

springboot集成kafka以及如何确保kafka保证消息一致性

原创 Marii_ 于 2020-06-29 18:39:52 发布 1151 收藏 13

版权

文章标签: kafka

先讲一下如何集成吧

配置文件 `kafka.properties`:

```
1 kafka.consumer.zookeeper.connect=172.16.0.20:2181
2 kafka.consumer.servers=172.16.0.20:9092
3 kafka.producer.servers=172.16.0.20:9092
4
5
6 kafka.consumer.enable.auto.commit=false
7 kafka.consumer.session.timeout=15000
8 kafka.consumer.auto.commit.interval=100
9 kafka.consumer.auto.offset.reset=earliest
10 kafka.consumer.group.id=test
11 kafka.consumer.concurrency=10
12 kafka.consumer.maxPollRecordsConfig=100
13
14
15 kafka.producer.retries=1
16 #最大每次批量发送个数2048个
17 kafka.producer.batch.size=2048
18 #延迟5ms
19 kafka.producer.linger=5
20 #Producer端用于缓存消息的缓冲区大小, 单位为字节 33554432= 32MB
21 kafka.producer.buffer.memory=33554432
22
23
24 #kafka主题
25 #测试主题
26 kafka.topic.test = topic_test
27 #订单业务消费主题
28 kafka.topic.order =topic_order
29
```

生产者:

配置类

```
1 @Component
2 @Configuration
3 @EnableKafka
4 @PropertySource(value = "classpath:kafka.properties",encoding = "utf-8")
5 public class KafkaProducerConfig {
6     @Value("${kafka.producer.servers}")
7     private String servers;
8     @Value("${kafka.producer.retries}")
9     private int retries;
10    @Value("${kafka.producer.batch.size}")
11    private int batchSize;
12    @Value("${kafka.producer.linger}")
13    private int linger;
14    @Value("${kafka.producer.buffer.memory}")
15    private int bufferMemory;
16
17    @SuppressWarnings("rawtypes")
18    @Bean
19    public KafkaTemplate<String, String> kafkaTemplate() {
20        return new KafkaTemplate<>(producerFactory());
21    }
22
23    public ProducerFactory<String, String> producerFactory() {
24        return new DefaultKafkaProducerFactory<>(producerConfigs());
25    }
26
27    ,
28
29
```

```

26     }
27     public Map<String, Object> producerConfigs() {
28         Map<String, Object> props = new HashMap<String, Object>();
29         props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, servers);
30         props.put(ProducerConfig.RETRIES_CONFIG, retries);
31         props.put(ProducerConfig.BATCH_SIZE_CONFIG, batchSize);
32         props.put(ProducerConfig.LINGER_MS_CONFIG, linger);
33         props.put(ProducerConfig.BUFFER_MEMORY_CONFIG, bufferMemory);
34         props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
35         props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
36         return props;
37     }
38 }

```

消费者:

配置类

```

1  @Component
2  @Configuration
3  @EnableKafka
4  @PropertySource(value = "classpath:kafka.properties", encoding = "utf-8")
5  public class KafkaConsumerConfig {
6      @Value("${kafka.consumer.servers}")
7      private String servers;
8      @Value("${kafka.consumer.enable.auto.commit}")
9      private boolean enableAutoCommit;
10     @Value("${kafka.consumer.session.timeout}")
11     private String sessionTimeout;
12     @Value("${kafka.consumer.auto.commit.interval}")
13     private String autoCommitInterval;
14     @Value("${kafka.consumer.group.id}")
15     private String groupId;
16     @Value("${kafka.consumer.auto.offset.reset}")
17     private String autoOffsetReset;
18     @Value("${kafka.consumer.concurrency}")
19     private int concurrency;
20     @Value("${kafka.consumer.maxPollRecordsConfig}")
21     private int maxPollRecordsConfig;
22
23     @Bean
24     public KafkaListenerContainerFactory<ConcurrentMessageListenerContainer<String, String>> kafkaListenerContainerFactory() {
25         ConcurrentKafkaListenerContainerFactory<String, String> factory = new ConcurrentKafkaListenerContainerFactory();
26         factory.setConsumerFactory(consumerFactory());
27         factory.setConcurrency(concurrency);
28         factory.getContainerProperties().setPollTimeout(1500);
29         factory.setBatchListener(false); // @KafkaListener 批量消费 每个批次数量在Kafka配置参数中设置ConsumerConfig.MAX_POLL
30         factory.getContainerProperties().setAckMode(AckMode.MANUAL_IMMEDIATE); // 设置提交偏移量的方式
31         return factory;
32     }
33
34     public ConsumerFactory<String, String> consumerFactory() {
35         return new DefaultKafkaConsumerFactory<String, String>(consumerConfigs());
36     }
37
38     public Map<String, Object> consumerConfigs() {
39         Map<String, Object> propsMap = new HashMap<String, Object>(8);
40         propsMap.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, servers);
41         propsMap.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, enableAutoCommit);
42         propsMap.put(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG, autoCommitInterval);
43         propsMap.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, sessionTimeout);
44         propsMap.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
45         propsMap.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
46         propsMap.put(ConsumerConfig.GROUP_ID_CONFIG, groupId);
47         propsMap.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, autoOffsetReset);
48         propsMap.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, maxPollRecordsConfig); // 每个批次获取数
49         return propsMap;
50     }
51 }
52 }

