者异常（如消费者故障、消费者网络异常等）导致无法消费消息。
- Topic分区设置不合理，或新增分区无消费者消费。
- Topic频繁重分配导致消费效率降低。

实施步骤

从消息堆积的原因看，消息堆积问题可以从消费者端、生产者端和服务端三个方面进行处理。

- **消费者端**

  - 根据实际业务需求，合理增加消费者个数（消费并发度），确保分区数/消费者数=整数，建议消费者数和分区数保持一致。
  - 提高消费者的消费速度，通过优化消费者处理逻辑（减少复杂计算、第三方接口调用和读库操作），减少消费时间。
  - 增加消费者每次拉取消息的数量：拉取数据/处理时间 >= 生产速度。
- **生产者端**

  生产消息时，给消息Key加随机后缀，使消息均衡分布到不同分区上。

  说明

  在实际业务场景中，为消息Key加随机后缀，会导致消息全局不保序，需根据实际业务判断是否适合给消息Key加随机后缀。
- **服务端**
  - 合理设置Topic的分区数，在不影响业务处理效率的情况下，调大Topic的分区数量。
  - 当服务端出现消息堆积时，对生产者进行熔断，或将生产者的消息转发到其他Topic。

6 业务过载处理建议

方案概述

Kafka业务过载，一般表现为CPU使用率高、磁盘写满的现象。

- 当CPU使用率过高时，系统的运行速度会降低，并有加速硬件损坏的风险。
- 当磁盘写满时，相应磁盘上的Kafka日志目录会出现offline问题。此时，该磁盘上的分区副本不可读写，降低了分区的可用性和容错能力。同时由于Leader分区迁移到其他节点，会增加其他节点的负载。

CPU使用率高的原因

- 数据操作相关线程数（num.io.threads、num.network.threads、num.replica.fetchers）过多，导致CPU繁忙。
- 分区设置不合理，所有的生产和消费都集中在某个节点上，导致CPU利用率高。

磁盘写满的原因

- 业务数据增长较快，已有的磁盘空间不能满足业务数据需要。
- 节点内磁盘使用率不均衡，生产的消息集中在某个分区，导致分区所在的磁盘写满。
- Topic的数据老化时间设置过大，保存了过多的历史数据，容易导致磁盘写满。

实施步骤

CPU使用率高的处理措施：

- 优化线程参数num.io.threads、num.network.threads和num.replica.fetchers的配置。
  - num.io.threads和num.network.threads建议配置为磁盘个数的倍数，但不能超过CPU核数。
  - num.replica.fetchers建议配置不超过5。
- 合理设置Topic的分区，分区一般设置为节点个数的倍数。
- 生产消息时，给消息Key加随机后缀，使消息均衡分布到不同分区上。

说明

在实际业务场景中，为消息Key加随机后缀，会导致消息全局不保序，需根据实际业务判断是否适合给消息Key加随机后缀。

磁盘写满的处理措施：

- 扩容磁盘，使磁盘具备更大的存储空间。
- 迁移分区，将已写满的磁盘中的分区迁移到本节点的其他磁盘中。
- 合理设置Topic的数据老化时间，减少历史数据的容量大小。
- 在CPU资源情况可控的情况下，使用压缩算法对数据进行压缩。

  常用的压缩算法包括：ZIP，GZIP，SNAPPY，LZ4等。选择压缩算法时，需考虑数据的压缩率和压缩耗时。通常压缩率越高的算法，压缩耗时也越高。对于性能要求高的系统，可以选择压缩速度快的算法，比如LZ4；对于需要更高压缩比的系统，可以选择压缩率高的算法，比如GZIP。

  可以在生产者端配置“compression.type”参数来启用指定类型的压缩算法。

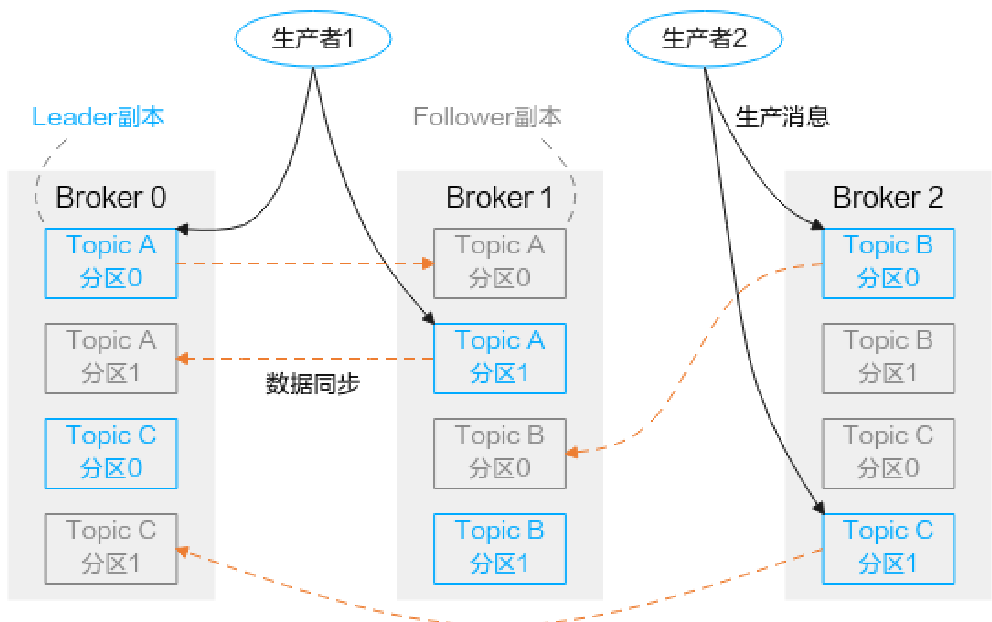
```
Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092");
props.put("acks", "all");
props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
// 开启GZIP压缩
props.put("compression.type", "gzip");

Producer<String, String> producer = new KafkaProducer<>(props);
```

