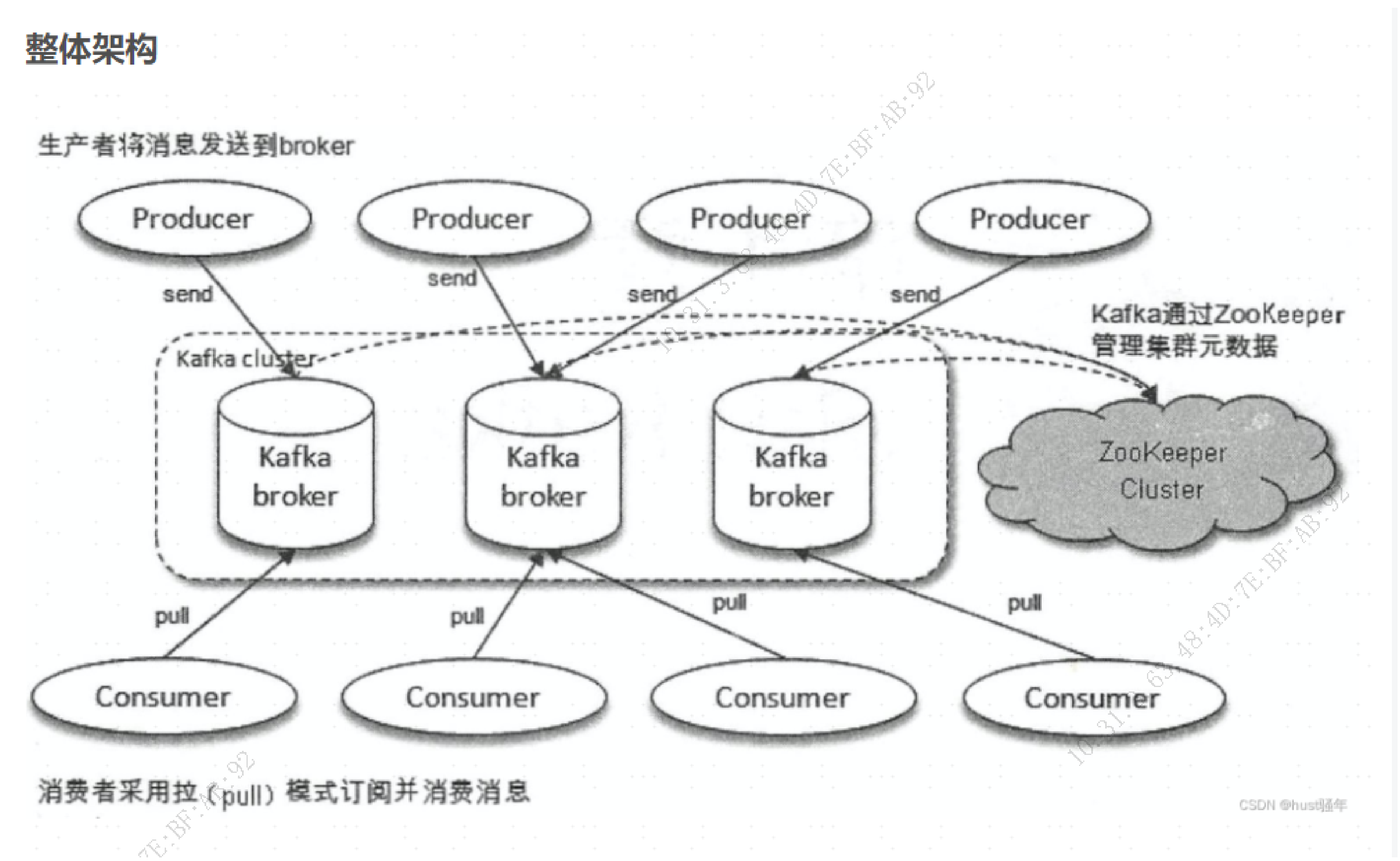
Spring-Kafka

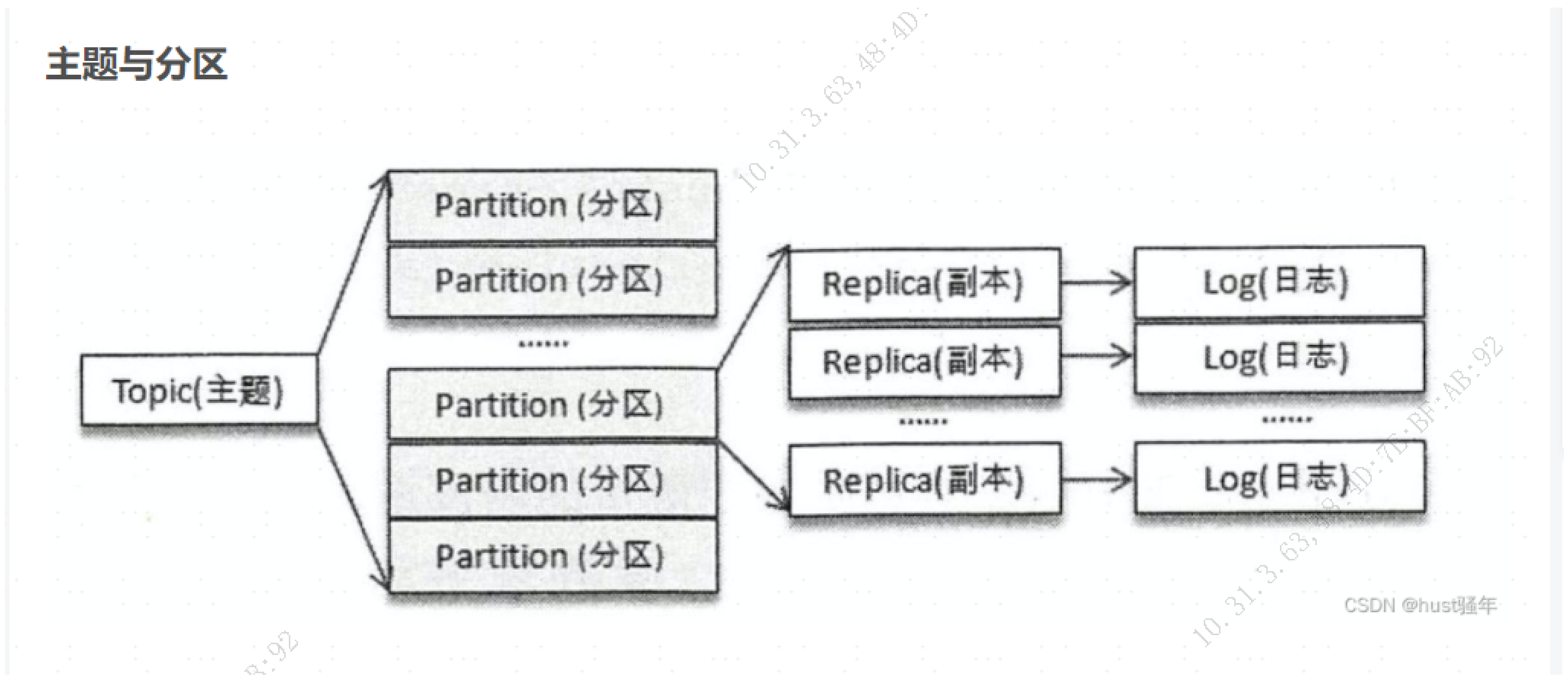
参考文献：https://blog.csdn.net/yuanlong122716/article/details/105160545/

