springboot集成kafka以及如何确保kafka保证消息一致性

原创 Mariii_ 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：

```
kafka.consumer.zookeeper.connect=172.16.0.20:2181
kafka.consumer.servers=172.16.0.20:9092
kafka.producer.servers=172.16.0.20:9092


kafka.consumer.enable.auto.commit=false
kafka.consumer.session.timeout=15000
kafka.consumer.auto.commit.interval=100
kafka.consumer.auto.offset.reset=earliest
kafka.consumer.group.id=test
kafka.consumer.concurrency=10
kafka.consumer.maxPollRecordsConfig=100


kafka.producer.retries=1
#最大每次批量发送个数2048个
kafka.producer.batch.size=2048
#延迟5ms
kafka.producer.linger=5
#Producer端用于缓存消息的缓冲区大小，单位为字节 33554432= 32MB
kafka.producer.buffer.memory=33554432


#kafka主题
#测试主题
kafka.topic.test = topic_test
#订单业务消费主题
kafka.topic.order =topic_order
```

生产者：
配置类

```
@Component
@Configuration
@EnableKafka
@PropertySource(value = "classpath:kafka.properties",encoding = "utf-8")
public class KafkaProducerConfig {
        @Value("${kafka.producer.servers}")
        private String servers;
        @Value("${kafka.producer.retries}")
        private int retries;
        @Value("${kafka.producer.batch.size}")
        private int batchSize;
        @Value("${kafka.producer.linger}")
        private int linger;
        @Value("${kafka.producer.buffer.memory}")
        private int bufferMemory;

        @SuppressWarnings("rawtypes")
        @Bean
        public KafkaTemplate<String, String> kafkaTemplate() {
                return new KafkaTemplate(producerFactory());
        }

        public ProducerFactory<String, String> producerFactory() {
                return new DefaultKafkaProducerFactory<String, String>(producerConfigs());

```

```
        }
        public Map<String, Object> producerConfigs() {
                Map<String, Object> props = new HashMap<String, Object>();
                props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, servers);
                props.put(ProducerConfig.RETRIES_CONFIG, retries);
                props.put(ProducerConfig.BATCH_SIZE_CONFIG, batchSize);
                props.put(ProducerConfig.LINGER_MS_CONFIG, linger);
                props.put(ProducerConfig.BUFFER_MEMORY_CONFIG, bufferMemory);
                props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
                props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
                return props;
        }

}
```

消费者:

配置类

```
@Component
@Configuration
@EnableKafka
@PropertySource(value = "classpath:kafka.properties",encoding = "utf-8")
public class KafkaConsumerConfig {
    @Value("${kafka.consumer.servers}")
    private String servers;
    @Value("${kafka.consumer.enable.auto.commit}")
    private boolean enableAutoCommit;
    @Value("${kafka.consumer.session.timeout}")
    private String sessionTimeout;
    @Value("${kafka.consumer.auto.commit.interval}")
    private String autoCommitInterval;
    @Value("${kafka.consumer.group.id}")
    private String groupId;
    @Value("${kafka.consumer.auto.offset.reset}")
    private String autoOffsetReset;
    @Value("${kafka.consumer.concurrency}")
    private int concurrency;
    @Value("${kafka.consumer.maxPollRecordsConfig}")
    private int maxPollRecordsConfig;

    @Bean
    public KafkaListenerContainerFactory<ConcurrentMessageListenerContainer<String, String>> kafkaListenerContainerFa
        ConcurrentKafkaListenerContainerFactory<String, String> factory = new ConcurrentKafkaListenerContainerFactory
        factory.setConsumerFactory(consumerFactory());
        factory.setConcurrency(concurrency);
        factory.getContainerProperties().setPollTimeout(1500);
        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
        factory.getContainerProperties().setAckMode(AckMode.MANUAL_IMMEDIATE);//设置提交偏移量的方式
        return factory;
    }

    public ConsumerFactory<String, String> consumerFactory() {
        return new DefaultKafkaConsumerFactory<String, String>(consumerConfigs());
    }

    public Map<String, Object> consumerConfigs() {
        Map<String, Object> propsMap = new HashMap<String, Object>(8);
        propsMap.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, servers);
        propsMap.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, enableAutoCommit);
        propsMap.put(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG, autoCommitInterval);
        propsMap.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, sessionTimeout);
        propsMap.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        propsMap.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        propsMap.put(ConsumerConfig.GROUP_ID_CONFIG, groupId);
        propsMap.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, autoOffsetReset);
        propsMap.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, maxPollRecordsConfig);//每个批次获取数
        return propsMap;
    }

}
```