 版权所有 © 华为云计算技术有限公司

7 业务数据不均衡处理建议

方案概述

Kafka将Topic划分为多个分区，所有消息分布式存储在各个分区上。每个分区有一个或多个副本，分布在不同的Broker节点上，每个副本存储一份全量数据，副本之间的消息数据保持同步。Kafka的Topic、分区、副本和代理的关系如下图所示：



在实际业务过程中可能会遇到各节点间或分区之间业务数据不均衡的情况，业务数据不均衡会降低Kafka集群的性能，降低资源使用率。

业务数据不均衡原因

- 业务中部分Topic的流量远大于其他Topic，会导致节点间的数据不均衡。
- 生产者发送消息时指定了分区，未指定的分区没有消息，会导致分区间的数据不均衡。
- 生产者发送消息时指定了消息Key，按照对应的Key发送消息至对应的分区，会导致分区间的数据不均衡。
- 系统重新实现了分区分配策略，但策略逻辑有问题，会导致分区间的数据不均衡。

 版权所有 © 华为云计算技术有限公司

- Kafka扩容了Broker节点，新增的节点没有分配分区，会导致节点间的数据不均衡。
- 业务使用过程中随着集群状态的变化，多少会发生一些Leader副本的切换或迁移，会导致个别Broker节点上的数据更多，从而导致节点间的数据不均衡。

实施步骤

业务数据不均衡的处理措施：

- 优化业务中Topic的设计，对于数据量特别大的Topic，可对业务数据做进一步的细分，并分配到不同的Topic上。
- 生产者生产消息时，尽量把消息均衡发送到不同的分区上，确保分区间的数据均衡。
- 创建Topic时，使分区的Leader副本分散到各个Broker节点中，以保障整体的数据均衡。
- Kafka提供了分区平衡的功能，可以把分区的副本重新分配到不同的Broker节点上，解决节点间负载不均衡的问题。具体分区平衡的操作请参考修改分区平衡。

8 Kafka Topic 分区数设置建议

方案概述

Topic分区数是影响Kafka系统性能、吞吐量以及并行度的关键配置之一：分区数过少，会限制生产者的并发能力，还可能导致消费者出现空闲现象；分区数过多，则可能造成集群性能下降、监控统计间隔增长等问题。因此，为Topic设置合理的分区数至关重要。

实施步骤

分区数的设置需结合集群规模、消费者数量及扩展性综合考量，以下针对这些因素提供具体设置建议：

表 8-1 分区数设置建议

考量因素	配置建议
集群规模	分区数建议设置为**Broker数量的整数倍**。 若设置不合理，可能导致各Broker上的分区数量不均，进而造成集群负载不均衡。
消费者数量	分区数建议设置为**消费者数量，或消费者数量的整数倍**。 一个分区只能由一个消费者消费，当分区数小于消费者数量时，会导致部分消费者空闲。
扩展性	针对业务流量变化大的业务，分区数建议设置为**业务高峰期的消费者数量**。 临时新增分区难以快速解决已有分区的消息堆积问题，因此建议提前将分区数设置为业务高峰期的消费者数量。

版权所有 © 华为云计算技术有限公司

典型问题案例

表 8-2 典型问题案例

问题现象	根因分析	处理建议
使用ClickHouse消费Kafka，其中Topic分区数为12，ClickHouse节点数为10，ClickHouse表参数中已配置consumer_num=12（即10个节点共有120个消费者），问题现象为仅1个ClickHouse节点在消费，其余节点均处于空闲状态。	正在消费消息的12个消费者通过重平衡分配到了Topic的全部分区，且这些消费者均集中在同一个ClickHouse节点上，导致其他节点上的消费者无消息可消费。	分区数应大于或等于消费者数量，且建议配置为消费者数量的整数倍，以确保每个消费者能分配到等量的分区，从而实现各消费者的负载均衡。
Topic分区数为3，随着业务流量增长将Kafka实例从3个Broker扩容至6个Broker，问题现象为新扩容的Broker上没有分区，业务仍集中在原有节点上。	扩容后未增加分区数量，且未执行分区平衡，导致分区数仍分布在原有节点上。	分区数建议设置为Broker数量的整数倍，然后执行分区平衡，确保分区能够均衡的分布在每个Broker上。