官方文档：https://docs.spring.io/spring-kafka/reference/html/

一、简介

```
1.kafka是一个分布式-订阅消息传递系统，使用scala语言开发，基于zookeeper进行协调，多分区、多副本；它的特性是高吞吐、可持久化、可水平扩展、支持流数据处理，它具备三大功能：消息系统、存储系统、流式处
2.kafka基本概念
    Producer : 生产者，发送消息的一方
    Consumer : 消费者，接收消息的一方
    Broker : kafka节点，一个节点就是一个kafka server进程
    Topic : 主题，消息以主题来进行归类
    Partition : 分区，主题的所有消息分布在不同的区中，每个分区的消息一定是不同的，分区可以分布在不同的broker中
    Replica : 副本机制，每个分区引入多副本，leader副本和follower副本，leader副本处理读写，follower副本负责同步leader副本的数据，出现故障时，
              follower副本中重新选举出新的leader副本，进行故障转移
    PacificA : kafka采用的一致性协议
3.Kafka特性
    高吞吐量、低延迟：kafka每秒可以处理几十万条消息，它的延迟最低只有几毫秒，每个topic可以分多个partition, consumer group 对partition进行consume操作；
    可扩展性：kafka集群支持热扩展；
    持久性、可靠性：消息被持久化到本地磁盘，并且支持数据备份防止数据丢失；
    容错性：允许集群中节点失败（若副本数量为n,则允许n-1个节点失败）；
    高并发：支持数千个客户端同时读写；
    支持实时在线处理和离线处理：可以使用Storm这种实时流处理系统对消息进行实时进行处理，同时还可以使用Hadoop这种批处理系统进行离线处理；
4.Kafka 使用场景
    日志收集：一个公司可以用Kafka可以收集各种服务的log，通过kafka以统一接口服务的方式开放给各种consumer，例如Hadoop、Hbase、Solr等；
    消息系统：解耦和生产者和消费者、缓存消息等；
    用户活动跟踪：Kafka经常被用来记录web用户或者app用户的各种活动，如浏览网页、搜索、点击等活动，这些活动信息被各个服务器发布到kafka的topic中，然后订阅者通过订阅这些topic来做实时的监控分析，或者
中做离线分析和挖掘；
    运营指标：Kafka也经常用来记录运营监控数据。包括收集各种分布式应用的数据，生产各种操作的集中反馈，比如报警和报告；
    流式处理：比如spark streaming和storm；
    事件源；
```



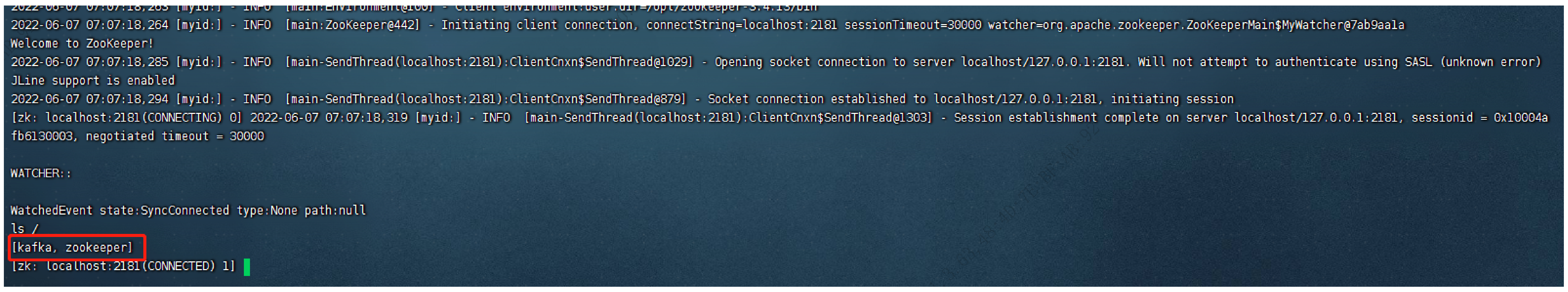


二、环境搭建

```
1.安装zk
//docker stop zookeeper; docker rm zookeeper
docker run -d --name zookeeper -p 2181:2181 -v /etc/localtime:/etc/localtime wurstmeister/zookeeper

2.安装kafka
//docker stop kafka; docker rm kafka
docker run -d --name kafka \
-p 9092:9092 \
-e KAFKA_BROKER_ID=0 \
-e KAFKA_ZOOKEEPER_CONNECT=10.207.0.167:2181/kafka \
-e KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://10.207.0.167:9092 \
-e KAFKA_LISTENERS=PLAINTEXT://0.0.0.0:9092 \
-t wurstmeister/kafka

//参数说明
-e KAFKA_BROKER_ID=0  在kafka集群中，每个kafka都有一个BROKER_ID来区分自己
-e KAFKA_ZOOKEEPER_CONNECT=10.207.0.167:2181/kafka 配置zookeeper管理kafka的路径10.207.0.167:2181/kafka
-e KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://10.207.0.167:9092  把kafka的地址端口注册给zookeeper
-e KAFKA_LISTENERS=PLAINTEXT://0.0.0.0:9092 配置kafka的监听端口
-v /etc/localtime:/etc/localtime 容器时间同步虚拟机的时间
```

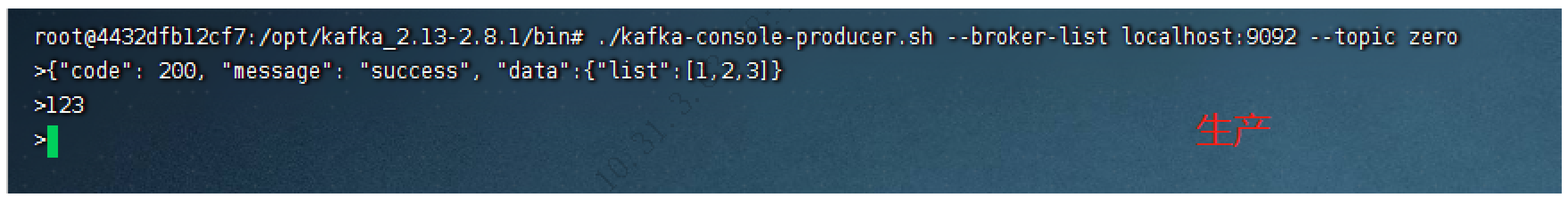
三、环境验证

```
1.zk验证
//进入容器验证
docker exec -it zookeeper bash
> cd /opt/zookeeper-3.4.13/bin
> ./zkCli.sh                                  //进入zk客户端
  > ls /                                      //查看根目录有俩节点 [kafka, zookeeper]
  > ls /kafka/brokers/topics/zero/partitions  //查看zero主题的partitions信息
  > get /kafka/brokers/topics/zero            //显示该节点的数据内容和属性信息
  > ls2 /kafka/brokers/topics/zero            //显示该节点的子节点信息和属性信息
```

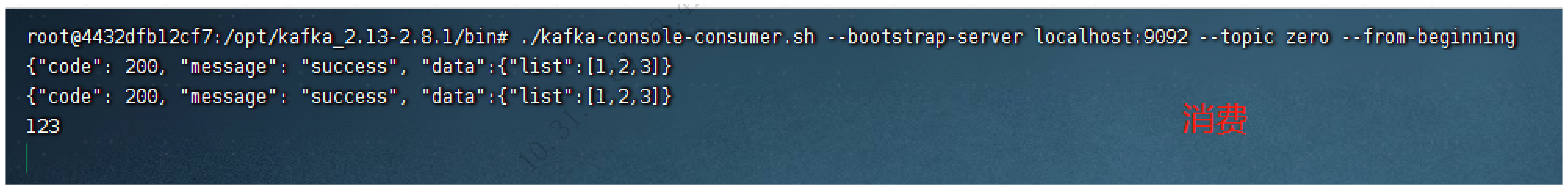


```
2022-06-07 07:07:18,264 [myid:] - INFO  [main:ZooKeeper@442] - Initiating client connection, connectString=localhost:2181 sessionTimeout=30000 watcher=org.apache.zookeeper.ZooKeeperMain$MyWatcher@7ab9aa1a
Welcome to ZooKeeper!
2022-06-07 07:07:18,285 [myid:] - INFO  [main-SendThread(localhost:2181):ClientCnxn$SendThread@1029] - Opening socket connection to server localhost/127.0.0.1:2181. Will not attempt to authenticate using SASL (unknown error)
JLine support is enabled
2022-06-07 07:07:18,294 [myid:] - INFO  [main-SendThread(localhost:2181):ClientCnxn$SendThread@879] - Socket connection established to localhost/127.0.0.1:2181, initiating session
[zk: localhost:2181(CONNECTING) 0] 2022-06-07 07:07:18,319 [myid:] - INFO  [main-SendThread(localhost:2181):ClientCnxn$SendThread@1303] - Session establishment complete on server localhost/127.0.0.1:2181, sessionid = 0x10004a
fb6130003, negotiated timeout = 30000

WATCHER::

WatchedEvent state:SyncConnected type:None path:null
ls /
[kafka, zookeeper]
[zk: localhost:2181(CONNECTED) 1]
```