```

这里面需要注意，我使用的是手动提交偏移量的方式，这里是配合下面的处理数据一致性用的

```
factory.getContainerProperties().setAckMode(AckMode.MANUAL_IMMEDIATE);
```

```
//设置提交偏移量的方式
```

以上内容为基础配置类

kafka生产者发送工具类

```

1  import java.util.concurrent.ExecutionException;
2
3  import org.slf4j.Logger;
4  import org.slf4j.LoggerFactory;
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.kafka.core.KafkaTemplate;
7  import org.springframework.util.concurrent.FailureCallback;
8  import org.springframework.util.concurrent.ListenableFuture;
9  import org.springframework.util.concurrent.SuccessCallback;
10
11 import com.alibaba.fastjson.JSON;
12
13 /**
14  *
15  * @ClassName: KafkaProducer.class
16  * @Description: Kafka生产者类
17  * @version: v1.0.0
18  * @author: mario
19  * @date: 2018年6月20日 <br/>
20  *      Modification History: <br/>
21  *      Date Author Version Description <br/>
22  *      -----* <br/>
23  *      2018年6月20日 mario v1.0.0 修改原因 <br/>
24  */
25 public abstract class KafkaProducer implements FailureCallback, SuccessCallback {
26     private Logger logger = LoggerFactory.getLogger(getClass());
27     @Autowired
28     private KafkaTemplate kafkaTemplate;
29
30     /**
31      * @Function: KafkaProducer.java
32      * @Description: 发送成功回调函数, 有需求请重写
33      * @param result
34      * @return
35      * @version: v1.0.0
36      * @author: mario
37      * @date: 2018年6月27日
38      *
39      *      <br/>
40      *      Modification History:<br/>
41      *      Date Author Version Description <br/>
42      *      -----* <br/>
43      *      2018年6月27日 mario v1.0.0 新增 <br/>
44      */
45     public abstract void onSuccess(Object result);
46
47     /**
48      * @Function: KafkaProducer.java
49      * @Description: 发送失败回调函数, 有需求请重写
50      * @param ex
51      * @return
52      * @version: v1.0.0
53      * @author: mario
54      * @date: 2018年6月27日
55      *
56      *      <br/>
57      *      Modification History:<br/>
58      *      Date Author Version Description <br/>
59      *      -----* <br/>
60      *      2018年6月27日 mario v1.0.0 新增 <br/>
61      */
62     public abstract void onFailure(Throwable ex);
63
64     @SuppressWarnings("unchecked")

