How to use Spring Kafka JsonSerializer (JsonDeserializer) to produce/consume Java Object messages

In the previous post, we had setup a **Spring Kafka Application** succesfully by explicitly configuration **Kafka Factories** with **SpringBoot**. But the messages had been used have **String** type. While in the development, **POJO** (Plain Old Java Object) are often used to construct messages. So with the tutorial, **JavaSampleApproach** will show how to use **Spring Kafka JsonSerializer** (**JsonDeserializer**) to produce/consume Java Object messages.

Related Articles:
– How to start Spring Kafka Application with Spring Boot
– How to start Spring Apache Kafka Application with SpringBoot Auto-Configuration

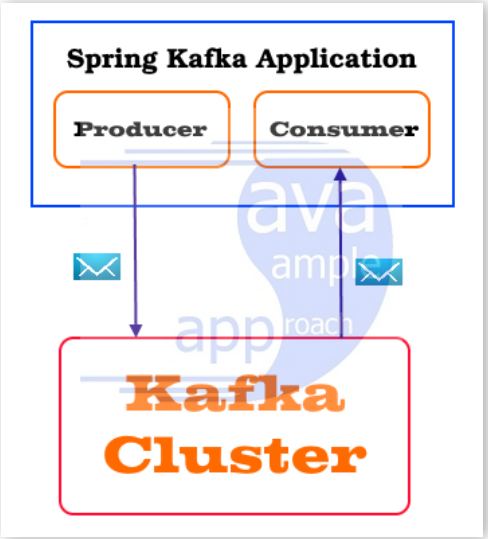
Contents [hide]

I. Technologies
II. Overview
III. Practice
1. Create a SpringBoot project
2. Create Customer model
3. Create Kafa Factories (ProducerFactory & ConsumerFactory)
3.1 Create ProducerFactory and KafkaTemplate
3.2 Create ConsumerFactory and KafkaListenerContainerFactory
4. Create Services (Producer and Consumer)
5. Implement Client
6. Deployment
IV. Sourcecode

I. Technologies

– Java 8
– Maven build
– Spring Boot
– Apache Kafka
– Spring Tool Suite editor

II. Overview



In the tutorial, we send and receive Java object messages to/from **Apache Kafka**, so **ProducerFactory** uses `JsonSerializer.class` and **ConsumerFactory** uses `JsonDeserializer.class` to serialize/deserialize Java objects to Json bytes.

– **KafkaProducerConfig**:

```
@Bean
public ProducerFactory<String, Customer> producerFactory() {
    ...
    configProps.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, JsonSerializer.class);
    return new DefaultKafkaProducerFactory<>(configProps);
}

@Bean
public KafkaTemplate<String, Customer> kafkaTemplate() {
    return new KafkaTemplate<>(producerFactory());
}
```

– **KafkaConsumerConfig**:

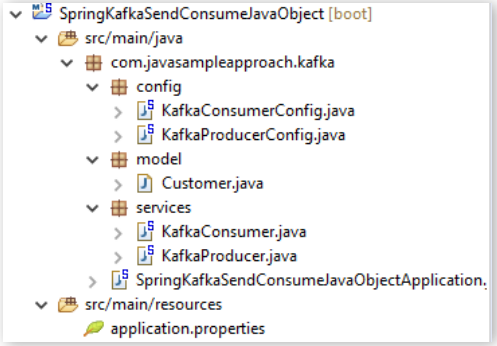
```
@Bean
public ConsumerFactory<String, Customer> consumerFactory() {
    ...
    props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, JsonDeserializer.class);
    return new DefaultKafkaConsumerFactory<>(props,
                        new StringDeserializer(),
                        new JsonDeserializer<>(Customer.class));
}

@Bean
public ConcurrentKafkaListenerContainerFactory<String, Customer> kafkaListenerContainerFactory() {
    ConcurrentKafkaListenerContainerFactory<String, Customer> factory = new ConcurrentKafkaListenerContainerFactory<>();
  ...
    return factory;
}
```

– Note: **SpringKafka** uses **Jackson library** to serialize/de-serialize Java objects to/from Json bytes so we need *jackson-databind* dependency.