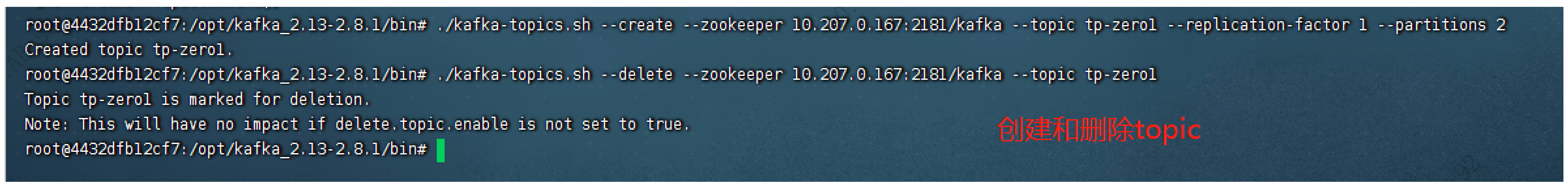
```
2.kafka验证
//进入容器验证
docker exec -it kafka bash
> cd /opt/kafka_2.13-2.8.1/bin
> ./kafka-console-producer.sh --broker-list localhost:9092 --topic zero      //生产者指定topic发送消息
  > {"code": 200, "message": "success", "data":{"list":[1,2,3]}}
> ./kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic zero --from-beginning    //另外打开窗口监听(消费)
> ./kafka-topics.sh --create --zookeeper 10.207.0.167:2181/kafka --topic tp-zero1 --replication-factor 1 --partitions 2  //创建topic和partitions
> ./kafka-topics.sh --delete --zookeeper 10.207.0.167:2181/kafka --topic tp-zero1    //同时删除容器中的topic数据和zk的topic目录
> cat /opt/kafka_2.13-2.8.1/config/server.properties    //查看配置信息，如日志目录为 /kafka/kafka-logs-4432dfb12cf7
> cd /kafka/kafka-logs-4432dfb12cf7/zero-0; cat 00000000000000000000.log
```



```
root@4432dfb12cf7:/opt/kafka_2.13-2.8.1/bin# ./kafka-console-producer.sh --broker-list localhost:9092 --topic zero
>{"code": 200, "message": "success", "data":{"list":[1,2,3]}
>123
>
```

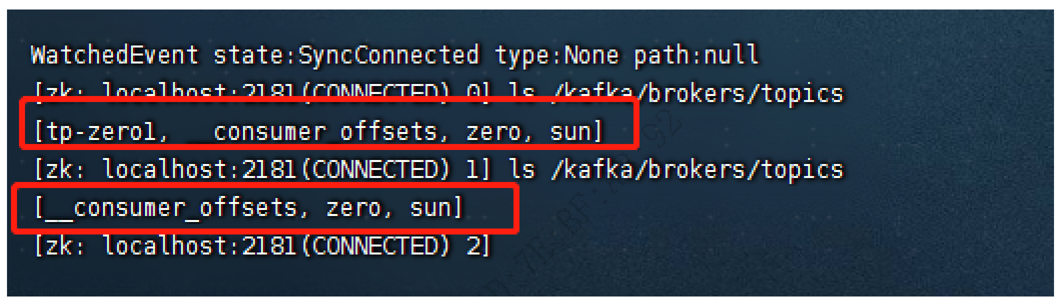


```
root@4432dfb12cf7:/opt/kafka_2.13-2.8.1/bin# ./kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic zero --from-beginning
{"code": 200, "message": "success", "data":{"list":[1,2,3]}
{"code": 200, "message": "success", "data":{"list":[1,2,3]}
123
```



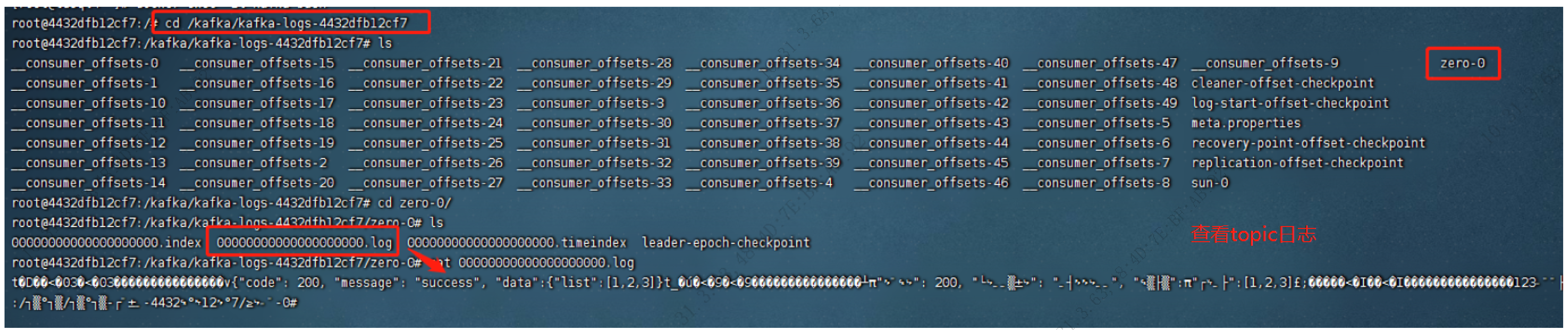
```
root@4432dfb12cf7:/opt/kafka_2.13-2.8.1/bin# ./kafka-topics.sh --create --zookeeper 10.207.0.167:2181/kafka --topic tp-zero1 --replication-factor 1 --partitions 2
Created topic tp-zero1.
root@4432dfb12cf7:/opt/kafka_2.13-2.8.1/bin# ./kafka-topics.sh --delete --zookeeper 10.207.0.167:2181/kafka --topic tp-zero1
Topic tp-zero1 is marked for deletion.
Note: This will have no impact if delete.topic.enable is not set to true.
root@4432dfb12cf7:/opt/kafka_2.13-2.8.1/bin#
```

zk



```
WatchedEvent state:SyncConnected type:None path:null
[zk: localhost:2181(CONNECTED) 0] ls /kafka/brokers/topics
[tp-zero1,  __consumer_offsets, zero, sun]
[zk: localhost:2181(CONNECTED) 1] ls /kafka/brokers/topics
[__consumer_offsets, zero, sun]
[zk: localhost:2181(CONNECTED) 2]
```

kafka



```
root@4432dfb12cf7:/# cd /kafka/kafka-logs-4432dfb12cf7
root@4432dfb12cf7:/kafka/kafka-logs-4432dfb12cf7# ls
__consumer_offsets-0   __consumer_offsets-15  __consumer_offsets-21  __consumer_offsets-28  __consumer_offsets-34  __consumer_offsets-40  __consumer_offsets-47  __consumer_offsets-9              zero-0
__consumer_offsets-1   __consumer_offsets-16  __consumer_offsets-22  __consumer_offsets-29  __consumer_offsets-35  __consumer_offsets-41  __consumer_offsets-48  cleaner-offset-checkpoint
__consumer_offsets-10  __consumer_offsets-17  __consumer_offsets-23  __consumer_offsets-3   __consumer_offsets-36  __consumer_offsets-42  __consumer_offsets-49  log-start-offset-checkpoint
__consumer_offsets-11  __consumer_offsets-18  __consumer_offsets-24  __consumer_offsets-30  __consumer_offsets-37  __consumer_offsets-43  __consumer_offsets-5   meta.properties
__consumer_offsets-12  __consumer_offsets-19  __consumer_offsets-25  __consumer_offsets-31  __consumer_offsets-38  __consumer_offsets-44  __consumer_offsets-6   recovery-point-offset-checkpoint
__consumer_offsets-13  __consumer_offsets-2   __consumer_offsets-26  __consumer_offsets-32  __consumer_offsets-39  __consumer_offsets-45  __consumer_offsets-7   replication-offset-checkpoint
__consumer_offsets-14  __consumer_offsets-20  __consumer_offsets-27  __consumer_offsets-33  __consumer_offsets-4   __consumer_offsets-46  __consumer_offsets-8   sun-0
root@4432dfb12cf7:/kafka/kafka-logs-4432dfb12cf7# cd zero-0/
root@4432dfb12cf7:/kafka/kafka-logs-4432dfb12cf7/zero-0# ls
00000000000000000000.index  00000000000000000000.log  00000000000000000000.timeindex  leader-epoch-checkpoint
root@4432dfb12cf7:/kafka/kafka-logs-4432dfb12cf7/zero-0# cat 00000000000000000000.log
t[illegible]{"code": 200, "message": "success", "data":{"list":[1,2,3]}t_[illegible]": 200, "[illegible]": "[illegible]", "[illegible]":[1,2,3]£;[illegible]123[illegible]
:[illegible]-4432[illegible]12[illegible]7/[illegible]-0#
```