```

```

61     public void asyncSendMessage(String topic, Object data) {
62         long startTime = System.currentTimeMillis();
63         logger.debug("开始异步发送kafka数据...");
64         ListenableFuture listenableFuture = kafkaTemplate.send(topic, JSON.toJSONString(data));
65         logger.debug("发送耗时={}....开始注册回调", (System.currentTimeMillis()-startTime));
66         listenableFuture.addCallback(this, this);
67         logger.debug("注册回调耗时={}", (System.currentTimeMillis()-startTime));
68     }
69
70     @SuppressWarnings("unchecked")
71     public Object syncSendMessage(String topic, Object data) throws InterruptedException, ExecutionException {
72         ListenableFuture listenableFuture = kafkaTemplate.send(topic, JSON.toJSONString(data));
73         return listenableFuture.get();
74     }
75
76     @SuppressWarnings("unchecked")
77     public void asyncSendMessage(String topic, String key, Object data) {
78         ListenableFuture listenableFuture = kafkaTemplate.send(topic, key, JSON.toJSONString(data));
79         listenableFuture.addCallback(this, this);
80     }
81
82     @SuppressWarnings("unchecked")
83     public Object syncSendMessage(String topic, String key, Object data)
84         throws InterruptedException, ExecutionException {
85         ListenableFuture listenableFuture = kafkaTemplate.send(topic, key, JSON.toJSONString(data));
86         return listenableFuture.get();
87     }
88
89     @SuppressWarnings("unchecked")
90     public void asyncSendMessage(String topic, int partition, String key, Object data) {
91         ListenableFuture listenableFuture = kafkaTemplate.send(topic, partition, key, JSON.toJSONString(data));
92         listenableFuture.addCallback(this, this);
93     }
94
95     @SuppressWarnings("unchecked")
96     public Object syncSendMessage(String topic, int partition, String key, Object data)
97         throws InterruptedException, ExecutionException {
98         ListenableFuture listenableFuture = kafkaTemplate.send(topic, partition, key, JSON.toJSONString(data));
99         return listenableFuture.get();
100    }
101
102     @SuppressWarnings("unchecked")
103     public void asyncSendMessage(String topic, int partition, Object data) {
104         ListenableFuture listenableFuture = kafkaTemplate.send(topic, partition, JSON.toJSONString(data));
105         listenableFuture.addCallback(this, this);
106     }
107
108     @SuppressWarnings("unchecked")
109     public Object syncSendMessage(String topic, int partition, Object data)
110         throws InterruptedException, ExecutionException {
111         ListenableFuture listenableFuture = kafkaTemplate.send(topic, partition, JSON.toJSONString(data));
112         return listenableFuture.get();
113     }
114 }
115

```

这里我一般经常使用的是异步发送的方式，`asyncSendMessage(String topic, Object data)`，这个方法发送数据来保证吞吐效率，所以需要手动处理回调信息，以保证发送端的消息不丢失
也就是要重写`onSuccess`和`onFailure`回调函数

以上就是springboot集成kafka的内容，下面来说一下如何保证消息一致性。

因为kafka本身是一个消息中间件，做消息转发，可以理解两个系统之间通讯的传话员吧
所以一致性也就是要保持发送端确保发送数据不丢失，消费端确保消费数据不丢失

发送端代码：

```

1  import java.util.concurrent.ExecutionException;
2
3  import com.vsj.model.KafkaSendModel;
4