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

```
import java.util.concurrent.ExecutionException;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.util.concurrent.FailureCallback;
import org.springframework.util.concurrent.ListenableFuture;
import org.springframework.util.concurrent.SuccessCallback;

import com.alibaba.fastjson.JSON;

/**
 *
 * @ClassName: KafkaProducer.class
 * @Description: Kafka生产者类
 * @version: v1.0.0
 * @author: mario
 * @date: 2018年6月20日 <br/>
 *        Modification History: <br/>
 *        Date Author Version Description <br/>
 *        ---------------------------------------------------------* <br/>
 *        2018年6月20日 mario v1.0.0 修改原因 <br/>
 *
 */
public abstract class KafkaProducer implements FailureCallback, SuccessCallback {
        private Logger logger = LoggerFactory.getLogger(getClass());
        @Autowired
        private KafkaTemplate kafkaTemplate;

        /**
         *
         * @Function: KafkaProducer.java
         * @Description: 发送成功回调函数,有需求请重写
         * @param result
         * @return
         *
         * @version: v1.0.0
         * @author: mario
         * @date: 2018年6月27日
         *
         *        <br/>
         *        Modification History:<br/>
         *        Date Author Version Description <br/>
         *        ---------------------------------------------------------* <br/>
         *        2018年6月27日 mario v1.0.0 新增 <br/>
         */
        public abstract void onSuccess(Object result);

        /**
         *
         * @Function: KafkaProducer.java
         * @Description: 发送失败回调函数,有需求请重写
         * @param ex
         * @return
         *
         * @version: v1.0.0
         * @author: mario
         * @date: 2018年6月27日
         *
         *        <br/>
         *        Modification History:<br/>
         *        Date Author Version Description <br/>
         *        ---------------------------------------------------------* <br/>
         *        2018年6月27日 mario v1.0.0 新增 <br/>
         */
        public abstract void onFailure(Throwable ex);

        @SuppressWarnings("unchecked")
```

```
        public void asyncSendMessage(String topic, Object data) {
                long startTime = System.currentTimeMillis();
                logger.debug("开始异步发送kafka数据...");
                ListenableFuture listenableFuture = kafkaTemplate.send(topic, JSON.toJSONString(data));
                logger.debug("发送耗时={}....开始注册回调",(System.currentTimeMillis()-startTime));
                listenableFuture.addCallback(this, this);
                logger.debug("注册回调耗时={}",(System.currentTimeMillis()-startTime));
        }

        @SuppressWarnings("unchecked")
        public Object syncSendMessage(String topic, Object data) throws InterruptedException, ExecutionException {
                ListenableFuture listenableFuture = kafkaTemplate.send(topic, JSON.toJSONString(data));
                return listenableFuture.get();
        }

        @SuppressWarnings("unchecked")
        public void asyncSendMessage(String topic, String key, Object data) {
                ListenableFuture listenableFuture = kafkaTemplate.send(topic, key, JSON.toJSONString(data));
                listenableFuture.addCallback(this, this);
        }

        @SuppressWarnings("unchecked")
        public Object syncSendMessage(String topic, String key, Object data)
                        throws InterruptedException, ExecutionException {
                ListenableFuture listenableFuture = kafkaTemplate.send(topic, key, JSON.toJSONString(data));
                return listenableFuture.get();
        }

        @SuppressWarnings("unchecked")
        public void asyncSendMessage(String topic, int partition, String key, Object data) {
                ListenableFuture listenableFuture = kafkaTemplate.send(topic, partition, key, JSON.toJSONString(data)
                listenableFuture.addCallback(this, this);
        }

        @SuppressWarnings("unchecked")
        public Object syncSendMessage(String topic, int partition, String key, Object data)
                        throws InterruptedException, ExecutionException {
                ListenableFuture listenableFuture = kafkaTemplate.send(topic, partition, key, JSON.toJSONString(data)
                return listenableFuture.get();
        }

        @SuppressWarnings("unchecked")
        public void asyncSendMessage(String topic, int partition, Object data) {
                ListenableFuture listenableFuture = kafkaTemplate.send(topic, partition, JSON.toJSONString(data));
                listenableFuture.addCallback(this, this);
        }

        @SuppressWarnings("unchecked")
        public Object syncSendMessage(String topic, int partition, Object data)
                        throws InterruptedException, ExecutionException {
                ListenableFuture listenableFuture = kafkaTemplate.send(topic, partition, JSON.toJSONString(data));
                return listenableFuture.get();
        }
}
```