四、Springboot整合Kafka

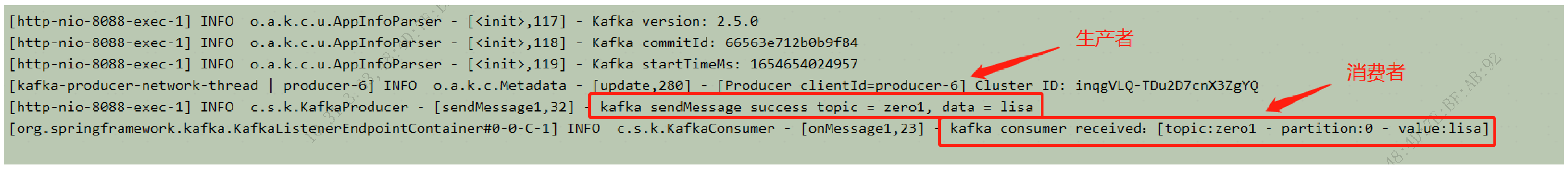
```
1.依赖
<!-- parent -->
<parent>
```

```
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.0.RELEASE</version>
</parent>
<!-- kafka -->
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
</dependency>
<!-- data 同步事务必须要有该依赖 -->
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-commons</artifactId>
</dependency>
<!-- fastjson -->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>1.2.79</version>
</dependency>

2.yml配置
spring:
  kafka:
    # 集群信息，多个用逗号隔开
    bootstrap-servers: 10.207.0.167:9092
    #【生产者】
    producer:
      # 事务ID=事务前缀+1，不为空即开启事务，生效必须 retries>0，acks=all
      #transaction-id-prefix: kfk_tx
      # 重试次数
      retries: 3
      # 应答级别:多少个分区副本备份完成时向生产者发送ack确认(可选0、1、all/-1)
      #procedure要求leader在考虑完成请求之前收到的确认数，用于控制发送记录在服务端的持久化，其值可以为如下：
      #acks = 0 如果设置为零，则生产者将不会等待来自服务器的任何确认，该记录将立即添加到套接字缓冲区并视为已发送。在这种情况下，无法保证服务器已收到记录，并且重试配置将不会生效（因为客户端通常不
记录返回的偏移量始终设置为-1。
      #acks = 1 这意味着leader会将记录写入其本地日志，但无需等待所有副本服务器的完全确认即可做出回应，在这种情况下，如果leader在确认记录后立即失败，但在将数据复制到所有的副本服务器之前，则记录将
      #acks = all 这意味着leader将等待完整的同步副本集以确认记录，这保证了只要至少一个同步副本服务器仍然存活，记录就不会丢失，这是最强有力的保证，这相当于acks = -1的设置。
      acks: 1
      # 每次批量发送消息的数量，16K
      batch-size: 16384
      # linger.ms为0表示每接收到一条消息就提交给kafka,这时候batch-size其实就没用了
      properties:
        linger.ms: 0
        # 自定义分区器，实现Partitioner接口
        # partitioner:
          # class: com.zeor.producer.CustomizePartitioner
      # 生产端缓冲区大小，32M
      buffer-memory: 33554432
      # 指定消息key和消息体的编解码方式，也可自定义
      key-serializer: org.apache.kafka.common.serialization.StringSerializer
      value-serializer: com.sf.kafka.serialization.ObjectSerializer
    #【生产者】
    consumer:
      # 指定默认消费者group id
      group-id: zeroGroupId
      # 是否自动提交offset
      enable-auto-commit: false
      properties:
        # 消费会话超时时间(超过这个时间consumer没有发送心跳,就会触发rebalance操作)
        session.timeout.ms: 120000
        # 消费请求超时时间
        request.timeout.ms: 180000
      # 默认为500，批量消费每次最多消费多少条消息
      #max-poll-records: 50
      # 提交offset延时(接收到消息后多久提交offset)
      auto-commit-interval: 100
      # 当kafka中没有初始offset或offset超出范围时将自动重置offset
      # 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，避免消息丢失
      # latest:重置为分区中最新的offset(消费分区中新产生的数据);
      # none:只要有一个分区不存在已提交的offset,就抛出异常;
      auto-offset-reset: earliest
      # 指定消息key和消息体的编解码方式
      key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
      value-deserializer: com.sf.kafka.serialization.ObjectDeserializer
    listener:
      # 指定listener容器中的线程数（同时消费的监听器），用于提高并发量，建议与分区数量一致
      concurrency: 3
      # 消费端监听的topic不存在时，项目启动会报错(关掉)
      missing-topics-fatal: false
      # 设置批量消费
      #type: batch
      # ACK模式：batch、record、time、count、count_time、manual、manual_immediate
      # 默认batch，手动调用Acknowledgment.acknowledge()后立即提交，一般使用这种
      ack-mode: manual_immediate

```

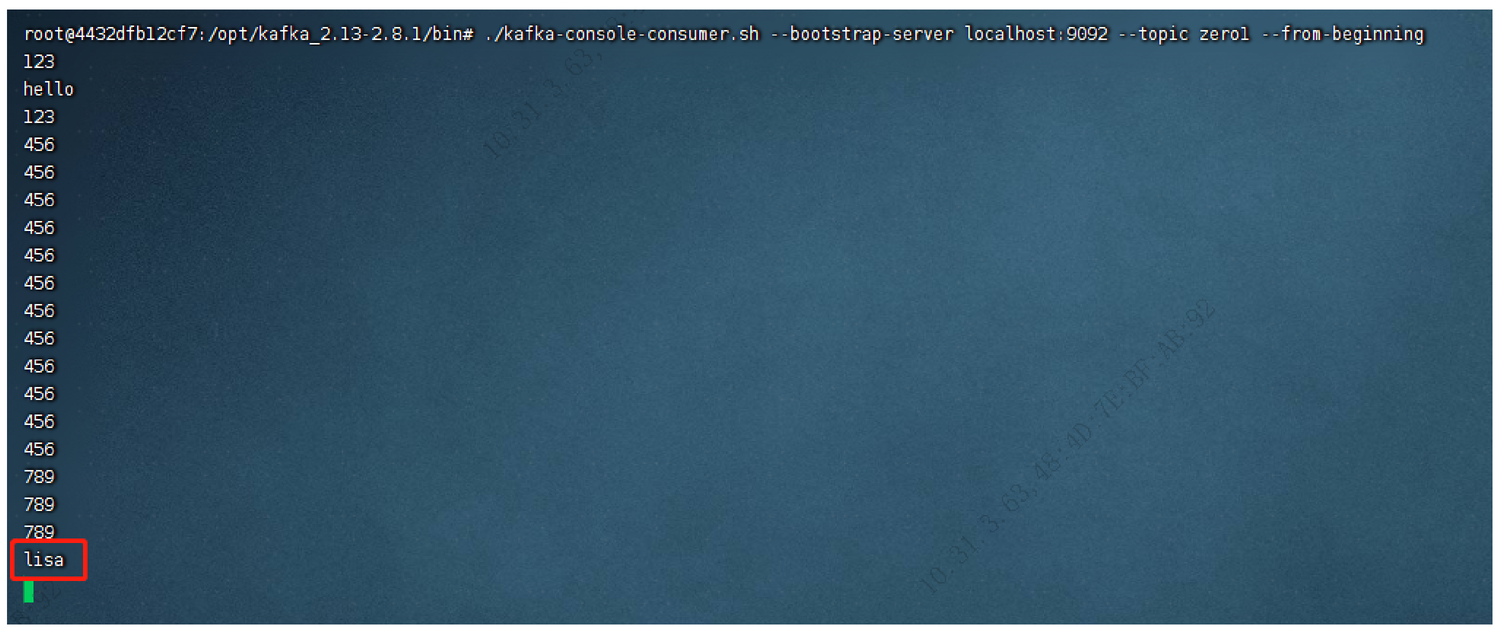
```
# 自定义配置
kafka:
  topic:
    group-id: zeroGroupId
    topic-name:
      - zero
      - zero1
      - sun
```

五、测试

```
1、简单发布-订阅
@GetMapping("send/{message}")
public void sendMessage1(@PathVariable("message") String message) throws Exception {
    kafkaTemplate.send(KFK_TOPIC_ZERO1, message).get(10, TimeUnit.SECONDS);
    log.info("kafka sendMessage success topic = {}, data = {}", KFK_TOPIC_ZERO1, message);
}

//kafkaTopicName = {"zero","zero1","sun"}  topicGroupId = "zeroGroupId"
@KafkaListener(topics = "#{kafkaTopicName}", groupId = "#{topicGroupId}")
public void onMessage1(ConsumerRecord<String, Object> record) {
    log.info("kafka consumer received: [topic:{} - partition:{} - value:{}]", record.topic(), record.partition(), record.value());
}
```