```

```
5  import org.apache.kafka.clients.producer.ProducerRecord;
6  import org.apache.kafka.common.KafkaException;
7  import org.slf4j.Logger;
8  import org.slf4j.LoggerFactory;
9  import org.springframework.beans.factory.annotation.Autowired;
10 import org.springframework.context.annotation.Scope;
11 import org.springframework.scheduling.annotation.Async;
12 import org.springframework.stereotype.Component;
13 import org.springframework.kafka.core.KafkaProducerException;
14 import org.springframework.kafka.support.SendResult;
15
16 import com.alibaba.fastjson.JSON;
17 import com.alibaba.fastjson.JSONObject;
18 import com.vsj.common.kafka.KafkaProducer;
19 import com.vsj.dao.KafkaSendDAO;
20
21 @Component
22 @Scope("prototype")
23 public class KafkaSenderHandler extends KafkaProducer{
24
25     private Logger logger = LoggerFactory.getLogger(getClass());
26     @Autowired
27     private KafkaSendDAO kafkaSendDAO;
28
29     @SuppressWarnings("rawtypes")
30     @Override
31     public void onSuccess(Object result) {
32         long startTime = System.currentTimeMillis();
33         ProducerRecord producerRecord = ((SendResult) result).getProducerRecord();
34         String value = producerRecord.value().toString();
35         logger.debug("kafka发送成功,回调数据={}", value);
36         JSONObject valueObject = JSONObject.parseObject(value);
37         Integer id=valueObject.getInteger("id");
38         logger.debug("开始删除...id={}", id);
39         kafkaSendDAO.deleteByPrimaryKey(id);
40         logger.debug("成功回调处理完成,耗时={}", (System.currentTimeMillis()-startTime));
41     }
42
43     @SuppressWarnings("rawtypes")
44     @Override
45     public void onFailure(Throwable ex) {
46         long startTime = System.currentTimeMillis();
47         logger.debug("kafka发送失败,回调数据={}", JSON.toJSONString(ex));
48         ProducerRecord producerRecord = ((KafkaProducerException) ex).getProducerRecord();
49         String value = producerRecord.value().toString();
50         KafkaSendModel kafkaSendModel = JSONObject.parseObject(value, KafkaSendModel.class);
51         Integer id= kafkaSendModel.getId();
52         logger.debug("开始更新kafka记录失败次数...id={}", id);
53         kafkaSendDAO.updateFileCount(id);
54         logger.debug("失败回调处理完成,耗时={}", (System.currentTimeMillis()-startTime));
55     }
56
57
58     @Override
59     @Async("kafkaAsync")
60     public void asyncSendMessage(String topic, Object data) {
61         super.asyncSendMessage(topic,data);
62     }
63
64     @Override
65     public Object syncSendMessage(String topic, Object data) throws InterruptedException, ExecutionException {
66         return super.syncSendMessage(topic, data);
67     }
68
69     @Override
70     @Async("kafkaAsync")
71     public void asyncSendMessage(String topic, String key, Object data) {
72         super.asyncSendMessage(topic, key, data);
73     }
74
75 }
```

```

76     @Override
77     public Object syncSendMessage(String topic, String key, Object data)
78         throws InterruptedException, ExecutionException {
79         return super.syncSendMessage(topic, key, data);
80     }
81
82     @Override
83     @Async("kafkaAsync")
84     public void asyncSendMessage(String topic, int partition, String key, Object data) {
85         super.asyncSendMessage(topic, partition, key, data);
86     }
87
88     @Override
89     public Object syncSendMessage(String topic, int partition, String key, Object data)
90         throws InterruptedException, ExecutionException {
91         return super.syncSendMessage(topic, partition, key, data);
92     }
93
94     @Override
95     @Async("kafkaAsync")
96     public void asyncSendMessage(String topic, int partition, Object data) {
97         super.asyncSendMessage(topic, partition, data);
98     }
99
100    @Override
101    public Object syncSendMessage(String topic, int partition, Object data)
102        throws InterruptedException, ExecutionException {
103        return super.syncSendMessage(topic, partition, data);
104    }
105 }

```

这里注意下@Async("kafkaAsync")这个注解，这里为了保证异步的吞吐量，增加了一个线程池，使得asyncSendMessage方法为异步调用。这里我在外面又套了一层，加了个工具类来调用

*还要注意，这个类必须要加@Scope("prototype")注解，否则spring默认为单例模式，会导致后续注册的回调覆盖之前的回调，导致无法处理每条数据的回调信息。

```

1
2  import com.vsj.common.handler.KafkaSenderHandler;
3  import com.vsj.dao.KafkaSendDAO;
4  import com.vsj.model.KafkaSendModel;
5  import org.apache.kafka.common.KafkaException;
6  import org.slf4j.Logger;
7  import org.slf4j.LoggerFactory;
8  import org.springframework.beans.factory.annotation.Autowired;
9  import org.springframework.stereotype.Component;
10
11  import java.util.concurrent.ExecutionException;
12
13
14  @Component
15  public class KafkaSenderHelper {
16
17      private Logger logger = LoggerFactory.getLogger(getClass());
18
19      @Autowired
20      private KafkaSendDAO kafkaSendDAO;
21
22      @Autowired
23      private KafkaSenderHandler kafkaSenderHandler;
24
25
26      public void asyncSendMessage(String topic, Object data) {
27          KafkaSendModel object = null;
28          try {
29              object = new KafkaSendModel(topic, data);
30              kafkaSendDAO.insert(object);
31          } catch (Exception e) {
32              throw new KafkaException("保存kafka发送消息入库失败", e);
33          }
34      }
35  }