III. Practice

We create a **SpringBoot** project with 2 main services: **KafkaProducer** and **KafkaConsumer** for sending and receiving messages from **Apache Kafka** cluster.



Step to do:
– Create a SpringBoot project
– Create Customer model
– Create Kafa Factories (ProducerFactory & ConsumerFactory)
– Create Services (Producer and Consumer)
– Implement Client
– Deployment

1. Create a SpringBoot project

Use **SpringToolSuite** to create a **SpringBoot** project, then add dependencies *{spring-kafka, jackson-databind}*:

```
<dependency>
  <groupId>org.springframework.kafka</groupId>
  <artifactId>spring-kafka</artifactId>
```

```
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
</dependency>
```

2. Create Customer model

```
package com.javasampleapproach.kafka.model;

public class Customer {
  private String name;
  private int age;

  public Customer(){
  }

  public Customer(String name, int age){
    this.name = name;
    this.age = age;
  }

  public void setName(String name){
    this.name = name;
  }

  public String getName(){
    return this.name;
  }

  public void setAge(int age){
    this.age = age;
  }

  public int getAge(){
    return this.age;
  }

  public String toString(){
    String info = String.format("{ 'name': %s, 'age': %d}", name, age);
    return info;
  }
}
```

3. Create Kafa Factories (ProducerFactory & ConsumerFactory)

Open **application.properties**, add kafka configuration:

```
jsa.kafka.bootstrap-servers=localhost:9092
jsa.kafka.consumer.group-id=jsa-group
jsa.kafka.topic=jsa-kafka-topic
```

- `jsa.kafka.bootstrap-servers` is used to indicate the **Kafka Cluster** address.
- `jsa.kafka.consumer.group-id` is used to indicate the **consumer-group-id**.
- `jsa.kafka.topic` is used to define a Kafka topic name to produce and receive messages.

3.1 Create ProducerFactory and KafkaTemplate

```
package com.javasampleapproach.kafka.config;

import java.util.HashMap;
import java.util.Map;

import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.common.serialization.StringSerializer;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.core.DefaultKafkaProducerFactory;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.kafka.core.ProducerFactory;
import org.springframework.kafka.support.serializer.JsonSerializer;

import com.javasampleapproach.kafka.model.Customer;


@Configuration
public class KafkaProducerConfig {

  @Value("${jsa.kafka.bootstrap-servers}")
  private String bootstrapServer;

  @Bean
  public ProducerFactory<String, Customer> producerFactory() {
      Map<String, Object> configProps = new HashMap<>();
      configProps.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServer);
      configProps.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
      configProps.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, JsonSerializer.class);
```

```
        return new DefaultKafkaProducerFactory<>(configProps);
    }

    @Bean
    public KafkaTemplate<String, Customer> kafkaTemplate() {
        return new KafkaTemplate<>(producerFactory());
    }
}
```

3.2 Create ConsumerFactory and KafkaListenerContainerFactory

```
package com.javasampleapproach.kafka.config;

import java.util.HashMap;
import java.util.Map;

import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.annotation.EnableKafka;
import org.springframework.kafka.config.ConcurrentKafkaListenerContainerFactory;
import org.springframework.kafka.core.ConsumerFactory;
import org.springframework.kafka.core.DefaultKafkaConsumerFactory;
import org.springframework.kafka.support.serializer.JsonDeserializer;

import com.javasampleapproach.kafka.model.Customer;


@EnableKafka
@Configuration
public class KafkaConsumerConfig {

    @Value("${jsa.kafka.bootstrap-servers}")
    private String bootstrapServer;

    @Value("${jsa.kafka.consumer.group-id}")
    private String groupId;

    @Bean
    public ConsumerFactory<String, Customer> consumerFactory() {
        Map<String, Object> props = new HashMap<>();
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServer);
        props.put(ConsumerConfig.GROUP_ID_CONFIG, groupId);
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, JsonDeserializer.class);
        return new DefaultKafkaConsumerFactory<>(props,
                            new StringDeserializer(),
                            new JsonDeserializer<>(Customer.class));
    }

    @Bean
    public ConcurrentKafkaListenerContainerFactory<String, Customer> kafkaListenerContainerFactory() {
```

```
      ConcurrentKafkaListenerContainerFactory<String, Customer> factory = new ConcurrentKafkaListenerContainerFactory<>();
      factory.setConsumerFactory(consumerFactory());
      return factory;
  }
}
```