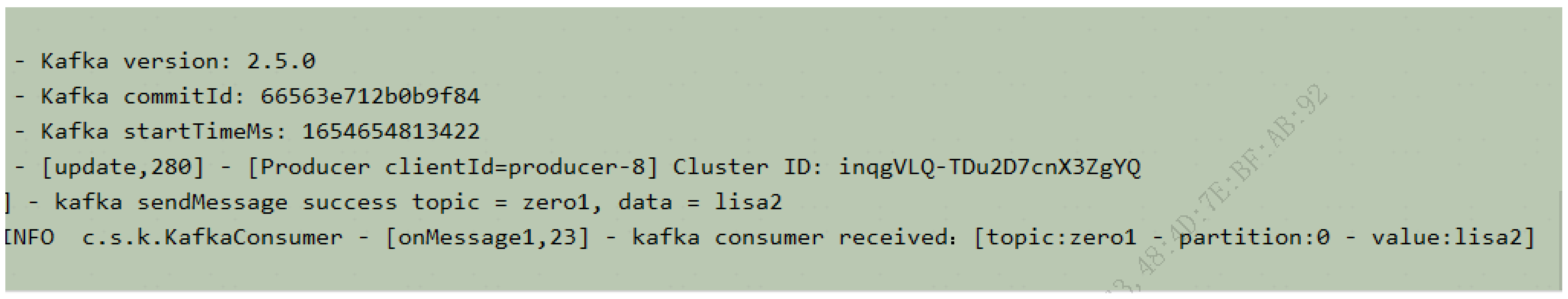


```
[http-nio-8088-exec-1] INFO  o.a.k.c.u.AppInfoParser - [<init>,117] - Kafka version: 2.5.0
[http-nio-8088-exec-1] INFO  o.a.k.c.u.AppInfoParser - [<init>,118] - Kafka commitId: 66563e712b0b9f84
[http-nio-8088-exec-1] INFO  o.a.k.c.u.AppInfoParser - [<init>,119] - Kafka startTimeMs: 1654654024957
[kafka-producer-network-thread | producer-6] INFO  o.a.k.c.Metadata - [update,280] - [Producer clientId=producer-6] Cluster ID: inqgVLQ-TDu2D7cnX3ZgYQ
[http-nio-8088-exec-1] INFO  c.s.k.KafkaProducer - [sendMessage1,32] - kafka sendMessage success topic = zero1, data = lisa
[org.springframework.kafka.KafkaListenerEndpointContainer#0-0-C-1] INFO  c.s.k.KafkaConsumer - [onMessage1,23] - kafka consumer received: [topic:zero1 - partition:0 - value:lisa]
```



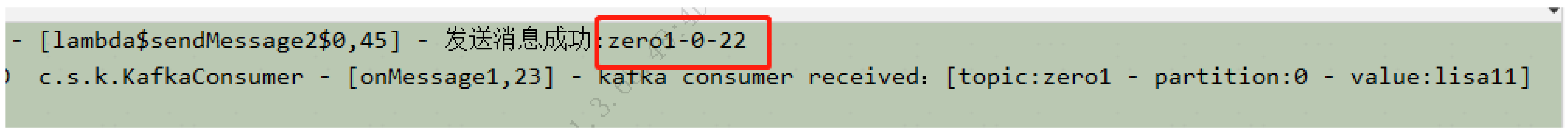
```
root@4432dfb12cf7:/opt/kafka_2.13-2.8.1/bin# ./kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic zero1 --from-beginning
123
hello
123
456
456
456
456
456
456
456
456
456
456
456
456
789
789
789
lisa
```

```
2、消费者手动ACK
//spring.kafka.consumer.enable-auto-commit=false
//spring.kafka.listener.ack-mode=manual_immediate
@KafkaListener(topics = "#{kafkaTopicName}", groupId = "#{topicGroupId}")
public void onMessage1(ConsumerRecord<String, Object> record, Acknowledgment ack) {
    log.info("kafka consumer received: [topic:{} - partition:{} - value:{}]", record.topic(), record.partition(), record.value());
    //手动提交offset
    ack.acknowledge();
}
```



```
- Kafka version: 2.5.0
- Kafka commitId: 66563e712b0b9f84
- Kafka startTimeMs: 1654654813422
- [update,280] - [Producer clientId=producer-8] Cluster ID: inqgVLQ-TDu2D7cnX3ZgYQ
] - kafka sendMessage success topic = zero1, data = lisa2
INFO  c.s.k.KafkaConsumer - [onMessage1,23] - kafka consumer received: [topic:zero1 - partition:0 - value:lisa2]
```

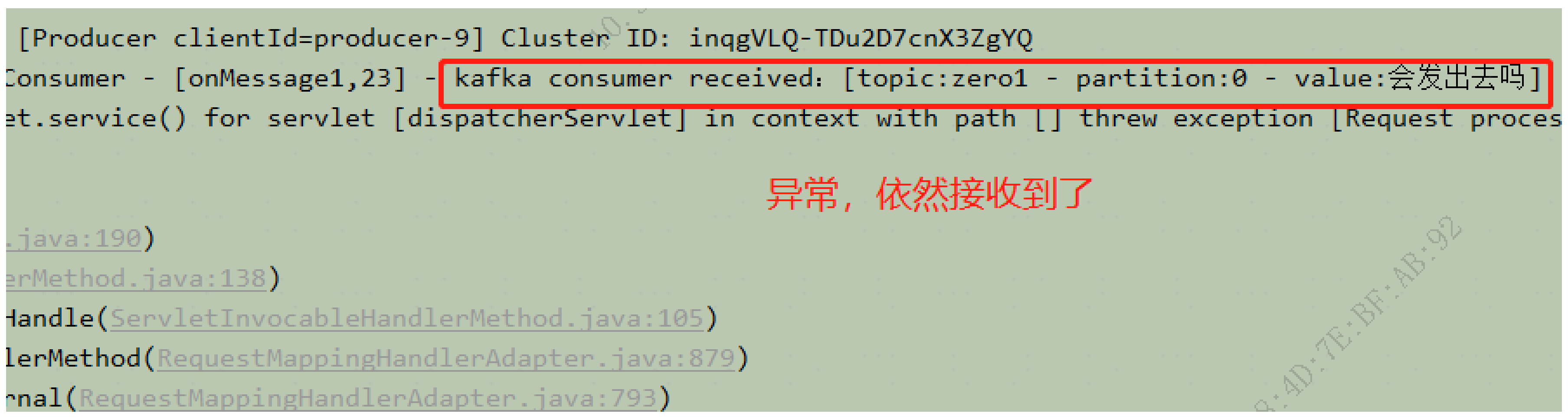
```
3、生产者-回调异步发送
@GetMapping("send2/{message}")
public void sendMessage2(@PathVariable("message") String message) {
    kafkaTemplate.send(KFK_TOPIC_ZERO1, message).addCallback(success -> {
        // 消息发送到的topic
        String topic = success.getRecordMetadata().topic();
        // 消息发送到的分区
        int partition = success.getRecordMetadata().partition();
        // 消息在分区内的offset
        long offset = success.getRecordMetadata().offset();
        log.info("发送消息成功:" + topic + "-" + partition + "-" + offset);
    }, failure -> {
        log.info("发送消息失败:" + failure.getMessage());
    });
}
或
@GetMapping("send3/{message}")
public void sendMessage3(@PathVariable("message") String message) {
    ListenableFuture<SendResult<String, Object>> future = kafkaTemplate.send(KFK_TOPIC_ZERO, message);
    future.addCallback(new ListenableFutureCallback<SendResult<String, Object>>() {
        @Override
        public void onFailure(Throwable ex) {
            log.error("发送消息失败, ex = {}, topic = {}, data = {}", ex, KFK_TOPIC_ZERO, message);
        }
        @Override
        public void onSuccess(SendResult<String, Object> result) {
            log.info("发送消息成功, topic = {}, data = {}", result.getRecordMetadata().topic(), message);
        }
    });
}
```