版权所有 © 华为云计算技术有限公司

9 配置消息堆积数监控

方案概述

在消息处理过程中，如果客户端的消费速度跟不上服务端的发送速度，未处理的消息会越来越多，这部分消息就被称为堆积消息。消息没有被及时消费就会产生消息堆积，从而会造成消息消费延迟。

如果您想要在消费组的消息堆积数超过阈值时，通过短信/邮件及时收到通知信息，可以参考本章节设置告警通知。您还可以参考本章节为分布式消息服务Kafka版的**其他监控指标**设置告警通知。

前提条件

已**购买Kafka实例**、**创建Topic**，并且已成功消费消息。

实施步骤

步骤1 登录**Kafka控制台**。

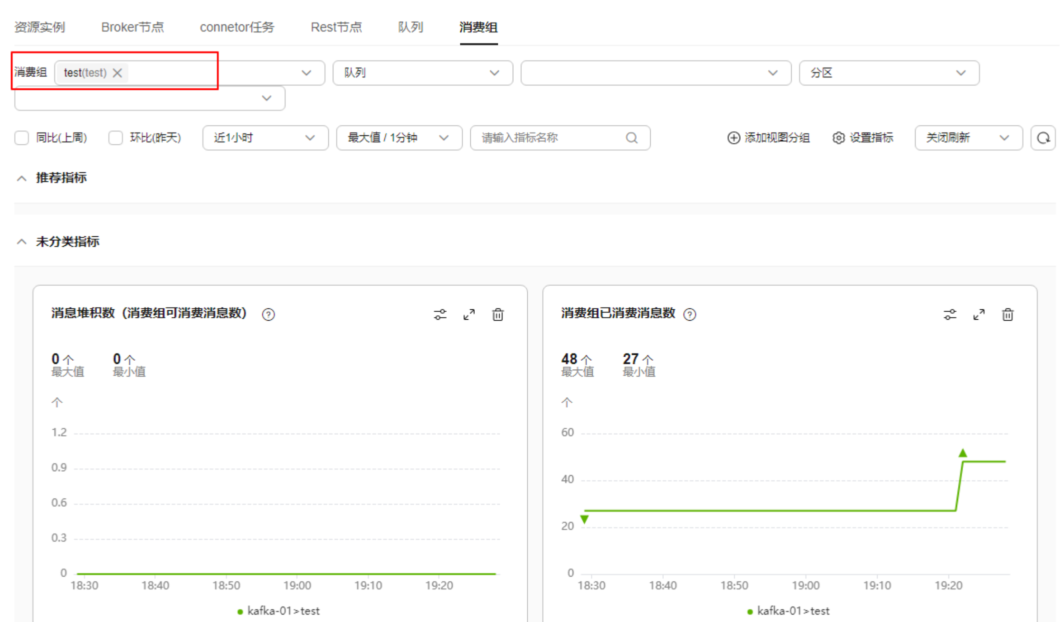
步骤2 单击待创建告警通知的实例名称，进入实例详情页。

步骤3 在左侧导航栏，选择“监控 > 监控详情”，进入监控详情页面。

步骤4 在“消费组”页签的“消费组”下拉框中，选择需要创建告警通知的消费组。

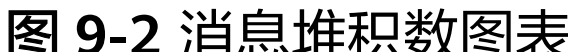
版权所有 © 华为云计算技术有限公司

图 9-1 选择需要创建告警通知的消费组



步骤5 在“消息堆积数（消费组可消费消息数）”图表上单击，创建告警规则。

图 9-2 消息堆积数图表



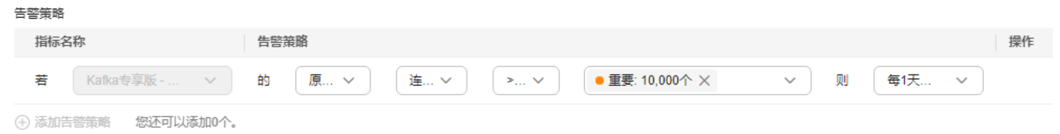
步骤6 在“创建告警规则”页面，设置告警名称。告警名称需要符合命名规则：只能由中文、英文字母、数字、下划线、中划线组成。