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，可以理解为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：

```
import java.util.concurrent.ExecutionException;

import com.vsj.model.KafkaSendModel;
```

```
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.common.KafkaException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Scope;
import org.springframework.scheduling.annotation.Async;
import org.springframework.stereotype.Component;
import org.springframework.kafka.core.KafkaProducerException;
import org.springframework.kafka.support.SendResult;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONObject;
import com.vsj.common.kafka.KafkaProducer;
import com.vsj.dao.KafkaSendDAO;


@Component
@Scope("prototype")
public class KafkaSenderHandler extends KafkaProducer{

        private Logger logger = LoggerFactory.getLogger(getClass());
        @Autowired
        private KafkaSendDAO kafkaSendDAO;

        @SuppressWarnings("rawtypes")
        @Override
        public void onSuccess(Object result) {
                long startTime = System.currentTimeMillis();
                ProducerRecord producerRecord = ((SendResult) result).getProducerRecord();
                String value = producerRecord.value().toString();
                logger.debug("kafka发送成功,回调数据={}",value);
                JSONObject valueObject = JSONObject.parseObject(value);
                Integer id=valueObject.getInteger("id");
                logger.debug("开始删除...id={}",id);
                kafkaSendDAO.deleteByPrimaryKey(id);
                logger.debug("成功回调处理完成,耗时={}",(System.currentTimeMillis()-startTime));
        }

        @SuppressWarnings("rawtypes")
        @Override
        public void onFailure(Throwable ex) {
                long startTime = System.currentTimeMillis();
                logger.debug("kafka发送失败,回调数据={}",JSON.toJSONString(ex));
                ProducerRecord producerRecord = ((KafkaProducerException) ex).getProducerRecord();
                String value = producerRecord.value().toString();
                KafkaSendModel kafkaSendModel = JSONObject.parseObject(value, KafkaSendModel.class);
                Integer id= kafkaSendModel.getId();
                logger.debug("开始更新kafka记录失败次数...id={}",id);
                kafkaSendDAO.updateFileCount(id);
                logger.debug("失败回调处理完成,耗时={}",(System.currentTimeMillis()-startTime));

        }

        @Override
        @Async("kafkaAsync")
        public void asyncSendMessage(String topic, Object data) {
        super.asyncSendMessage(topic,data);
        }

        @Override
        public Object syncSendMessage(String topic, Object data) throws InterruptedException, ExecutionException {
                return super.syncSendMessage(topic, data);
        }

        @Override
        @Async("kafkaAsync")
        public void asyncSendMessage(String topic, String key, Object data) {
                super.asyncSendMessage(topic, key, data);
        }

```

```
        @Override
        public Object syncSendMessage(String topic, String key, Object data)
                        throws InterruptedException, ExecutionException {
                return super.syncSendMessage(topic, key, data);
        }

        @Override
        @Async("kafkaAsync")
        public void asyncSendMessage(String topic, int partition, String key, Object data) {
                super.asyncSendMessage(topic, partition, key, data);
        }

        @Override
        public Object syncSendMessage(String topic, int partition, String key, Object data)
                        throws InterruptedException, ExecutionException {
                return super.syncSendMessage(topic, partition, key, data);
        }

        @Override
        @Async("kafkaAsync")
        public void asyncSendMessage(String topic, int partition, Object data) {
                super.asyncSendMessage(topic, partition, data);
        }

        @Override
        public Object syncSendMessage(String topic, int partition, Object data)
                        throws InterruptedException, ExecutionException {
                return super.syncSendMessage(topic, partition, data);
        }

}
```