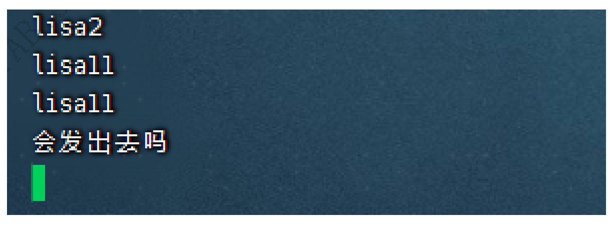
```
- [lambda$sendMessage2$0,45] - 发送消息成功:zero1-0-22
)  c.s.k.KafkaConsumer - [onMessage1,23] - kafka consumer received: [topic:zero1 - partition:0 - value:lisa11]
```

```
4.生产者-事务
@GetMapping("send4/{message}")
public void sendMessage4(@PathVariable("message") String message){
    // 不声明事务：后面报错但前面消息已经发送成功了
    kafkaTemplate.send(KFK_TOPIC_ZERO1, message);
    throw new RuntimeException("fail");
}

//spring.kafka.producer.transaction-id-prefix=kfk_tx_
//spring.kafka.producer.acks=all
@GetMapping("send4/{message}")
public void sendMessage4(@PathVariable("message") String message){
    // 声明事务：后面报错消息不会发出去
    kafkaTemplate.executeInTransaction(operations -> {
        operations.send(KFK_TOPIC_ZERO1, message);
        throw new RuntimeException("fail");
    });
    log.info("异常消息, 发送成功");
}
```

//未加事务

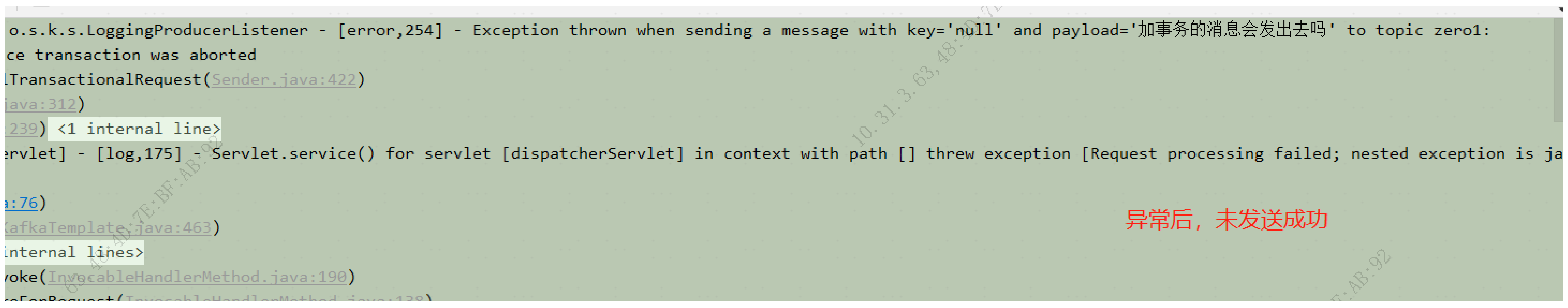


```
 [Producer clientId=producer-9] Cluster ID: inqgVLQ-TDu2D7cnX3ZgYQ
Consumer - [onMessage1,23] - kafka consumer received: [topic:zero1 - partition:0 - value:会发出去吗]
et.service() for servlet [dispatcherServlet] in context with path [] threw exception [Request proces

.java:190)
erMethod.java:138)
Handle(ServletInvocableHandlerMethod.java:105)
lerMethod(RequestMappingHandlerAdapter.java:879)
rnal(RequestMappingHandlerAdapter.java:793)
```

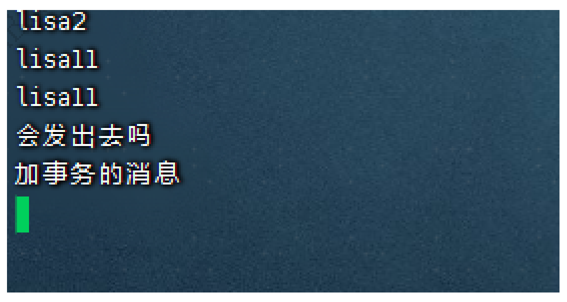


```
lisa2
lisall
lisall
会发出去吗
```

//加事务

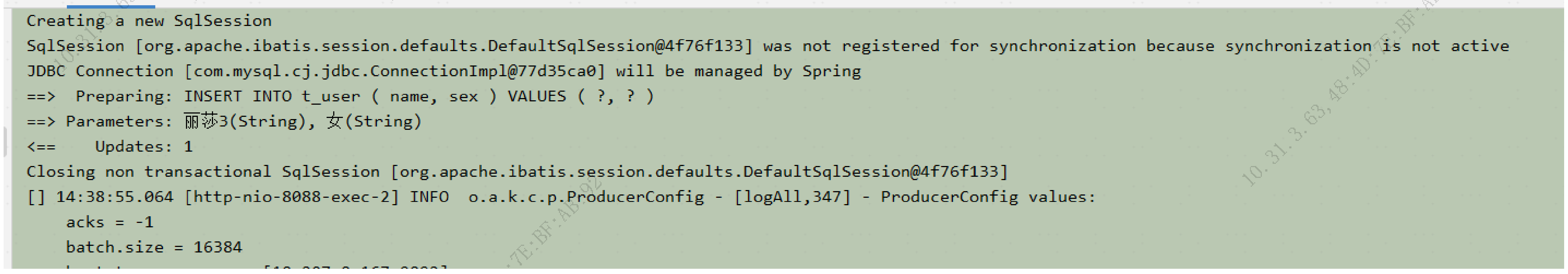


```
o.s.k.s.LoggingProducerListener - [error,254] - Exception thrown when sending a message with key='null' and payload='加事务的消息会发出去吗' to topic zero1:
ce transaction was aborted
lTransactionalRequest(Sender.java:422)
java:312)
239) <1 internal line>
ervlet] - [log,175] - Servlet.service() for servlet [dispatcherServlet] in context with path [] threw exception [Request processing failed; nested exception is ja

a:76)
KafkaTemplate.java:463)
internal lines>
voke(InvocableHandlerMethod.java:190)
```

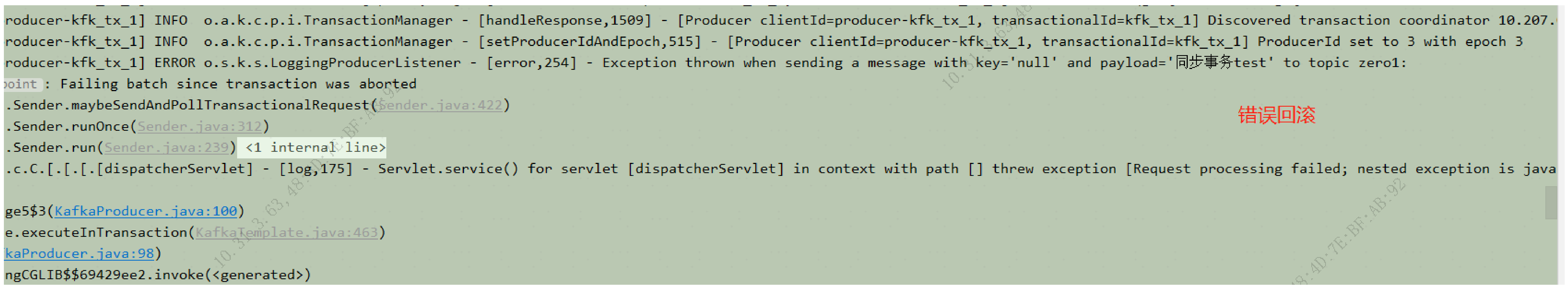


```
lisa2
lisall
lisall
会发出去吗
加事务的消息
```