图 9-3 设置告警名称

步骤7 在“创建告警规则”页面，设置监控范围。保持当前设置，无需修改。

步骤8 在“创建告警规则”页面，设置告警策略。

图 9-4 设置告警策略



告警策略：连续1次原始值>=10000个时，触发重要告警，每天发送一次告警通知。

步骤9 在“创建告警规则”页面，单击“创建通知策略”，进入“创建通知策略”页面。

步骤10 设置通知策略，单击“确定”。

图 9-5 创建通知策略

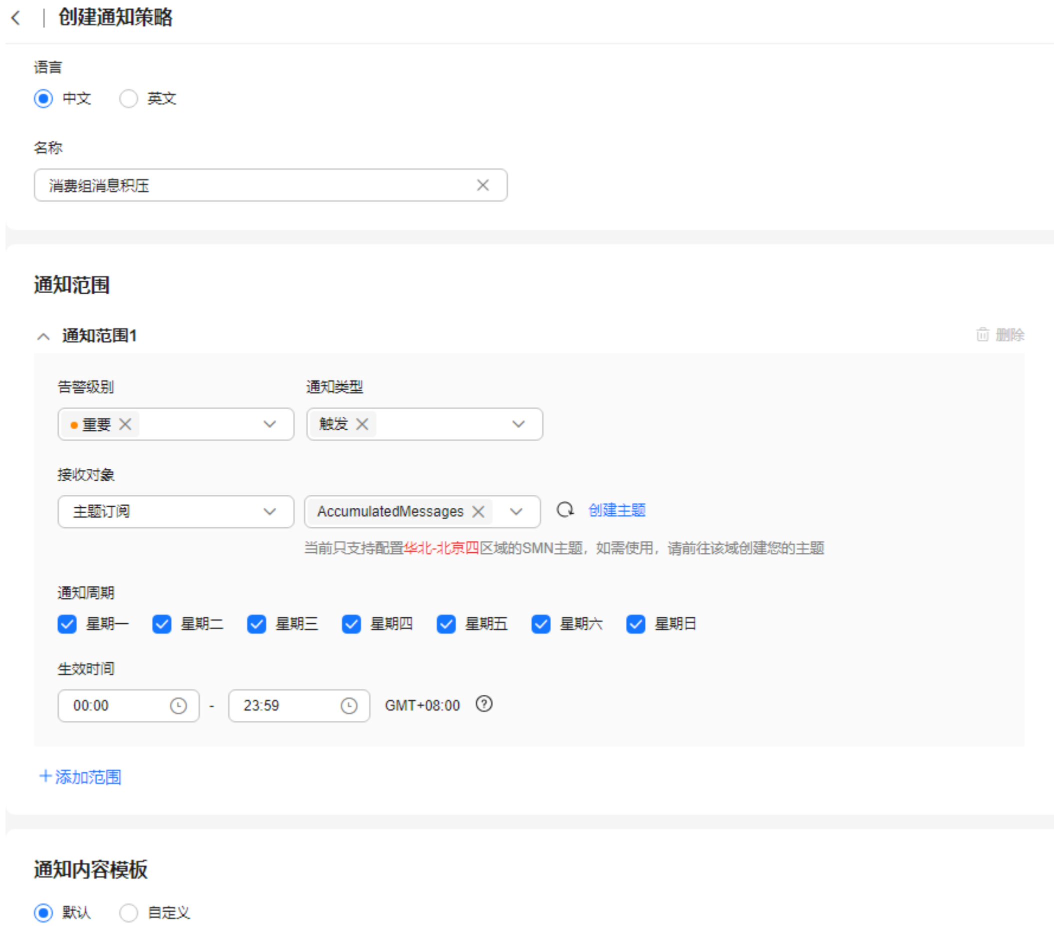


表 9-1 通知策略参数说明

参数名称	说明
语言	选择通知策略的语言。
名称	设置通知策略的名称。

 版权所有 © 华为云计算技术有限公司

参数名称	说明
告警级别	选择“重要”。
通知类型	选择“触发”，即触发告警时发送通知。
接收对象	选择“主题订阅”，单击“创建主题”，进入消息通知服务中，创建主题和添加订阅。创建完成后，返回“创建告警规则”页面，在“通知对象”后单击C，然后选择创建的告警通知主题。 **说明** 在添加订阅后，对应的订阅终端会收到订阅通知，用户要选择确认订阅，后续才能收到告警信息。
通知周期	保持默认，即如果触发告警，每天发送告警通知。
生效时间	告警规则仅在生效时间内发送通知消息，保持默认。
通知内容模板	选择“默认”。

步骤11 通知策略创建成功后，返回“创建告警规则”页面。

步骤12 在“通知策略”后，单击，然后在下拉框中选择新创建的通知策略。

图 9-6 设置通知策略

步骤13 单击“确定”，完成告警规则的设置。

告警规则创建完成后，在云监控服务的“告警 > 告警规则”界面，查看新创建的告警规则。

图 9-7 查看新创建的告警规则

----结束

版权所有 © 华为云计算技术有限公司

10 DMS for Kafka 安全使用建议

安全性是华为云与您的共同责任。华为云负责云服务自身的安全，提供安全的云；作为租户，您需要合理使用云服务提供的安全能力对数据进行保护，安全地使用云。详情请参见责任共担。

本文提供了使用DMS for Kafka过程中的安全最佳实践，旨在为提高整体安全能力提供可操作的规范性指导。根据该指导文档您可以持续评估DMS for Kafka资源的安全状态，更好的组合使用DMS for Kafka提供的多种安全能力，提高对DMS for Kafka资源的整体安全防御能力，保护存储在DMS for Kafka内的数据不泄露、不被篡改，以及数据在传输过程中不泄露、不被篡改。

本文从以下几个维度给出建议，您可以评估DMS for Kafka的使用情况，并根据业务需要在本指导的基础上进行安全配置。

- 通过访问控制，保护数据安全性
- **SSL链路传输加密方式访问DMS for Kafka**
- 不存储敏感数据
- 构建数据的恢复和容灾能力
- 审计是否存在异常数据访问
- 使用最新版本SDK获得更好的操作体验和更强的安全能力

