1.生产者概述

向kafka发送消息的步骤：

a) 创建ProducerRecord对象，该对象主要包含目标主题和要发送的内容，

b) 如果ProducerRecord中未指定分区，分区器会通过ProducerRecord对象的键选择分区

c) 数据加入到批次中，该批次被发送到相同的主题和分区

d) 服务器返回RecordMetaData对象，该对象包括主题和分区信息以及记录在分区中的偏移量，写入失败就返回错误

2.创建生产者

2.1 3个必须写的属性 bootstrap.servers key.serializer value.serializer

2.2 3种发送方式

a)发送并忘记

只是发送出去，并不关心是否到达

b)同步发送

使用send方法发送，会返回一个Future对象，调用get()方法等待

```
producer.send(record).get()
```

c)异步发送

调用send()方法，并指定一个回调函数，服务器在返回响应时调用该函数

```
class DemoProducerCallBack implements Callback {
public void onCompletion(RecordMetadata recordMetadata, Exception e) {
if (e != null) {
e.printStackTrace();}
}
}
producer.send(record,new DemoProducerCallBack());
```

2.3代码相关

在github的kafka-study上

4.生产者配置

4.1ack:指定了要有多少分区接收到数据,producer才认为消息写入是成功的

ack=0 不确认分区消息，直接返回

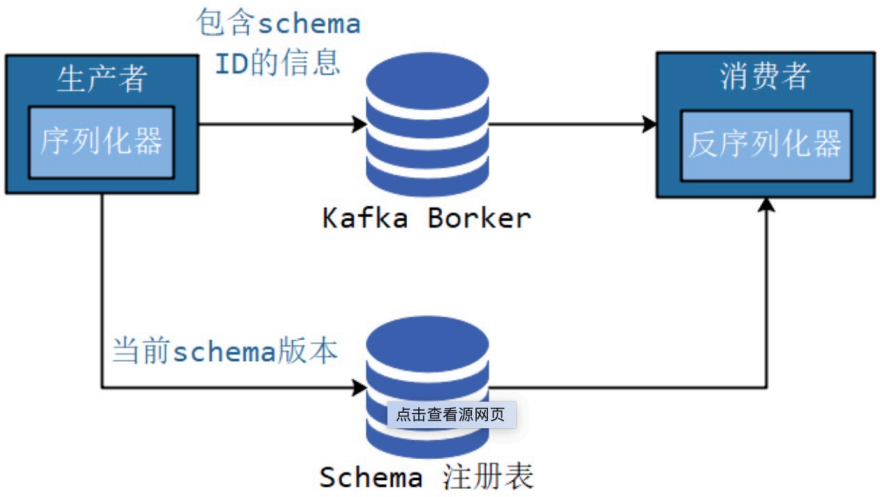
ack=1 集群的首领接受到消息

ack=all 所有节点都要收到消息，延迟比ack=1高很多

4.2 buffer.momory：生产者缓存区的大小
4.3 compression.type 消息压缩,默认情况不压缩
snappy/gzip/lz4
4.4 retries 重试次数
4.5 batch.size 该参数指定了一个批次可以使用的内存大小
4.6 max.in.flight.requests.per.connection：生产者接受到服务器响应前可以发送多少消息
顺序保证：如果retries不为0，且max.in.flight.requests.per.connection比1大，那么可能出现消息批次反过来，所以如果对消息顺序又要求，那么应该把max.in.flight.requests.per.connection设置为1

5.序列化器
5.1 自定义序列化器
5.2Avro序列化
Apache Avro（以下简称 Avro）是一种与编程语言无关的序列化格式。Doug Cutting 创建了这个项目，目的是提供一种共享数据文件的方式。
Avro 数据通过与语言无关的 schema 来定义。schema 通过 JSON 来描述，数据被序列化成二进制文件或 JSON 文件，不过一般会使用二进制文件。Avro 在读写文件时需要用到 schema，schema 一般会被内嵌在数据文件里。
Avro 有一个很有意思的特性是，当负责写消息的应用程序使用了新的schema['ski:mə]，负责读消息的应用程序可以继续处理消息而无需做任何改动，其中被删除的字段读取到的是null
schema注册表推荐使用confluent schema registry

5.3 Kafka中使用Avro
1)将所有写入数据需要的schema保持到注册表，在Kafka记录中引用schema标志符
2)负责读取数据的应用程序使用标志符从注册表里拉取schema来反序列化记录
3)序列化器和反序列化器分别负责处理schema的注册和拉取



代码实现：

方式一：使用Avro生成的对象

方式二：

使用一般的Avro对象而非生成的Avro，需要自己提供schema

6.分区

ProducerRecord中的键，一个作用是作为消息的附加，一个作用是决定消息写到哪个分区，相同的键分到相同的分区

1）key为null+默认分区器，分区器采用轮询(Round Robin)算法均匀的分布消息

2）key不为null+默认分区，kafka对键进行散列操作，然后根据散列值把消息映射到特定分区。意味着同一个键总是到同一个分区

只有不改变主题分区和数量的情况下，键与分区的映射才能不变。

自定义分区策略

partitioner接口包括configure,partition和close，这里只定义partition

```
import org.apache.kafka.clients.producer.Partitioner;
import org.apache.kafka.common.Cluster;
import org.apache.kafka.common.PartitionInfo;
import org.apache.kafka.common.record.InvalidRecordException;
```

```
import org.apache.kafka.common.utils.Utils;

import java.util.List;

public class BananaPartitioner implements Partitioner {
    public void configure(Map<String, ?> configs) {
    }

    public int partition(String topic, Object key, byte[] keyBytes, Object value, byte[]
valueBytes, Cluster cluster) {
        List<PartitionInfo> partitions = cluster.partitionsForTopic(topic);
        int numPartitions = partitions.size();
        if ((keyBytes == null) || (!(key instanceOf String)))
        throw new InvalidRecordException("We expect all messages to have customer
name as key")
        if (((String) key).equals("Banana"))
            return numPartitions - 1; // Banana总是被分配到最后一个分区
        // Other records will get hashed to the rest of the partitions
        return (Math.abs(Utils.murmur2(keyBytes)) % numPartitions)
    }

    public void close() {
    }
}
```

使用分区器：

Producer中，

```
props.put("partitioner.class",
"com.bonc.rdpe.kafka110.partitioner.PhonenumPartitioner");
```