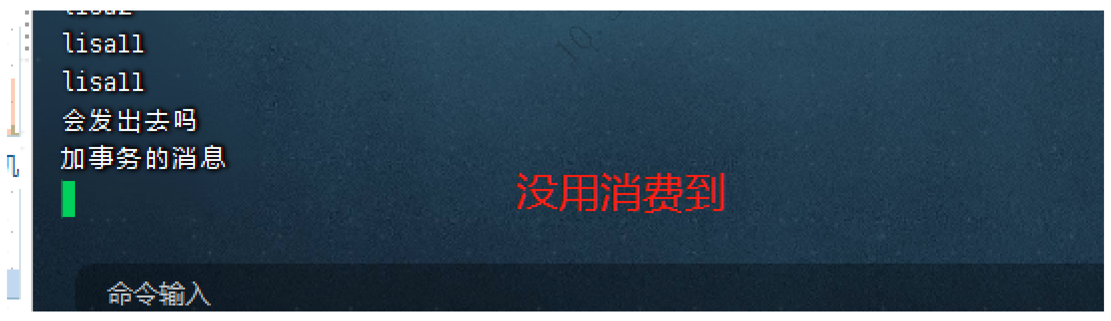
```
5.同步事务(spring + kafka)
@Transactional(transactionManager ="chainedKafkaTransactionManager", rollbackFor = Exception.class)
@GetMapping("send5/{message}")
public void sendMessage5(@PathVariable("message") String message){
    //db
    userService.addUser(new UserBean().setName("丽莎3").setSex("女"));
    //kafka
    kafkaTemplate.executeInTransaction(operations -> {
        operations.send(KFK_TOPIC_ZERO1, message);
        throw new RuntimeException("fail");
    });
    log.info("事务消息, 发送成功");
}
```



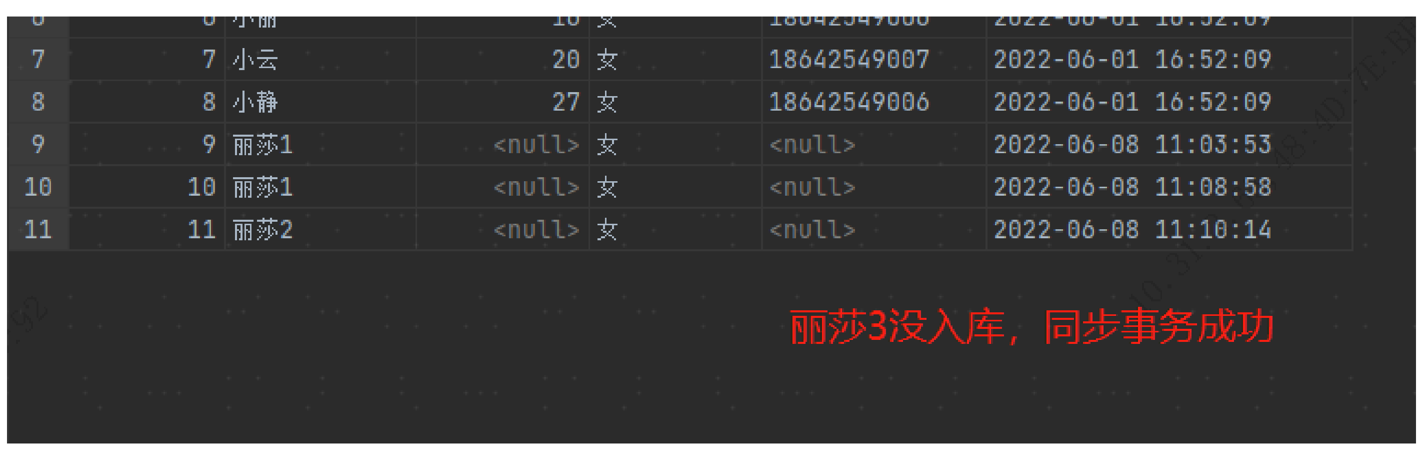
```
Creating a new SqlSession
SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@4f76f133] was not registered for synchronization because synchronization is not active
JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@77d35ca0] will be managed by Spring
==>  Preparing: INSERT INTO t_user ( name, sex ) VALUES ( ?, ? )
==> Parameters: 丽莎3(String), 女(String)
<==    Updates: 1
Closing non transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@4f76f133]
[] 14:38:55.064 [http-nio-8088-exec-2] INFO  o.a.k.c.p.ProducerConfig - [logAll,347] - ProducerConfig values:
     acks = -1
     batch.size = 16384
```



```
roducer-kfk_tx_1] INFO  o.a.k.c.p.i.TransactionManager - [handleResponse,1509] - [Producer clientId=producer-kfk_tx_1, transactionalId=kfk_tx_1] Discovered transaction coordinator 10.207.
roducer-kfk_tx_1] INFO  o.a.k.c.p.i.TransactionManager - [setProducerIdAndEpoch,515] - [Producer clientId=producer-kfk_tx_1, transactionalId=kfk_tx_1] ProducerId set to 3 with epoch 3
roducer-kfk_tx_1] ERROR o.s.k.s.LoggingProducerListener - [error,254] - Exception thrown when sending a message with key='null' and payload='同步事务test' to topic zero1:
oint : Failing batch since transaction was aborted
.Sender.maybeSendAndPollTransactionalRequest(Sender.java:422)
.Sender.runOnce(Sender.java:312)
.Sender.run(Sender.java:239) <1 internal line>
.c.C.[.[.[.[dispatcherServlet] - [log,175] - Servlet.service() for servlet [dispatcherServlet] in context with path [] threw exception [Request processing failed; nested exception is java

ge5$3(KafkaProducer.java:100)
e.executeInTransaction(KafkaTemplate.java:463)
kaProducer.java:98)
ngCGLIB$$69429ee2.invoke(<generated>)
```

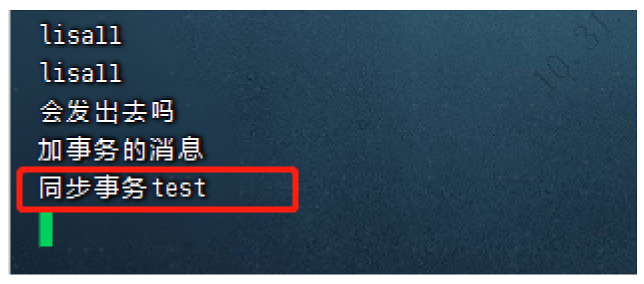


```
lisall
lisall
会发出去吗
加事务的消息
```

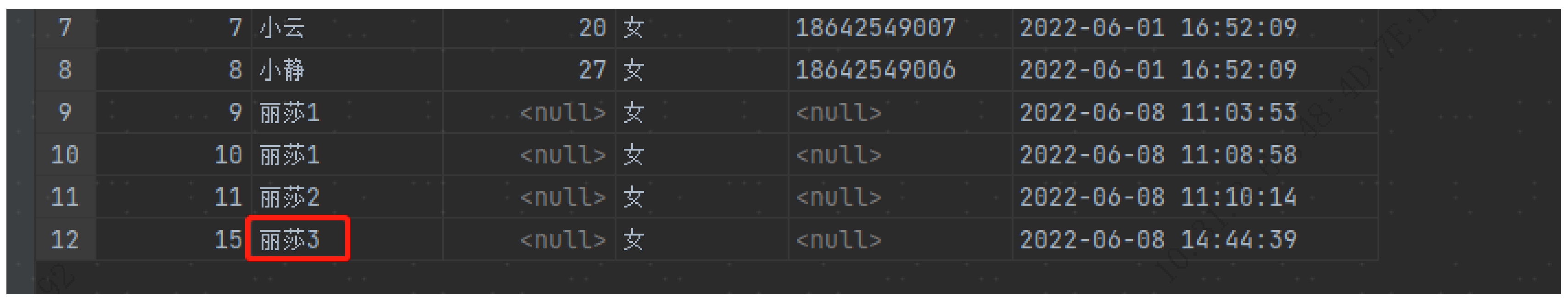


7	7	小云	20	女	18642549007	2022-06-01 16:52:09
8	8	小静	27	女	18642549006	2022-06-01 16:52:09
9	9	丽莎1	<null>	女	<null>	2022-06-08 11:03:53
10	10	丽莎1	<null>	女	<null>	2022-06-08 11:08:58
11	11	丽莎2	<null>	女	<null>	2022-06-08 11:10:14

//注释 throw new RuntimeException("fail")

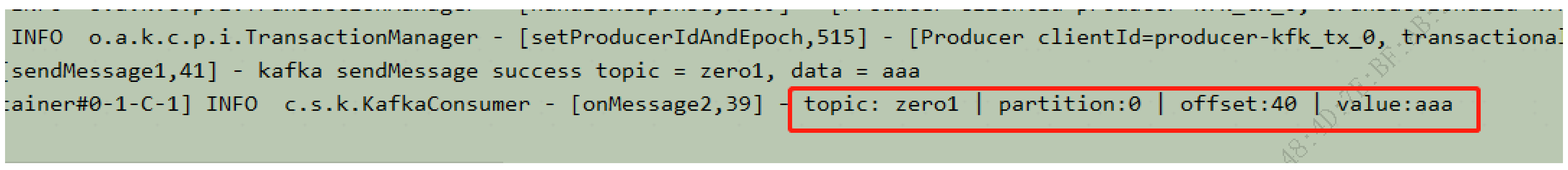


```
lisall
lisall
会发出去吗
加事务的消息
同步事务test
```