4. Create Services (Producer and Consumer)

– Create a **KafkaProducer** service:

```
package com.javasampleapproach.kafka.services;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Service;

import com.javasampleapproach.kafka.model.Customer;

@Service
public class KafkaProducer {
  @Autowired
  private KafkaTemplate<String, Customer> kafkaTemplate;

  @Value("${jsa.kafka.topic}")
  String kafkaTopic = "jsa-test";

  public void send(Customer customer) {
      System.out.println("sending data=" + customer);
      kafkaTemplate.send(kafkaTopic, customer);
  }
}
```

– Create a **KafkaConsumer** service:

```
package com.javasampleapproach.kafka.services;

import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Service;

import com.javasampleapproach.kafka.model.Customer;

@Service
public class KafkaConsumer {

  @KafkaListener(topics="${jsa.kafka.topic}")
    public void processMessage(Customer customer) {
    System.out.println("received content = " + customer);
    }
}
```

5. Implement Client

```
package com.javasampleapproach.kafka;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

import com.javasampleapproach.kafka.model.Customer;
import com.javasampleapproach.kafka.services.KafkaProducer;


@SpringBootApplication
public class SpringKafkaSendConsumeJavaObjectApplication implements CommandLineRunner{

  public static void main(String[] args) {
    SpringApplication.run(SpringKafkaSendConsumeJavaObjectApplication.class, args);
  }

  @Autowired
  KafkaProducer producer;

  @Override
  public void run(String... arg0) throws Exception {
    // Send Mary customer
    Customer mary = new Customer("Mary", 31);
    producer.send(mary);

    // Send Peter customer
    Customer peter = new Customer("Peter", 24);
    producer.send(peter);
  }
}
```

6. Deployment

Start **Apache Kafka Cluster:**
– Start a ZooKeeper:

```
C:\kafka_2.12-0.10.2.1>.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties
```

– Start the **Apache Kafka server**:

```
.\bin\windows\kafka-server-start.bat .\config\server.properties
```

>>> More details at: How to start Apache Kafka

Build and **Install** the SpringBoot project with commandlines: `mvn clean install` and `mvn spring-boot:run`

-> **Logs**:

```
...
```

```
sending data={ 'name': Mary, 'age': 31}
...

sending data={ 'name': Peter, 'age': 24}
...

received content = { 'name': Mary, 'age': 31}
received content = { 'name': Peter, 'age': 24}

...
```

IV. Sourcecode

SpringKafkaSendConsumeJavaObject

By grokonez | June 11, 2017.

Related Posts

- How to start Spring Kafka Application with Spring Boot
- How to start Spring Apache Kafka Application with SpringBoot Auto-Configuration
- How to start Apache Kafka
- RabbitMq Queue Durability and Persistent MessageDelivery | SpringBoot
- SpringBoot RabbitMq Exchange to Exchange Topology
- SpringBoot RabbitMq Headers Exchange
- SpringBoot RabbitMQ Topic Exchange
- SpringBoot Hazelcast cache with PostgreSQL backend
- Apache Artemis – How to produce/consume JMS messages with SpringBoot Artemis applications.
- Spring JMS ActiveMq – How to implement a runtime SpringBoot ActiveMQ JmsResponse application

Post Tags

Apache Kafka | distributed system | JsonDeserializer | JsonSerializer | messaging system | spring boot | spring kafka

1 thought on “How to use Spring Kafka JsonSerializer (JsonDeserializer) to produce/consume Java Object messages”

Nikhitha

November 20, 2018 at 12:32 pm

hi I am doing similar thing like the above code in my project, but I am using kafka not spring. Could you please assist me how to do that? would appreciate your reply

Thank you.