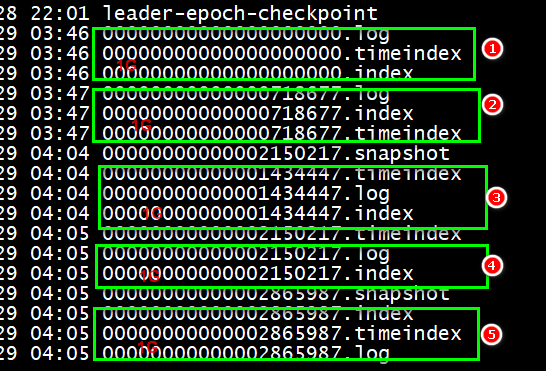
Kafka具有存储功能，默认保存数据时间为7天，也就是说kafka broker上的数据超7天或者1G，就会被清理掉。这些数据存放在broker[服务器](https://www.baidu.com/s?wd=%E6%9C%8D%E5%8A%A1%E5%99%A8&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)上，以log文件的形式存在，当log文件大于1G时会重新写入另一个segment中，1G大约50W条数据。

log的路径配置在conf/server.properties配置文件中，我的日志路径配置在log1文件夹下。log文件的命名那一长串0，是这个日志文件的offset位置。当日志文件达到时间或者大小的上限时，就会生成下一个日志文件，命名的就是下一个offset位置了。

log日志文件是二进制文件，无法通过文本查看，但是可以通过kafka.tools.DumpLogSegments类的方法，可以查看日志的内容。

另外还有两个文件，index文件存放的是topic的offset，timeindex是存放的是时间戳



bin/kafka-run-class.sh kafka.tools.DumpLogSegments --files ./logs1/demo1-0/00000000000000000000.log --print-data-log

kafka是scala语言编写的

创建topic

bin/kafka-topics.sh --create --zookeeper zk01:2181 --replication-factor 1 --partitions 1 --topic order

创建生产者开始生产数据

bin/kafka-console-producer.sh --broker-list kafka01:9092 --topic order

创建消费者消费数据

bin/kafka-console-consumer.sh --zookeeper zk01:2181 --from-beginning --topic order

consumer的offset是怎么存储的呢？consumer有两种消息方式，一种是存放在broker的日志目录中，另一种方式是存放在zookeeper中。两种存放方式和你使用kafka-console-consumer命令使用的选项有关。如果使用的是bootstrap-server，那么就存放在broker，broker存放offset是kafka从0.9版本开始提供的新的消费方式。；如果使用的是–zookeeper那么就存放在zookeeper。

消费的是分区0中的消息，offset从5开始。

kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic demo1 --partition 0 --offset 5

打印所有的消费组名称

kafka-consumer-groups.sh --bootstrap-server localhost:9092 --list

kafka-consumer-groups.sh --zookeeper localhost:2181/kafka1 --list

查看明细消费信息

kafka-consumer-groups.sh --bootstrap-server localhost:9092 --desc --group console-consumer-33936

kafka-consumer-groups.sh --zookeeper localhost:2181/kafka1 --desc --group console-consumer-69524

1.java工程-maven，依赖。

|  |
| --- |
| <**dependency**>  <**groupId**>org.apache.kafka</**groupId**>  <**artifactId**>kafka-clients</**artifactId**>  <**version**>0.11.0.1</**version**> </**dependency**> |

2.编写代码-写生产者的代码

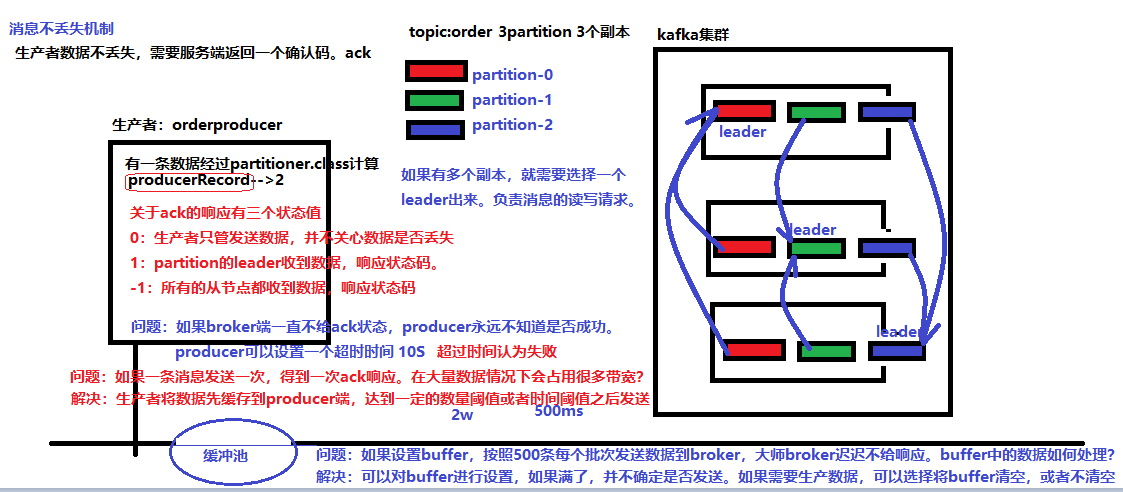
|  |
| --- |
| */\*\*  \* 订单的生产者代码  \*/* **public class** OrderProducer {  **public static void** main(String[] args) **throws** InterruptedException {  */\* 1、连接集群，通过配置文件的方式  \* 2、发送数据-topic:order，value  \*/* Properties props = **new** Properties();  props.put(**"bootstrap.servers"**, **"node01:9092"**);  props.put(**"acks"**, **"all"**);  props.put(**"retries"**, 0);  props.put(**"batch.size"**, 16384);  props.put(**"linger.ms"**, 1);  props.put(**"buffer.memory"**, 33554432);  props.put(**"key.serializer"**,  **"org.apache.kafka.common.serialization.StringSerializer"**);  props.put(**"value.serializer"**,  **"org.apache.kafka.common.serialization.StringSerializer"**);  KafkaProducer<String, String> kafkaProducer = **new** KafkaProducer<String, String>(props);  **for** (**int** i = 0; i < 1000; i++) {  *// 发送数据 ,需要一个producerRecord对象,最少参数 String topic, V value* kafkaProducer.send(**new** ProducerRecord<String, String>(**"order"**, **"订单信息！"**+i));  Thread.*sleep*(100);  }  } } |