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。

```

import com.vsj.common.handler.KafkaSenderHandler;
import com.vsj.dao.KafkaSendDAO;
import com.vsj.model.KafkaSendModel;
import org.apache.kafka.common.KafkaException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.util.concurrent.ExecutionException;


@Component
public class KafkaSenderHelper {

    private Logger logger = LoggerFactory.getLogger(getClass());

    @Autowired
    private KafkaSendDAO kafkaSendDAO;

    @Autowired
    private KafkaSenderHandler kafkaSenderHandler;


    public void asyncSendMessage(String topic, Object data) {
        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);
    }

    public Object syncSendMessage(String topic, Object data) throws InterruptedException, ExecutionException {
        KafkaSendModel object = new KafkaSendModel(topic, data);
        return kafkaSenderHandler.syncSendMessage(topic, object);
    }

    public void asyncSendMessage(String topic, String key, Object data) {
        KafkaSendModel object = null;
        try {
            object = new KafkaSendModel(topic,data,key);
            kafkaSendDAO.insert(object);
        } catch (Exception e) {
            throw new KafkaException("保存消息实体入库失败", e);
        }
        kafkaSenderHandler.asyncSendMessage(topic, key, object);
    }

    public Object syncSendMessage(String topic, String key, Object data)
            throws InterruptedException, ExecutionException {
        KafkaSendModel object = new KafkaSendModel(topic,data,key);
        return kafkaSenderHandler.syncSendMessage(topic, key, object);
    }

    public void asyncSendMessage(String topic, int partition, String key, Object data) {
        KafkaSendModel object = null;
        try {
            object = new KafkaSendModel(topic,data,key);
            kafkaSendDAO.insert(object);
        } catch (Exception e) {
            throw new KafkaException("保存消息实体入库失败", e);
        }
        kafkaSenderHandler.asyncSendMessage(topic, partition, key, object);
    }

    public Object syncSendMessage(String topic, int partition, String key, Object data)
            throws InterruptedException, ExecutionException {
        KafkaSendModel object = new KafkaSendModel(topic,data,key);
        return kafkaSenderHandler.syncSendMessage(topic, partition, key, object);
    }

    public void asyncSendMessage(String topic, int partition, Object data) {
        KafkaSendModel object = null;
        try {
            object = new KafkaSendModel(topic,data);
            kafkaSendDAO.insert(object);
        } catch (Exception e) {
            throw new KafkaException("保存消息实体入库失败", e);
        }
        kafkaSenderHandler.asyncSendMessage(topic, partition, object);
    }

    public Object syncSendMessage(String topic, int partition, Object data)
            throws InterruptedException, ExecutionException {
        KafkaSendModel object = new KafkaSendModel(topic,data);
        return kafkaSenderHandler.syncSendMessage(topic, partition, object);
    }

}
```

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：

```
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.kafka.support.Acknowledgment;
import org.springframework.stereotype.Component;

import com.vsj.common.handler.KafkaConsumerHandler;
import com.vsj.consumer.OrderTopicConsumer;


@Component
public class OrderListener {

        private Logger logger = LoggerFactory.getLogger(getClass());
        @Autowired
        private KafkaConsumerHandler<OrderTopicConsumer> kafkaConsumerHandler;

        //单条消费
        @KafkaListener(groupId = "group1" ,topics = {"${kafka.topic.order}"}, containerFactory = "kafkaListenerContai
        public void cloudSeatEventTopic(@SuppressWarnings("rawtypes") ConsumerRecord record, Acknowledgment ack) {
                try {
                        kafkaConsumerHandler.consume(record, OrderTopicConsumer.class);
                } catch (Exception e) {
                        logger.error("kafka消费异常,主题={},exception={}",record.topic(),e);
                } finally {
                        ack.acknowledge();// 手动提交偏移量
                }
        }

}
```

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

```
package com.vsj.common.handler;

import cn.hutool.core.date.DateTime;
import com.vsj.common.utils.DateUtil;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import com.alibaba.fastjson.JSON;
import com.vsj.common.service.IKafkaTopicConsumer;
import com.vsj.dao.KafkaConsumerDAO;
import com.vsj.model.KafkaConsumeModel;
import com.vsj.model.KafkaSendModel;

import java.text.SimpleDateFormat;

/**
 *
 * @ClassName: KafkaConsumerHandler
```

```
 * @Description: kafka消费处理类
 * 泛型对象为具体的业务处理类
 * @author: mario
 * @date: 2019年7月24日 下午3:24:48
 * @copyright: 青岛微视角文化传媒有限公司
 * @param <T>泛型对象为具体的业务处理类
 */
@Component
public class KafkaConsumerHandler<T extends IKafkaTopicConsumer> {
        private Logger logger = LoggerFactory.getLogger(getClass());

        @Autowired
        private KafkaConsumerDAO kafkaConsumerDAO;

        /**
         *
         * @Title: consume
         * @Description: kafka消费者统一处理类
         * @param record
         * @param cls
         * @author mario
         * @return: void
         */
        @SuppressWarnings("rawtypes")
        public void consume(ConsumerRecord record,Class<T> cls) {
                try {
                        T consumer = (T)cls.newInstance();
                        logger.debug("接受到kafka消费,record={},cls={}...",record.toString(),cls.getName());
                        KafkaSendModel kafkaSendModel = JSON.parseObject(String.valueOf(record.value()), KafkaSendMoc
                        consumer.doConsume(kafkaSendModel.getRecord());
                } catch (Exception e) {
                        logger.error("消费者消费异常:{}",e);
                        try {
                                KafkaConsumeModel object = new KafkaConsumeModel(record.topic(),String.valueOf(recorc
                                kafkaConsumerDAO.insert(object);
                        } catch (Exception e2) {
                                logger.error("消费者异常数据写库异常....e={}",e2);
                        }
                }
        }

}
```

