(/)

Serializadores personalizados en Apache Kafka

Última modificación: 29 de septiembre de 2021

> por baeldung (https://www.baeldung.com/author/baeldung/) (https://www.baeldung.com/author/baeldung/)

Primavera (https://www.baeldung.com/category/spring/) +

Kafka (https://www.baeldung.com/tag/kafka/)

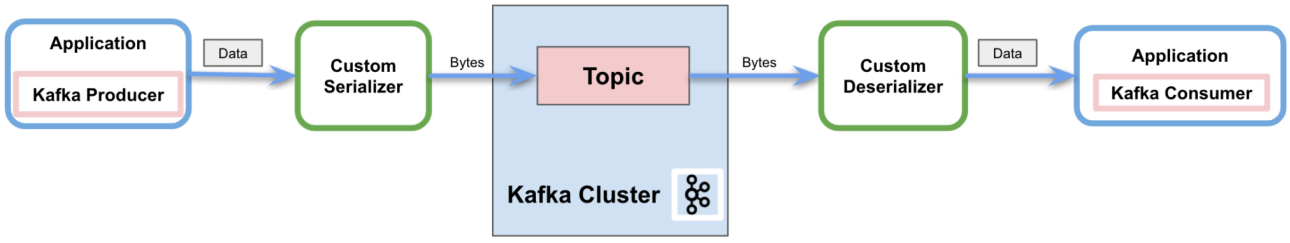
1. Introducción

Durante la transmisión de mensajes en Apache Kafka, el cliente y el servidor acuerdan el uso de un formato sintáctico común. Apache Kafka trae convertidores predeterminados (como *String* y *Long*), pero también admite serializadores personalizados para casos de uso específicos. En este tutorial, veremos cómo implementarlos.

2. Serializadores en Apache Kafka

La serialización es el proceso de convertir objetos en bytes . La deserialización es el proceso inverso: convertir un flujo de bytes en un objeto. En pocas palabras, **transforma el contenido en información legible e interpretable** .

Como mencionamos, Apache Kafka proporciona serializadores predeterminados para varios tipos básicos y nos permite implementar serializadores personalizados:



(/wp-content/uploads/2021/08/kafka1.png)

La figura anterior muestra el proceso de envío de mensajes a un tema de Kafka a través de la red. En este proceso, el serializador personalizado convierte el objeto en bytes antes de que el productor envíe el mensaje al tema. De manera similar, también muestra cómo el deserializador vuelve a transformar los bytes en el objeto para que el consumidor lo procese correctamente.

2.1. Serializadores personalizados

Apache Kafka proporciona un serializador y deserializador prediseñado para varios tipos básicos:

- ***StringSerializer*** **(https://kafka.apache.org/24/javadoc/org/apache/kafka/common/serialization/StringSerializer.html)**
- ***ShortSerializer*** **(https://kafka.apache.org/24/javadoc/org/apache/kafka/common/serialization/ShortSerializer.html)**
- ***IntegerSerializer*** **(https://kafka.apache.org/24/javadoc/org/apache/kafka/common/serialization/IntegerSerializer.html)**
- ***LongSerializer*** **(https://kafka.apache.org/24/javadoc/org/apache/kafka/common/serialization/LongSerializer.html)**
- ***DoubleSerializer*** **(https://kafka.apache.org/24/javadoc/org/apache/kafka/common/serialization/DoubleSerializer.html)**
- ***BytesSerializer*** **(https://kafka.apache.org/24/javadoc/org/apache/kafka/common/serialization/BytesSerializer.html)**

Pero también ofrece la capacidad de implementar (des) serializadores personalizados. Para serializar nuestros propios objetos, implementaremos la interfaz *Serializer*

(https://kafka.apache.org/24/javadoc/org/apache/kafka/common/serialization/Serializer.html) . De manera similar, para crear un deserializador personalizado, implementaremos la interfaz *Deserializer* (https://kafka.apache.org/24/javadoc/org/apache/kafka/common/serialization/Deserializer.html) .

Hay métodos disponibles para anular para ambas interfaces:

- *configure* : se utiliza para implementar detalles de configuración
- *serializar / deserializar* : **estos métodos incluyen la implementación real de nuestra serialización y deserialización personalizadas** .
- *cerrar* : use este método para cerrar la sesión de Kafka

3. Implementación de serializadores personalizados en Apache Kafka

Apache Kafka proporciona la capacidad de personalizar los serializadores . Es posible implementar convertidores específicos no solo para el valor del mensaje sino también para la clave.

3.1. Dependencias

Para implementar los ejemplos, simplemente agregaremos la dependencia de la API del consumidor de Kafka (https://search.maven.org/classic/#search%7Cgav%7C1%7Cg%3A%22org.apache.kafka%22%20AND%20a%3A%22kafka-clients%22) a nuestro *pom.xml* :

```
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>2.8.0</version>
</dependency>
```

3.2. Serializador personalizado

Primero, usaremos Lombok (/intro-to-project-lombok) para especificar el objeto personalizado para enviar a través de Kafka:

```
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class MessageDto {
    private String message;
    private String version;
}
```


A continuación, implementaremos la interfaz *Serializer (https://kafka.apache.org/24/javadoc/org/apache/kafka/common/serialization/Serializer.html)* proporcionada por Kafka para que el productor envíe los mensajes:

```
@Slf4j
public class CustomSerializer implements Serializer {
    private final ObjectMapper objectMapper = new ObjectMapper();

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

    @Override
    public byte[] serialize(String topic, MessageDto data) {
        try {
            if (data == null){
                System.out.println("Null received at serializing");
                return null;
            }
            System.out.println("Serializing...");
            return objectMapper.writeValueAsBytes(data);
        } catch (Exception e) {
            throw new SerializationException("Error when serializing
MessageDto to byte[]");
        }
    }

    @Override
    public void close() {
    }
}
```

Anularemos el método de *serialización* de la interfaz . Por lo tanto, en nuestra implementación, transformaremos el objeto personalizado usando un *ObjectMapper de Jackson* . Luego, devolveremos el flujo de bytes para enviar correctamente el mensaje a la red.