3.编写代码-写消费者的代码

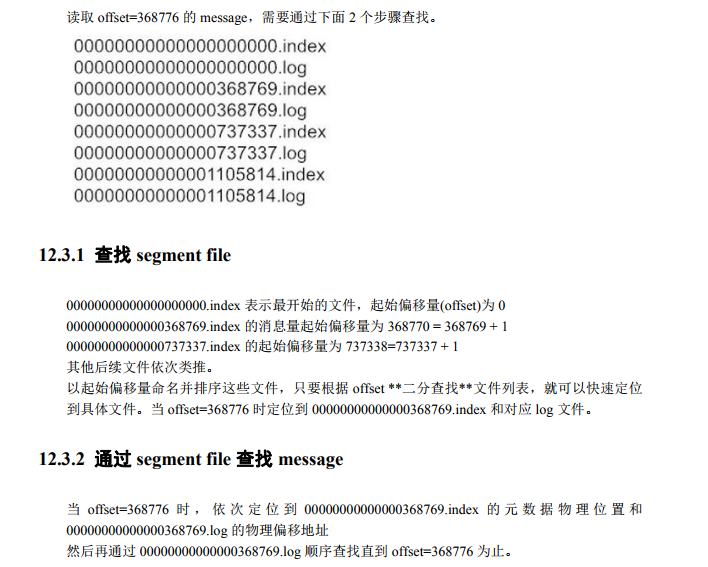
|  |
| --- |
| */\*\*  \* 消费订单数据--- javaben.tojson  \*/* **public class** OrderConsumer {  **public static void** main(String[] args) {  *// 1\连接集群* Properties props = **new** Properties();  props.put(**"bootstrap.servers"**, **"node01:9092"**);  props.put(**"group.id"**, **"test"**);  props.put(**"enable.auto.commit"**, **"true"**);  props.put(**"auto.commit.interval.ms"**, **"1000"**);  props.put(**"key.deserializer"**,  **"org.apache.kafka.common.serialization.StringDeserializer"**);  props.put(**"value.deserializer"**,  **"org.apache.kafka.common.serialization.StringDeserializer"**);  KafkaConsumer<String, String> kafkaConsumer = **new** KafkaConsumer<String, String>(props); *// 2、发送数据 发送数据需要，订阅下要消费的topic。 order* kafkaConsumer.subscribe(Arrays.*asList*(**"order"**));  **while** (**true**) {  ConsumerRecords<String, String> consumerRecords = kafkaConsumer.poll(100);*// jdk queue offer插入、poll获取元素。 blockingqueue put插入原生，take获取元素* **for** (ConsumerRecord<String, String> record : consumerRecords) {  System.***out***.println(**"消费的数据为："** + record.value());  }  }  } } |

auto.offset.reset值含义解释

earliest   
当各分区下有已提交的offset时，从提交的offset开始消费；无提交的offset时，从头开始消费   
latest   
当各分区下有已提交的offset时，从提交的offset开始消费；无提交的offset时，消费新产生的该分区下的数据   
none   
topic各分区都存在已提交的offset时，从offset后开始消费；只要有一个分区不存在已提交的offset，则抛出异常



**如果broker迟迟不给ack，而buffer又满了。开发者可以设置是否直接清空buffer中的数据。**



kafka在数据生产的时候，有一个数据分发策略。默认的情况使用DefaultPartitioner.class类。

这个类中就定义数据分发的策略。

数据分发策略的时候，可以指定数据发往哪个partition。

当ProducerRecord 的构造参数中有partition的时候，就可以发送到对应partition上

|  |
| --- |
| */\*\*  \* Creates a record to be sent to a specified topic and partition  \*  \** ***@param topic*** *The topic the record will be appended to  \** ***@param partition*** *The partition to which the record should be sent  \** ***@param key*** *The key that will be included in the record  \** ***@param value*** *The record contents  \*/* **public** ProducerRecord(String topic, Integer partition, K key, V value) {  **this**(topic, partition, **null**, key, value, **null**); } |