```

```

34     }
35     kafkaSenderHandler.asyncSendMessage(topic, object);
36 }
37
38 public Object syncSendMessage(String topic, Object data) throws InterruptedException, ExecutionException {
39     KafkaSendModel object = new KafkaSendModel(topic, data);
40     return kafkaSenderHandler.syncSendMessage(topic, object);
41 }
42
43 public void asyncSendMessage(String topic, String key, Object data) {
44     KafkaSendModel object = null;
45     try {
46         object = new KafkaSendModel(topic,data,key);
47         kafkaSendDAO.insert(object);
48     } catch (Exception e) {
49         throw new KafkaException("保存消息实体入库失败", e);
50     }
51     kafkaSenderHandler.asyncSendMessage(topic, key, object);
52 }
53
54 public Object syncSendMessage(String topic, String key, Object data)
55     throws InterruptedException, ExecutionException {
56     KafkaSendModel object = new KafkaSendModel(topic,data,key);
57     return kafkaSenderHandler.syncSendMessage(topic, key, object);
58 }
59
60 public void asyncSendMessage(String topic, int partition, String key, Object data) {
61     KafkaSendModel object = null;
62     try {
63         object = new KafkaSendModel(topic,data,key);
64         kafkaSendDAO.insert(object);
65     } catch (Exception e) {
66         throw new KafkaException("保存消息实体入库失败", e);
67     }
68     kafkaSenderHandler.asyncSendMessage(topic, partition, key, object);
69 }
70
71 public Object syncSendMessage(String topic, int partition, String key, Object data)
72     throws InterruptedException, ExecutionException {
73     KafkaSendModel object = new KafkaSendModel(topic,data,key);
74     return kafkaSenderHandler.syncSendMessage(topic, partition, key, object);
75 }
76
77 public void asyncSendMessage(String topic, int partition, Object data) {
78     KafkaSendModel object = null;
79     try {
80         object = new KafkaSendModel(topic,data);
81         kafkaSendDAO.insert(object);
82     } catch (Exception e) {
83         throw new KafkaException("保存消息实体入库失败", e);
84     }
85     kafkaSenderHandler.asyncSendMessage(topic, partition, object);
86 }
87
88 public Object syncSendMessage(String topic, int partition, Object data)
89     throws InterruptedException, ExecutionException {
90     KafkaSendModel object = new KafkaSendModel(topic,data);
91     return kafkaSenderHandler.syncSendMessage(topic, partition, object);
92 }
93 }

```

这里看下这个方法asyncSendMessage, KafkaSendModel 是一个数据库的实体对象

这里在发送消息到kafka之前, 先将要发送的信息存入数据库备份。存入成功后再调用发送方法发送

KafkaSendModel object = null;

```

try {
    object = new KafkaSendModel(topic,data);
    kafkaSendDAO.insert(object);
} catch (Exception e) {

```



```
throw new KafkaException("保存kafka发送消息入库失败", e);
}
kafkaSenderHandler.asyncSendMessage(topic, object);
```

当发送成功后，会进入**KafkaSenderHandler.onSuccess**回调函数，这时删掉备份数据即可

当发送失败，会进入**KafkaSenderHandler.onFailure**回调函数，这时更新状态，标记发送失败，后续通过定时任务进行重发，或做预警机制。正常来说KafkaSendModel实体的表中，应该是一条数据都不存在，即表示所有数据发送成功。

下面再说一下消费端：

```
1  import org.apache.kafka.clients.consumer.ConsumerRecord;
2  import org.slf4j.Logger;
3  import org.slf4j.LoggerFactory;
4  import org.springframework.beans.factory.annotation.Autowired;
5  import org.springframework.kafka.annotation.KafkaListener;
6  import org.springframework.kafka.support.Acknowledgment;
7  import org.springframework.stereotype.Component;
8
9  import com.vsj.common.handler.KafkaConsumerHandler;
10 import com.vsj.consumer.OrderTopicConsumer;
11
12
13 @Component
14 public class OrderListener {
15
16     private Logger logger = LoggerFactory.getLogger(getClass());
17     @Autowired
18     private KafkaConsumerHandler<OrderTopicConsumer> kafkaConsumerHandler;
19
20     // 单条消费
21     @KafkaListener(groupId = "group1", topics = {"${kafka.topic.order}"}, containerFactory = "kafkaListenerContainerFactory")
22     public void cloudSeatEventTopic(@SuppressWarnings("rawtypes") ConsumerRecord record, Acknowledgment ack) {
23         try {
24             kafkaConsumerHandler.consume(record, OrderTopicConsumer.class);
25         } catch (Exception e) {
26             logger.error("kafka消费异常,主题={},exception={}", record.topic(), e);
27         } finally {
28             ack.acknowledge();// 手动提交偏移量
29         }
30     }
31 }
32 }
```