通过访问控制，保护数据安全性

1. **建议对不同角色的IAM用户仅设置最小权限，避免权限过大导致数据泄露或被误操作。**

   为了更好的进行权限隔离和管理，建议您配置独立的IAM管理员，授予IAM管理员IAM策略的管理权限。IAM管理员可以根据您业务的实际诉求创建不同的用户组，用户组对应不同的数据访问场景，通过将用户添加到用户组并将IAM策略绑定到对应用户组，IAM管理员可以为不同职能部门的员工按照最小权限原则授予不同的数据访问权限，详情请参见权限管理。

2. **建议配置安全组访问控制，保护您的数据不被异常读取和操作。**

   租户配置安全组的入方向、出方向规则限制，可以控制连接实例的网络范围，避免DMS for Kafka暴露给不可信第三方，详情请参见配置安全组。安全组入方向规则的“源地址”应避免设置为0.0.0.0/0。

3. **建议将访问Kafka实例方式设置为密码访问（即开启SASL），防止未经认证的客户端误操作实例。**

4. **开启敏感操作多因子认证保护您的数据不被误删。**

   DMS for Kafka支持敏感操作保护，开启后执行删除实例等敏感操作时，系统会进行身份验证，进一步对数据的高危操作进行控制，保证DMS for Kafka数据的安全性。详情请参见**敏感操作**。

SSL 链路传输加密方式访问 DMS for Kafka

为了确保数据传输过程中不被窃取和破坏，建议使用SSL链路传输加密方式（即Kafka安全协议设置为“SASL_SSL”）访问DMS for Kafka。

不存储敏感数据

DMS for Kafka暂不支持数据加密，建议不要将敏感数据存入消息队列。

构建数据的恢复和容灾能力

预先构建数据的容灾和恢复能力，可以有效避免异常数据处理场景下数据误删、破坏的问题。

1. **建议Topic配置多副本，获得异常场景数据快速恢复能力。**

   Kafka实例支持配置Topic副本数量，配置多副本后Kafka实例会主动建立和维护同步复制，在实例某个broker故障的情况下，实例会自动将该节点上分区leader切换到其它可用的broker上，从而达到高可用的目的。

2. **建议使用多个可用区构建数据容灾能力。**

   Kafka集群实例支持跨可用区部署，支持跨可用区容灾。如果创建实例时选择了多个可用区，当一个可用区异常时，不影响Kafka实例持续提供服务。

审计是否存在异常数据访问

1. **开启云审计服务，记录Kafka的所有访问操作，便于事后审查。**

   云审计服务（Cloud Trace Service，CTS），是华为云安全解决方案中专业的日志审计服务，提供对各种云资源操作记录的收集、存储和查询功能，可用于支撑安全分析、合规审计、资源跟踪和问题定位等常见应用场景。

   您开通云审计服务并创建和配置追踪器后，CTS可记录Kafka的管理事件和数据事件用于审计。详情请参见**查看Kafka审计日志**。

2. **使用云监控服务对Kafka进行实时监控和告警。**

   为使您更好地掌握Kafka实例状态，华为云提供了云监控服务（Cloud Eye）。您可使用该服务监控自己的Kafka实例，执行自动实时监控、告警和通知操作，帮助您实时掌握Kafka实例中所产生的请求、流量等信息。