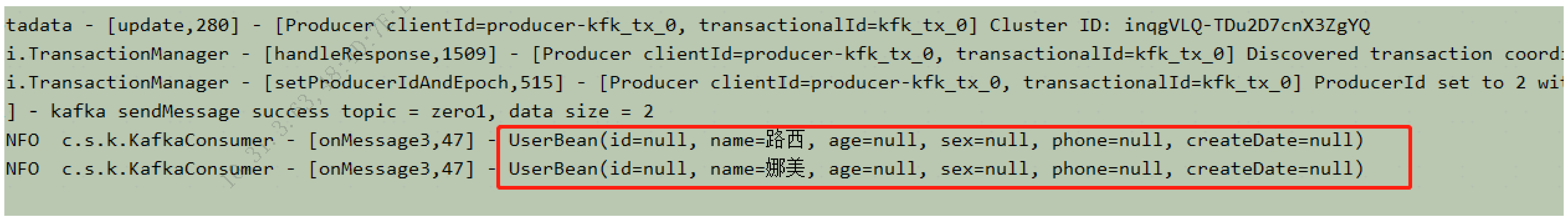
7	7	小云	20	女	18642549007	2022-06-01 16:52:09
8	8	小静	27	女	18642549006	2022-06-01 16:52:09
9	9	丽莎1	<null>	女	<null>	2022-06-08 11:03:53
10	10	丽莎1	<null>	女	<null>	2022-06-08 11:08:58
11	11	丽莎2	<null>	女	<null>	2022-06-08 11:10:14
12	15	丽莎3	<null>	女	<null>	2022-06-08 14:44:39

```
6.指定颗粒度消费
//指定topic、partition、offset消费
//同时监听topic1和topic2，监听topic1的0号分区、topic2的 "0号和1号" 分区，指向1号分区的offset初始值为8
@KafkaListener(id = "", groupId = "zeroGroupId", topicPartitions = {
    @TopicPartition(topic = "zero", partitions = {"0"}),
    @TopicPartition(
            topic = "zero1",
            partitions = "0",
            partitionOffsets = @PartitionOffset(partition = "1", initialOffset = "8")
    )
})
public void onMessage2(ConsumerRecord<?, ?> record) {
    log.info("topic: {} | partition:{} | offset:{} | value:{}", record.topic(), record.partition(), record.offset(), record.value());
}
```



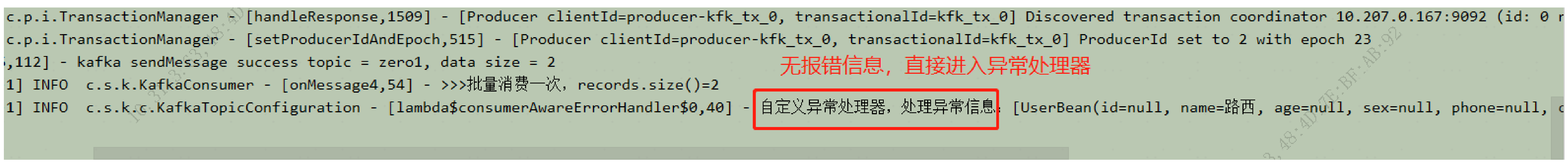
```
INFO  o.a.k.c.p.i.TransactionManager - [setProducerIdAndEpoch,515] - [Producer clientId=producer-kfk_tx_0, transactiona
sendMessage1,41] - kafka sendMessage success topic = zero1, data = aaa
ainer#0-1-C-1] INFO  c.s.k.KafkaConsumer - [onMessage2,39] - topic: zero1 | partition:0 | offset:40 | value:aaa
```

```
7.发送list数据
//spring.kafka.producer.value-serializer=com.sf.kafka.serialization.ObjectSerializer
//spring.kafka.consumer.value-serializer=com.sf.kafka.serialization.ObjectSerializer

// 批量发送
@Transactional
@GetMapping("send6/list")
public void sendMessage6() throws Exception {
    List<UserBean> list = Arrays.asList(new UserBean().setName("路西"), new UserBean().setName("娜美"));
    kafkaTemplate.send(KFK_TOPIC_ZERO1, list).get(2, TimeUnit.SECONDS);
    log.info("kafka sendMessage success topic = {}, data size = {}", KFK_TOPIC_ZERO1, list.size());
}

//List来接收
@KafkaListener(groupId = "zeroGroupId", topics = "zero1")
public void onMessage3(List<UserBean> list) {
    for (UserBean record : list) {
        log.info(record.toString());
    }
}
```



```
tadata - [update,280] - [Producer clientId=producer-kfk_tx_0, transactionalId=kfk_tx_0] Cluster ID: inqgVLQ-TDu2D7cnX3ZgYQ
i.TransactionManager - [handleResponse,1509] - [Producer clientId=producer-kfk_tx_0, transactionalId=kfk_tx_0] Discovered transaction coord
i.TransactionManager - [setProducerIdAndEpoch,515] - [Producer clientId=producer-kfk_tx_0, transactionalId=kfk_tx_0] ProducerId set to 2 wi
] - kafka sendMessage success topic = zero1, data size = 2
NFO  c.s.k.KafkaConsumer - [onMessage3,47] - UserBean(id=null, name=路西, age=null, sex=null, phone=null, createDate=null)
NFO  c.s.k.KafkaConsumer - [onMessage3,47] - UserBean(id=null, name=娜美, age=null, sex=null, phone=null, createDate=null)
```

```
8.异常降级
//异常处理器
@Bean
public ConsumerAwareListenerErrorHandler consumerAwareErrorHandler() {
    return (message, exception, consumer) -> {
        log.info("自定义异常处理器，处理异常信息：" + message.getPayload());
        return null;
    };
}

//指定异常处理
@KafkaListener(topics = "zero1", errorHandler = "consumerAwareErrorHandler")
public void onMessage4(List<ConsumerRecord<?, ?>> records) {
    log.info(">>>批量消费一次，records.size()=" + records.size());
    for (ConsumerRecord<?, ?> record : records) {
        log.info(record.value().toString());
    }
}
```



```
c.p.i.TransactionManager - [handleResponse,1509] - [Producer clientId=producer-kfk_tx_0, transactionalId=kfk_tx_0] Discovered transaction coordinator 10.207.0.167:9092 (id: 0 r
c.p.i.TransactionManager - [setProducerIdAndEpoch,515] - [Producer clientId=producer-kfk_tx_0, transactionalId=kfk_tx_0] ProducerId set to 2 with epoch 23
,112] - kafka sendMessage success topic = zero1, data size = 2
1] INFO  c.s.k.KafkaConsumer - [onMessage4,54] - >>>批量消费一次，records.size()=2
1] INFO  c.s.k.c.KafkaTopicConfiguration - [lambda$consumerAwareErrorHandler$0,40] - 自定义异常处理器，处理异常信息：[UserBean(id=null, name=路西, age=null, sex=null, phone=null, c
```

【异常】

问题：org.apache.kafka.clients.consumer.ConsumerRecord; nested exception is org.springframework.core.convert.ConversionFailedException: Failed to convert from type [java.util.ArrayList<?>]
to type[org.apache.kafka.clients.consumer.ConsumerRecord<?, ?>] for value '[789]';

解决：关闭批量消费，注释掉 spring.kafka.consumer.listener.type=batch

问题：Producer factory does not support transactions | Must set acks to all in order to use the idempotent producer. Otherwise we cannot guarantee idempotence.

解决：开启事务，spring.kafka.producer.transaction-id-prefix=kfk_tx   spring.kafka.producer.acks=all

问题：Caused by: java.lang.ClassNotFoundException: org.springframework.data.transaction.ChainedTransactionManager

解决：添加依赖，spring-data-commons

问题：No transaction is in process; possible solutions: run the template operation within the scope of a template.executeInTransaction() operation, start a transaction with @Transactional before invoking the template method

解决：开启事务后，要么使用 kafkaTemplate.executeInTransaction(todo...) 执行，要么在方法上添加事务注解 @Transactional

问题：java.lang.ClassCastException: java.util.Arrays$ArrayList cannot be cast to java.lang.String

解决：配置的序列化器为StringSerializer，无法将List转String。自定义Object序列号器，并在配置里指定