使用@KafkaListener注解来配置监听者，topics属性指定监听的topic。

这里要注意 我使用的是ack.acknowledge();// 手动提交偏移量的方式在finally来处理topic。

下面来介绍一下KafkaConsumerHandler，这里我使用的是一个装饰者的模式

```
1  package com.vsj.common.handler;
2
3  import cn.hutool.core.date.DateTime;
4  import com.vsj.common.utils.DateUtil;
5  import org.apache.kafka.clients.consumer.ConsumerRecord;
6  import org.slf4j.Logger;
7  import org.slf4j.LoggerFactory;
8  import org.springframework.beans.factory.annotation.Autowired;
9  import org.springframework.stereotype.Component;
10
11 import com.alibaba.fastjson.JSON;
12 import com.vsj.common.service.IKafkaTopicConsumer;
13 import com.vsj.dao.KafkaConsumerDAO;
14 import com.vsj.model.KafkaConsumeModel;
15 import com.vsj.model.KafkaSendModel;
16
17 import java.text.SimpleDateFormat;
18
19 /**
20  *
21  * @ClassName: KafkaConsumerHandler
```



```

22  * @Description: kafka消费处理类
23  * 泛型对象为具体的业务处理类
24  * @author: mario
25  * @date: 2019年7月24日 下午3:24:48
26  * @copyright: 青岛微视角文化传媒有限公司
27  * @param <T> 泛型对象为具体的业务处理类
28  */
29 @Component
30 public class KafkaConsumerHandler<T extends IKafkaTopicConsumer> {
31     private Logger logger = LoggerFactory.getLogger(getClass());
32
33     @Autowired
34     private KafkaConsumerDAO kafkaConsumerDAO;
35
36     /**
37      * @Title: consume
38      * @Description: kafka消费者统一处理类
39      * @param record
40      * @param cls
41      * @author mario
42      * @return: void
43      */
44     @SuppressWarnings("rawtypes")
45     public void consume(ConsumerRecord record, Class<T> cls) {
46         try {
47             T consumer = (T) cls.newInstance();
48             logger.debug("接受到kafka消费,record={},cls={}", record.toString(), cls.getName());
49             KafkaSendModel kafkaSendModel = JSON.parseObject(String.valueOf(record.value()), KafkaSendModel.class);
50             consumer.doConsume(kafkaSendModel.getRecord());
51         } catch (Exception e) {
52             logger.error("消费者消费异常:{}", e);
53             try {
54                 KafkaConsumeModel object = new KafkaConsumeModel(record.topic(), String.valueOf(record.value()));
55                 kafkaConsumerDAO.insert(object);
56             } catch (Exception e2) {
57                 logger.error("消费者异常数据写库异常....e={}", e2);
58             }
59         }
60     }
61 }
62

```

注意看catch里面的代码，当consumer.doConsume消费异常时，会进入catch中，定义一个KafkaConsumeModel 实体，来存储至消费失败的数据表中，以便后续定时任务重新消费，或预警机制提示人工核对数据，确保了消费和发送的数据一致性。如果全部数据正常消费成功，则该表中不应存在任何数据。

这时会有人问，如果消费中途，服务崩溃如何处理，那不是没有写进这张表么？

你忘了前面的提交偏移量方式为手动提交了么。进行到中途如果服务崩溃，则偏移量并未提交，后续服务恢复正常，则会重新消费此条数据。以确保消费不丢失。

下面给一个实际业务消费的具体类，看一下

```

1  public interface IKafkaTopicConsumer {
2      void doConsume(String record ) throws Exception;
3  }

1  package com.vsj.consumer;
2
3  import com.alibaba.fastjson.JSON;
4  import com.vsj.common.AbstractObjectConverter;
5  import com.vsj.common.model.Order;
6  import com.vsj.common.service.CommonOrderService;
7  import com.vsj.common.service.IKafkaTopicConsumer;
8  import com.vsj.common.utils.SpringContextUtils;
9  import com.vsj.consumer.service.IOrderBountyService;
10 import com.vsj.model.VsjOrder;
11 import org.slf4j.Logger;
12 import org.slf4j.LoggerFactory;
13 import org.springframework.beans.factory.annotation.Autowired;
14