如果生产者没有指定partition，但是发送消息中有key，就key的hash值。

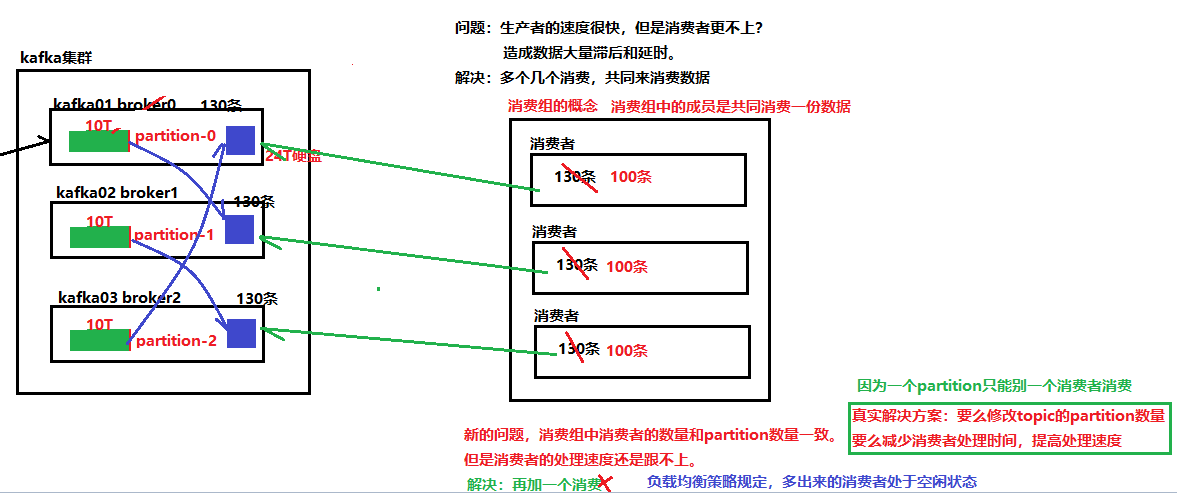
|  |
| --- |
| */\*\*  \* Create a record to be sent to Kafka  \*   \** ***@param topic*** *The topic the record will be appended to  \** ***@param key*** *The key that will be included in the record  \** ***@param value*** *The record contents  \*/* **public** ProducerRecord(String topic, K key, V value) {  **this**(topic, **null**, **null**, key, value, **null**); } |

既没有指定partition，也没有key的情况下如何发送数据。

使用轮询的方式发送数据。

|  |
| --- |
| */\*\*  \* Create a record with no key  \*   \** ***@param topic*** *The topic this record should be sent to  \** ***@param value*** *The record contents  \*/* **public** ProducerRecord(String topic, V value) {  **this**(topic, **null**, **null**, **null**, value, **null**); } |

消费者负载均衡



offset更新的方式，不区分是用的哪种api，大致分为两类：

1、自动提交，设置enable.auto.commit=true，更新的频率根据参数【auto.commit.interval.ms】来定。这种方式也被称为【at most once】，fetch到消息后就可以更新offset，无论是否消费成功。  
2、手动提交，设置enable.auto.commit=false，这种方式称为【at least once】。fetch到消息后，等消费完成再调用方法【consumer.commitSync()】，手动更新offset；如果消费失败，则offset也不会更新，此条消息会被重复消费一次。

Spark Streaming  +Kafka 使用底层API直接读取Kafka的Partition数据，正常Offset存储在CheckPoint中。但是这样无法实现Kafka监控工具对Kafka的监控，所以手动更新Offset到Zookeeper集群中

1:TopicAndPartition是对 topic和partition的id的封装的一个样例类

case class TopicAndPartition(topic: String, partitionId: Int)

2：OffsetRange 是对topic name，partition id，fromOffset（当前消费的开始偏移），untilOffset（当前消费的结束偏移）的封装。

final class OffsetRange private(val topic: String, val partition: Int, val fromOffset: Long, val untilOffset: Long) extends Serializable

KafkaRDD和HasOffsetRanges关系(构造参数和泛型省略，具体见源码)：

KafkaRDD extends RDD[R](sc, Nil) with Logging with HasOffsetRanges

KafkaCluster 完成对更新偏移到zookeeper集群的封装工具类。https://blog.csdn.net/Dax1n/article/details/53413111

val kc = new KafkaCluster(kafkaParams)

对offsetRanges数组遍历 强调：OffsetRange是对什么的封装？答案：topic名字，分区Id，开始偏移，结束偏移

for (offsets <- offsetsList) {

//offsets.untilOffset是结束偏移量

val o = kc.setConsumerOffsets(groupID, Map((topicAndPartition, offsets.untilOffset)))

if (o.isLeft) { println(s"Error updating the offset to Kafka cluster: ${o.left.get}") }

}