   云监控服务不需要开通，会在您创建Kafka实例后自动启动。相关文档请参见**Kafka支持的监控指标**。

使用最新版本 SDK 获得更好的操作体验和更强的安全能力

建议您升级SDK并使用最新版本，从客户侧对您的数据和Kafka使用过程提供更好的保护。最新版本SDK在各语言对应界面下载，请参见**SDK概述**。

11 优化消费者轮询（Polling）

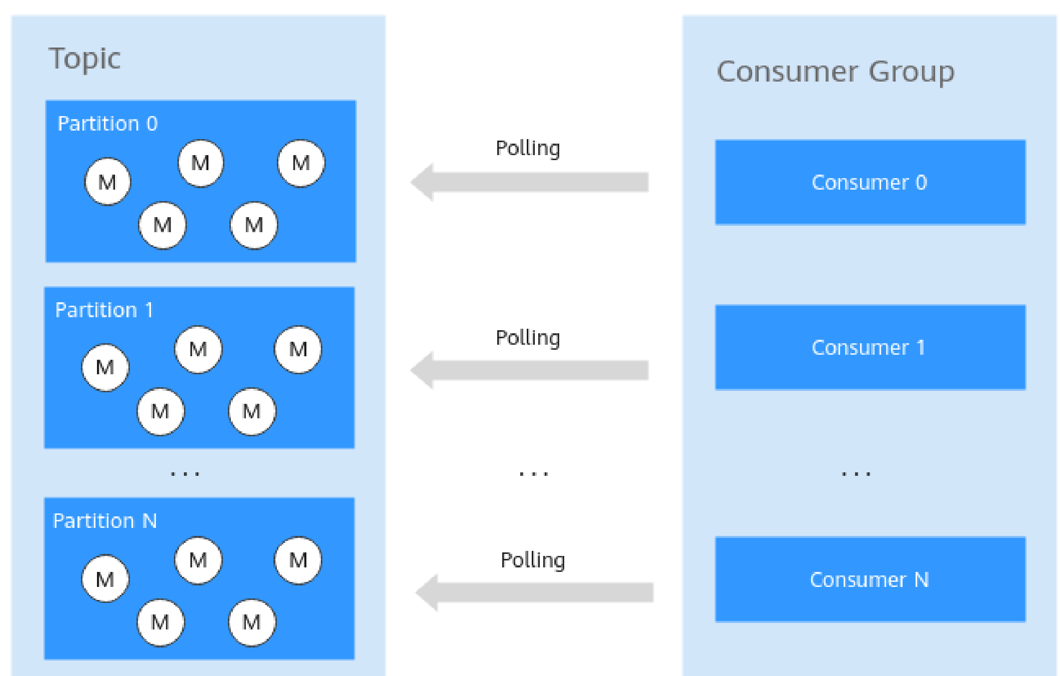
方案概述

应用场景

在分布式消息服务Kafka版提供的原生Kafka SDK中，消费者可以自定义拉取消息的时长，如果需要长时间的拉取消息，只需要把poll(long)方法的参数设置合适的值即可。但是这样的长连接可能会对客户端和服务端造成一定的压力，特别是分区数较多且每个消费者开启多个线程的情况下。

如图11-1所示，Topic含有多个分区，消费组中有多个消费者同时进行消费，每个线程均为长连接。当Topic中消息较少或者没有消息时，连接不断开，所有消费者不间断地拉取消息，这样造成了一定的资源浪费。

图 11-1 Kafka 消费者多线程消费模式



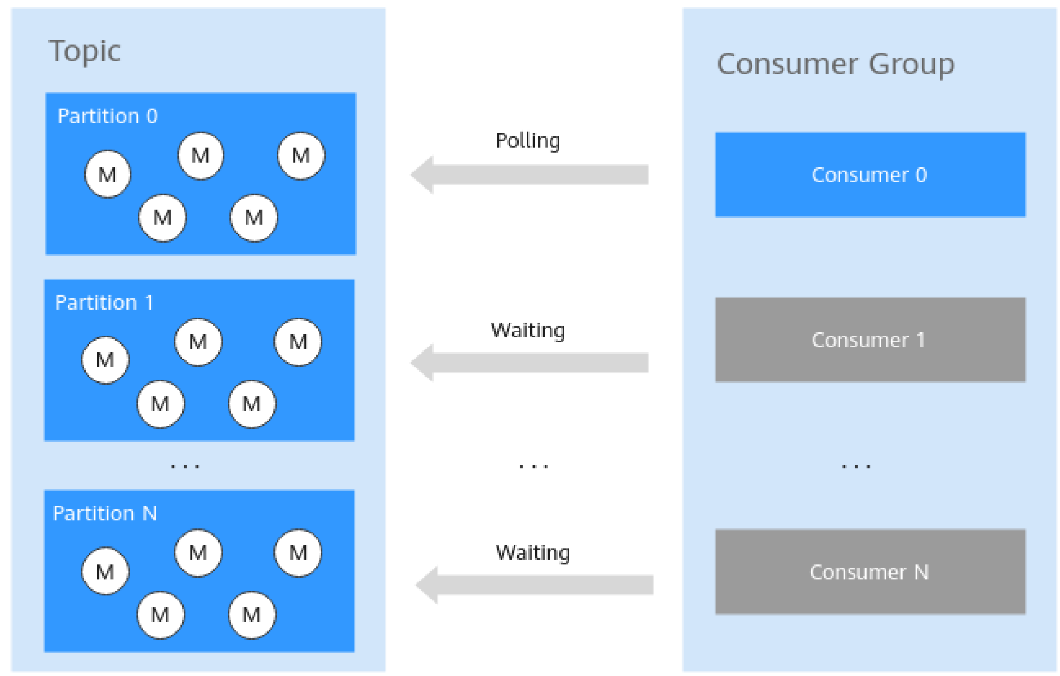
解决方案

在开了多个线程同时访问的情况下，如果Topic里已经没有消息了，其实不需要所有的线程都在poll，只需要有一个线程poll各分区的消息就足够了，当在polling的线程发现

Topic中有消息，可以唤醒其他线程一起消费消息，以达到快速响应的目的。如图11-2所示。

这种方案适用于对消费消息的实时性要求不高的应用场景。如果要求准实时消费消息，则建议保持所有消费者处于活跃状态。

图 11-2 优化后的多线程消费方案



说明

消费者（Consumer）和消息分区（Partition）并不强制数量相等，Kafka的poll(long)方法帮助实现获取消息、分区平衡、消费者与Kafka broker节点间的心跳检测等功能。

