Why Custom Serializer

If you are coming from a database background, you can think of a topic like a table and each message sent to the topic like a record. Those records are not always just a single string or a number. Normally, we have multiple columns in a record. So, when working with Kafka, we need to be able to send a record of multiple columns.

Similarly, if you are coming from an object-oriented programming background, you will see Kafka message as an object. And normally, these objects will have multiple fields and methods. We should be able to send these objects to Kafka as a message. Sending simple strings to Kafka may fulfil some requirements.

But in a complex condition, you may need to send custom objects, for example, A supplier object or an invoice object. If you want to send such custom objects or a row like structure, you need to implement a custom serializer and a deserializer.

To understand the idea of serializer and deserializer, we need to create an example.In this example, we will do following things.

1. Create a Supplier class. We will serialize the supplier class and send the supplier object as a message to Kafka.
2. Create a Kafka producer. This producer will send supplier object as a Kafka record. Earlier we were sending strings, but in this example, we are going to push an object instead of a simple string.
3. Create a serializer to convert a supplier object into a byte array.
4. Create a deserializer to convert a byte array back into a supplier object. Kafka doesn't know how to serialize and deserialize our object, and so we must create a serializer and a deserializer.
5. Create a consumer. Finally, we will create a consumer that will read supplier objects from Kafka and just print the details on the console.

The serializer class is *SupplierSerializer* and it implements *Serializer* interface and sets the generic type as *Supplier*. This interface is defined under Kafka common package. As per this interface, we need to override three methods.

1. Configure
2. Serialize
3. Close

The main action is happening in the serialize method. The code is straightforward. If the data is null, we return null because we have nothing to serialize. We simply convert supplier name and supplier start date into UTF8 bytes. Then we allocate a byte buffer and encode everything into the byte buffer. Since we will need to know the length of supplier name and supplier date strings at the time of deserialization, we also encode their sizes into the byte buffer. Finally, we return the byte buffer array. That's it. Done. That's what the serialization means, convert your object into bytes, and that's what we have done in the above example.

```
package com.jpmc;
import java.util.Date;
public class Supplier{
    private int supplierId;
    private String supplierName;
    private Date supplierStartDate;

    public Supplier(int id, String name, Date dt){
        this.supplierId = id;
        this.supplierName = name;
        this.supplierStartDate = dt;
    }

    public int getID(){
        return supplierId;
    }

    public String getName(){
        return supplierName;
    }

    public Date getStartDate(){
        return supplierStartDate;
    }
}

package com.jpmc;
import java.util.*;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.ConsumerRecord;

public class SupplierConsumer{

    public static void main(String[] args) throws Exception{

        String topicName = "SupplierTopic";
        String groupName = "SupplierTopicGroup";

        Properties props = new Properties();
        props.put("bootstrap.servers", "localhost:9092,localhost:9093");
        props.put("group.id", groupName);
        props.put("key.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");
        props.put("value.deserializer", "com.jpmc.SupplierDeserializer");


        KafkaConsumer<String, Supplier> consumer = new KafkaConsumer<>(props);
        consumer.subscribe(Arrays.asList(topicName));

        while (true){
            ConsumerRecords<String, Supplier> records = consumer.poll(100);
            for (ConsumerRecord<String, Supplier> record : records){
                System.out.println("Supplier id= " + String.valueOf(record.value().getID()) +
" Supplier Name = " + record.value().getName() + " Supplier Start Date = " +
record.value().getStartDate().toString());
            }
        }
        }
}

package com.jpmc;
import java.nio.ByteBuffer;
import java.util.Date;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import org.apache.kafka.common.errors.SerializationException;
import org.apache.kafka.common.serialization.Deserializer;
import java.io.UnsupportedEncodingException;
import java.util.Map;

public class SupplierDeserializer implements Deserializer<Supplier> {
    private String encoding = "UTF8";

    public void configure(Map<String, ?> configs, boolean isKey) {
                //Nothing to configure
        }

    public Supplier deserialize(String topic, byte[] data) {

        try {
            if (data == null){
                System.out.println("Null recieved at deserialize");
                                return null;
                                }
            ByteBuffer buf = ByteBuffer.wrap(data);
            int id = buf.getInt();
            int sizeOfName = buf.getInt();
            byte[] nameBytes = new byte[sizeOfName];
            buf.get(nameBytes);
            String deserializedName = new String(nameBytes, encoding);

            int sizeOfDate = buf.getInt();
            byte[] dateBytes = new byte[sizeOfDate];
            buf.get(dateBytes);
            String dateString = new String(dateBytes,encoding);

            DateFormat df = new SimpleDateFormat("EEE MMM dd HH:mm:ss Z yyyy");

            return new Supplier(id,deserializedName,df.parse(dateString));



        } catch (Exception e) {
            throw new SerializationException("Error when deserializing byte[] to Supplier");
        }
    }

    public void close() {
        // nothing to do
    }
}

package com.jpmc;
import java.util.*;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import org.apache.kafka.clients.producer.*;
public class SupplierProducer {

  public static void main(String[] args) throws Exception{

    String topicName = "SupplierTopic";

    Properties props = new Properties();
    props.put("bootstrap.servers", "localhost:9092,localhost:9093");
    props.put("key.serializer","org.apache.kafka.common.serialization.StringSerializer");
    props.put("value.serializer", "com.jpmc.SupplierSerializer");

    Producer<String, Supplier> producer = new KafkaProducer <>(props);

       DateFormat df = new SimpleDateFormat("yyyy-MM-dd");
       Supplier sp1 = new Supplier(101,"Xyz Pvt Ltd.",df.parse("2016-04-01"));
       Supplier sp2 = new Supplier(102,"Abc Pvt Ltd.",df.parse("2012-01-01"));

      producer.send(new ProducerRecord<String,Supplier>(topicName,"SUP",sp1)).get();
      producer.send(new ProducerRecord<String,Supplier>(topicName,"SUP",sp2)).get();

            System.out.println("SupplierProducer Completed.");
      producer.close();

  }
}
```

```
package com.jpmc;

import org.apache.kafka.common.serialization.Serializer;

import org.apache.kafka.common.errors.SerializationException;

import java.io.UnsupportedEncodingException;

import java.util.Map;

import java.nio.ByteBuffer;


public class SupplierSerializer implements Serializer<Supplier> {

   private String encoding = "UTF8";


   @Override

   public void configure(Map<String, ?> configs, boolean isKey) {

            // nothing to configure

   }


   @Override

   public byte[] serialize(String topic, Supplier data) {


            int sizeOfName;

            int sizeOfDate;

            byte[] serializedName;

            byte[] serializedDate;


      try {

         if (data == null)

            return null;

                     serializedName = data.getName().getBytes(encoding);

                        sizeOfName = serializedName.length;

                        serializedDate = data.getStartDate().toString().getBytes(encoding);
                        sizeOfDate = serializedDate.length;


                        ByteBuffer buf = ByteBuffer.allocate(4+4+sizeOfName+4+sizeOfDate);
                        buf.putInt(data.getID());
                        buf.putInt(sizeOfName);
                        buf.put(serializedName);
                        buf.putInt(sizeOfDate);
                        buf.put(serializedDate);



                return buf.array();


            } catch (Exception e) {
                throw new SerializationException("Error when serializing Supplier to byte[]");
            }
        }


        @Override
        public void close() {
            // nothing to do
        }
    }
```