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

```
public interface IKafkaTopicConsumer {
        void doConsume(String record ) throws Exception;
}
```

```
package com.vsj.consumer;

import com.alibaba.fastjson.JSON;
import com.vsj.common.AbstractObjectConverter;
import com.vsj.common.model.Order;
import com.vsj.common.service.CommonOrderService;
import com.vsj.common.service.IKafkaTopicConsumer;
import com.vsj.common.utils.SpringContextUtils;
import com.vsj.consumer.service.IOrderBountyService;
import com.vsj.model.VsjOrder;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;

```

```
public class OrderTopicConsumer implements IKafkaTopicConsumer {

        private Logger logger = LoggerFactory.getLogger(getClass());

        public static IOrderService orderServiceImpl = null;

        public static CommonOrderService commonOrderService = null;


        @Override
        public void doConsume(String record, boolean isTask) throws Exception {
                long startTime = System.currentTimeMillis();
                //实体转换
                Order order = JSON.parseObject(record, Order.class);
                logger.debug("kafka接收到的待消费订单记录,开始处理...order={}",order);
                //处理订单信息
                getOrderService().computingOrder(order);
                //处理库存
                getCommonOrderService().editStock(order);
                logger.debug("kafka订单信息处理完成,耗时={}",(System.currentTimeMillis() - startTime));
        }

        private IOrderService getOrderService(){
                if(null == orderServiceImpl){
                        orderServiceImpl = SpringContextUtils.getBean("orderServiceImpl ",IOrderService.class);
                }
                return orderServiceImpl ;
        }

        private CommonOrderService getCommonOrderService(){
                if(null == commonOrderService){
                        commonOrderService = SpringContextUtils.getBean("commonOrderServiceImpl",CommonOrderService.c
                }
                return commonOrderService;
        }

}
```

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，也顺带贴上吧

```
package com.vsj.config;

import java.util.concurrent.Executor;
import java.util.concurrent.ThreadPoolExecutor;

import com.vsj.common.config.KafkaExecutorConfig;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.annotation.EnableAsync;
import org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor;

/**
 * @ClassName ExecutorConfig
 * @Description: TODO
 * @Author mario
 * @Date 2019/11/22
 * @Version V1.0
 * @copyright: 青岛微视角文化传媒有限公司
 **/
@Configuration
@EnableAsync
class ExecutorConfigs {

    @Bean
    public Executor kafkaAsync(){
        ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
        executor.setCorePoolSize(KafkaExecutorConfig.corePoolSize);
```

```
        executor.setMaxPoolSize(KafkaExecutorConfig.maxPoolSize);
        executor.setQueueCapacity(KafkaExecutorConfig.queueCapacity);
        //线程名称前缀
        executor.setThreadNamePrefix("KafkaExecutor-");
        // rejection-policy：当pool已经达到max size的时候，如何处理新任务
        // 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
        //等待任务在关机时完成--表明等待所有线程执行完
        //executor.setWaitForTasksToCompleteOnShutdown(true);
        // 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
        executor.setAwaitTerminationSeconds(KafkaExecutorConfig.keepAliveSeconds);
        executor.setRejectedExecutionHandler(new ThreadPoolExecutor.CallerRunsPolicy());
        executor.initialize();
        return executor;
    }

}
```

application.yml中添加以下配置

```
kafka-executor:
  core-pool-size: 30
  max-pool-size: 60
  keep-alive-seconds: 60
  queue-capacity: 10240
```

搞定收工，撒有那拉