```

```

15 public class OrderTopicConsumer implements IKafkaTopicConsumer {
16
17     private Logger logger = LoggerFactory.getLogger(getClass());
18
19     public static IOrderService orderServiceImpl = null;
20
21     public static CommonOrderService commonOrderService = null;
22
23
24     @Override
25     public void doConsume(String record, boolean isTask) throws Exception {
26         long startTime = System.currentTimeMillis();
27         // 实体转换
28         Order order = JSON.parseObject(record, Order.class);
29         logger.debug("kafka接收到的待消费订单记录,开始处理...order={}", order);
30         // 处理订单信息
31         getOrderService().computingOrder(order);
32         // 处理库存
33         getCommonOrderService().editStock(order);
34         logger.debug("kafka订单信息处理完成,耗时={}", (System.currentTimeMillis() - startTime));
35     }
36
37     private IOrderService getOrderService(){
38         if(null == orderServiceImpl){
39             orderServiceImpl = SpringContextUtils.getBean("orderServiceImpl", IOrderService.class);
40         }
41         return orderServiceImpl ;
42     }
43
44     private CommonOrderService getCommonOrderService(){
45         if(null == commonOrderService){
46             commonOrderService = SpringContextUtils.getBean("commonOrderServiceImpl", CommonOrderService.class);
47         }
48         return commonOrderService;
49     }
50
51 }

```

这里注意service不能用@Autowired注解来写，KafkaConsumerHandler是以反射的形式来实例化业务类的，相当于new出来对象，你可以这么理解，所以注解是失效的。

上面线程池的类，也顺带贴上也

```

1 package com.vsj.config;
2
3 import java.util.concurrent.Executor;
4 import java.util.concurrent.ThreadPoolExecutor;
5
6 import com.vsj.common.config.KafkaExecutorConfig;
7 import org.springframework.context.annotation.Bean;
8 import org.springframework.context.annotation.Configuration;
9 import org.springframework.scheduling.annotation.EnableAsync;
10 import org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor;
11
12 /**
13  * @ClassName ExecutorConfig
14  * @Description: TODO
15  * @Author mario
16  * @Date 2019/11/22
17  * @Version V1.0
18  * @copyright: 青岛微视角文化传媒有限公司
19  */
20 @Configuration
21 @EnableAsync
22 class ExecutorConfigs {
23
24     @Bean
25     public Executor kafkaAsync(){
26         ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
27         executor.setCorePoolSize(KafkaExecutorConfig.corePoolSize);
28         executor.setMaxPoolSize(KafkaExecutorConfig.maxPoolSize);
29         executor.setQueueCapacity(KafkaExecutorConfig.queueCapacity);
30         executor.setThreadNamePrefix("Kafka-");
31         executor.initialize();
32         return executor;
33     }
34 }

```

```
27 | executor.setMaxPoolSize(KafkaExecutorConfig.maxPoolSize);
28 | executor.setQueueCapacity(KafkaExecutorConfig.queueCapacity);
29 | // 线程名称前缀
30 | executor.setThreadNamePrefix("KafkaExecutor-");
31 | // rejection-policy: 当pool已经达到max size的时候, 如何处理新任务
32 | // CALLER_RUNS: 不在新线程中执行任务, 而是有调用者所在的线程来执行
33 | // 等待任务在关机时完成--表明等待所有线程执行完
34 | //executor.setWaitForTasksToCompleteOnShutdown(true);
35 | // 等待时间 (默认为0, 此时立即停止), 并没等待xx秒后强制停止
36 | executor.setAwaitTerminationSeconds(KafkaExecutorConfig.keepAliveSeconds);
37 | executor.setRejectedExecutionHandler(new ThreadPoolExecutor.CallerRunsPolicy());
38 | executor.initialize();
39 | return executor;
40 | }
41 |
42 | }
```

application.yml中添加以下配置

```
1 | kafka-executor:
2 |   core-pool-size: 30
3 |   max-pool-size: 60
4 |   keep-alive-seconds: 60
5 |   queue-capacity: 10240
```

搞定收工, 撒有那拉