因此在对消费消息的实时性要求不高场景下，当消息数量不多的时候，可以选择让一部分消费者处于wait状态。

代码示例

须知

以下仅贴出与消费者线程唤醒与睡眠相关代码，如需运行整个demo，请先下载完整的代码示例包，同时参考开发指南进行部署和运行。

- **消费消息代码示例：**

```
package com.huawei.dms.kafka;

import java.io.IOException;
import java.util.Arrays;
import java.util.Collection;
import java.util.Iterator;
import java.util.Properties;

import org.apache.kafka.clients.consumer.ConsumerRebalanceListener;
import org.apache.kafka.clients.consumer.ConsumerRecord;
```

```
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.common.TopicPartition;
import org.apache.log4j.Logger;

public class DmsKafkaConsumeDemo
{
    private static Logger logger = Logger.getLogger(DmsKafkaProduceDemo.class);

    public static void WorkerFunc(int workerId, KafkaConsumer<String, String> kafkaConsumer)
throws IOException
    {
        Properties consumerConfig = Config.getConsumerConfig();
        RecordReceiver receiver = new RecordReceiver(workerId, kafkaConsumer,
consumerConfig.getProperty("topic"));
        while (true)
        {
            ConsumerRecords<String, String> records = receiver.receiveMessage();
            Iterator<ConsumerRecord<String, String>> iter = records.iterator();
            while (iter.hasNext())
            {
                ConsumerRecord<String, String> cr = iter.next();
                System.out.println("Thread" + workerId + " recievedrecords" + cr.value());
                logger.info("Thread" + workerId + " recievedrecords" + cr.value());

            }

        }
    }

    public static KafkaConsumer<String, String> getConsumer() throws IOException
    {
        Properties consumerConfig = Config.getConsumerConfig();

        consumerConfig.put("ssl.truststore.location", Config.getTrustStorePath());
        System.setProperty("java.security.auth.login.config", Config.getSaslConfig());

        KafkaConsumer<String, String> kafkaConsumer = new KafkaConsumer<>(consumerConfig);
        kafkaConsumer.subscribe(Arrays.asList(consumerConfig.getProperty("topic")),
                new ConsumerRebalanceListener()
                {
                    @Override
                    public void onPartitionsRevoked(Collection<TopicPartition> arg0)
                    {

                    }

                    @Override
                    public void onPartitionsAssigned(Collection<TopicPartition> tps)
                    {

                    }
                });
        return kafkaConsumer;
    }

    public static void main(String[] args) throws IOException
    {

        //创建当前消费组的consumer
        final KafkaConsumer<String, String> consumer1 = getConsumer();
        Thread thread1 = new Thread(new Runnable()
        {
            public void run()
            {
                try
                {
                    WorkerFunc(1, consumer1);
                }
```


版权所有 © 华为云计算技术有限公司

```
                catch (IOException e)
                {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        });
        final KafkaConsumer<String, String> consumer2 = getConsumer();

        Thread thread2 = new Thread(new Runnable()
        {
            public void run()
            {
                try
                {
                    WorkerFunc(2, consumer2);
                }
                catch (IOException e)
                {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        });
        final KafkaConsumer<String, String> consumer3 = getConsumer();

        Thread thread3 = new Thread(new Runnable()
        {
            public void run()
            {
                try
                {
                    WorkerFunc(3, consumer3);
                }
                catch (IOException e)
                {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        });

        //启动线程
        thread1.start();
        thread2.start();
        thread3.start();

        try
        {
            Thread.sleep(5000);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
        //线程加入
        try
        {
            thread1.join();
            thread2.join();
            thread3.join();
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
    }
}
```

- **消费者线程管理代码示例：**

  示例仅提供简单的设计思路，开发者可结合实际场景优化线程休眠和唤醒机制。

  说明

  topicName配置为Topic名称。

```
package com.huawei.dms.kafka;

import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;

import org.apache.log4j.Logger;

public class RecordReceiver
{
    private static Logger logger = Logger.getLogger(DmsKafkaProduceDemo.class);

    //polling的间隔时间
    public static final int WAIT_SECONDS = 10 * 1000;

    protected static final Map<String, Object> sLockObjMap = new HashMap<String, Object>();

    protected static Map<String, Boolean> sPollingMap = new ConcurrentHashMap<String,
Boolean>();

    protected Object lockObj;

    protected String topicName;

    protected KafkaConsumer<String, String> kafkaConsumer;

    protected int workerId;

    public RecordReceiver(int id, KafkaConsumer<String, String> kafkaConsumer, String queue)
    {
        this.kafkaConsumer = kafkaConsumer;
        this.topicName = queue;
        this.workerId = id;

        synchronized (sLockObjMap)
        {
            lockObj = sLockObjMap.get(topicName);
            if (lockObj == null)
            {
                lockObj = new Object();
                sLockObjMap.put(topicName, lockObj);
            }
        }
    }

    public boolean setPolling()
    {
        synchronized (lockObj)
        {
            Boolean ret = sPollingMap.get(topicName);
            if (ret == null || !ret)
            {
                sPollingMap.put(topicName, true);
                return true;
            }
            return false;
        }
    }
```

```
//唤醒全部线程
public void clearPolling()
{
    synchronized (lockObj)
    {
        sPollingMap.put(topicName, false);
        lockObj.notifyAll();
        System.out.println("Everyone WakeUp and Work!");
        logger.info("Everyone WakeUp and Work!");
    }
}

public ConsumerRecords<String, String> receiveMessage()
{
    boolean polling = false;
    while (true)
    {
        //检查线程的poll状态，必要时休眠
        synchronized (lockObj)
        {
            Boolean p = sPollingMap.get(topicName);
            if (p != null && p)
            {
                try
                {
                    System.out.println("Thread" + workerId + " Have a nice sleep!");
                    logger.info("Thread" + workerId +" Have a nice sleep!");
                    polling = false;
                    lockObj.wait();
                }
                catch (InterruptedException e)
                {
                    System.out.println("MessageReceiver Interrupted! topicName is " + topicName);
                    logger.error("MessageReceiver Interrupted! topicName is "+topicName);

                    return null;
                }
            }
        }

        //开始消费，必要时唤醒其他线程消费
        try
        {
            ConsumerRecords<String, String> Records = null;
            if (!polling)
            {
                Records = kafkaConsumer.poll(100);
                if (Records.count() == 0)
                {
                    polling = true;
                    continue;
                }
            }
            else
            {
                if (setPolling())
                {
                    System.out.println("Thread" + workerId + " Polling!");
                    logger.info("Thread " + workerId + " Polling!");
                }
                else
                {
                    continue;
                }
                do
                {
                    System.out.println("Thread" + workerId + " KEEP Poll records!");
                    logger.info("Thread" + workerId + " KEEP Poll records!");
                    try
```


版权所有 © 华为云计算技术有限公司

```
                        {
                            Records = kafkaConsumer.poll(WAIT_SECONDS);
                        }
                        catch (Exception e)
                        {
                            System.out.println("Exception Happened when polling records: " + e);
                            logger.error("Exception Happened when polling records: " + e);

                        }
                    } while (Records.count()==0);
                    clearPolling();
                }
                //消息确认
                kafkaConsumer.commitSync();
                return Records;
            }
            catch (Exception e)
            {
                System.out.println("Exception Happened when poll records: " + e);
                logger.error("Exception Happened when poll records: " + e);
            }
        }
    }
}
```

代码示例运行结果

```
[2018-01-25 22:40:51,841] INFO Thread 2 Polling! (com.huawei.dms.kafka.DmsKafkaProduceDemo:119)
[2018-01-25 22:40:51,841] INFO Thread2 KEEP Poll records!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:128)
[2018-01-25 22:40:52,122] INFO Everyone WakeUp and Work!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:69)
[2018-01-25 22:40:52,169] INFO Thread2 recievedrecordshello, dms kafka.
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
[2018-01-25 22:40:52,169] INFO Thread2 recievedrecordshello, dms kafka.
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
[2018-01-25 22:40:52,216] INFO Thread2 recievedrecordshello, dms kafka.
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
[2018-01-25 22:40:52,325] INFO Thread 2 Polling! (com.huawei.dms.kafka.DmsKafkaProduceDemo:119)
[2018-01-25 22:40:52,325] INFO Thread2 KEEP Poll records!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:128)
[2018-01-25 22:40:54,947] INFO Thread1 Have a nice sleep!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:87)
[2018-01-25 22:40:54,979] INFO Thread3 Have a nice sleep!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:87)
[2018-01-25 22:41:32,347] INFO Thread2 KEEP Poll records!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:128)
[2018-01-25 22:41:42,353] INFO Thread2 KEEP Poll records!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:128)
[2018-01-25 22:41:47,816] INFO Everyone WakeUp and Work!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:69)
[2018-01-25 22:41:47,847] INFO Thread2 recievedrecordshello, dms kafka.
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
[2018-01-25 22:41:47,925] INFO Thread 3 Polling! (com.huawei.dms.kafka.DmsKafkaProduceDemo:119)
[2018-01-25 22:41:47,925] INFO Thread1 Have a nice sleep!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:87)
[2018-01-25 22:41:47,925] INFO Thread3 KEEP Poll records!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:128)
[2018-01-25 22:41:47,957] INFO Thread2 Have a nice sleep!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:87)
[2018-01-25 22:41:48,472] INFO Everyone WakeUp and Work!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:69)
[2018-01-25 22:41:48,503] INFO Thread3 recievedrecordshello, dms kafka.
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
[2018-01-25 22:41:48,518] INFO Thread1 recievedrecordshello, dms kafka.
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
[2018-01-25 22:41:48,550] INFO Thread2 recievedrecordshello, dms kafka.
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
[2018-01-25 22:41:48,597] INFO Thread1 recievedrecordshello, dms kafka.
```

```
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
[2018-01-25 22:41:48,659] INFO Thread 2 Polling! (com.huawei.dms.kafka.DmsKafkaProduceDemo:119)
[2018-01-25 22:41:48,659] INFO Thread2 KEEP Poll records!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:128)
[2018-01-25 22:41:48,675] INFO Thread3 recievedrecordshello, dms kafka.
(com.huawei.dms.kafka.DmsKafkaProduceDemo:32)
[2018-01-25 22:41:48,675] INFO Everyone WakeUp and Work!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:69)
[2018-01-25 22:41:48,706] INFO Thread 1 Polling! (com.huawei.dms.kafka.DmsKafkaProduceDemo:119)
[2018-01-25 22:41:48,706] INFO Thread1 KEEP Poll records!
(com.huawei.dms.kafka.DmsKafkaProduceDemo:128)
```