3.3. Deserializador personalizado

De la misma forma, implementaremos la interfaz *Deserializer* (https://kafka.apache.org/24/javadoc/org/apache/kafka/common/serialization/Deserializer.html) para el consumidor:

```
@Slf4j
public class CustomDeserializer implements Deserializer<MessageDto> {
    private ObjectMapper objectMapper = new ObjectMapper();

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

    @Override
    public MessageDto deserialize(String topic, byte[] data) {
        try {
            if (data == null){
                System.out.println("Null received at deserializing");
                return null;
            }
            System.out.println("Deserializing...");
            return objectMapper.readValue(new String(data, "UTF-8"),
MessageDto.class);
        } catch (Exception e) {
            throw new SerializationException("Error when deserializing
byte[] to MessageDto");
        }
    }

    @Override
    public void close() {
    }
}
```

Como en la sección anterior, **reemplazaremos el método *deserializar* de la interfaz** . En consecuencia, convertiremos el flujo de bytes en el objeto personalizado utilizando el mismo *Jackson ObjectMapper* .

3.4. Consumir un mensaje de ejemplo

Veamos un ejemplo de trabajo enviando y recibiendo un mensaje de ejemplo con el serializador y deserializador personalizados.

En primer lugar, crearemos y configuraremos Kafka Producer:

```
private static KafkaProducer<String, MessageDto> createKafkaProducer() {
    Properties props = new Properties();
    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
kafka.getBootstrapServers());
    props.put(ProducerConfig.CLIENT_ID_CONFIG, CONSUMER_APP_ID);
    props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
"org.apache.kafka.common.serialization.StringSerializer");
    props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
"com.baeldung.kafka.serdes.CustomSerializer");

    return new KafkaProducer(props);
}
```

Configuraremos la propiedad del serializador de valor con nuestra clase personalizada y el serializador de claves con el *StringSerializer* predeterminado .

En segundo lugar, crearemos el consumidor de Kafka:

```
private static KafkaConsumer<String, MessageDto> createKafkaConsumer() {
    Properties props = new Properties();
    props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
kafka.getBootstrapServers());
    props.put(ConsumerConfig.CLIENT_ID_CONFIG, CONSUMER_APP_ID);
    props.put(ConsumerConfig.GROUP_ID_CONFIG, CONSUMER_GROUP_ID);
    props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
    props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
"org.apache.kafka.common.serialization.StringDeserializer");
    props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
"com.baeldung.kafka.serdes.CustomDeserializer");

    return new KafkaConsumer<>(props);
}
```

Además de los deserializadores de clave y valor con nuestra clase personalizada , es obligatorio incluir la identificación del grupo. Aparte de eso, colocamos la configuración de reinicio de compensación automática al

principio para asegurarnos de que el productor envíe todos los mensajes antes de que comience el consumidor.

Una vez que hemos creado los clientes productores y consumidores, es hora de enviar un mensaje de ejemplo:

```
MessageDto msgProd =
MessageDto.builder().message("test").version("1.0").build();

KafkaProducer<String, MessageDto> producer = createKafkaProducer();
producer.send(new ProducerRecord<String, MessageDto>(TOPIC, "1",
msgProd));
System.out.println("Message sent " + msgProd);
producer.close();
```

Y podemos recibir el mensaje con el consumidor suscribiéndonos al tema:

```
AtomicReference<MessageDto> msgCons = new AtomicReference<>();

KafkaConsumer<String, MessageDto> consumer = createKafkaConsumer();
consumer.subscribe(Arrays.asList(TOPIC));

ConsumerRecords<String, MessageDto> records =
consumer.poll(Duration.ofSeconds(1));
records.forEach(record -> {
    msgCons.set(record.value());
    System.out.println("Message received " + record.value());
});

consumer.close();
```

El resultado en la consola es:

(https://freestar.com/?

```
Serializing...
Message sent MessageDto(message=test, version=1.0)
Deserializing...
Message received MessageDto(message=test, version=1.0)
```

4. Conclusión

En este tutorial, mostramos cómo los productores usan serializadores en Apache Kafka para enviar mensajes a través de la red. De la misma forma, también mostramos cómo los consumidores utilizan deserializadores para interpretar el mensaje recibido.

Además, aprendimos los serializadores predeterminados disponibles y, lo que es más importante, la capacidad de implementar serializadores y deserializadores personalizados.

Como siempre, el código está disponible en GitHub (https://github.com/eugenp/tutorials/tree/master/apache-kafka) .

¡Los comentarios están cerrados sobre este artículo!

CURSOS

TODOS LOS CURSOS (/ALL-COURSES)
TODOS LOS CURSOS A GRANEL (/ALL-BULK-COURSES)
LA PLATAFORMA DE CURSOS (HTTPS://COURSES.BAELDUNG.COM)

SERIE

TUTORIAL "VOLVER A LO BÁSICO" DE JAVA (/JAVA-TUTORIAL)
TUTORIAL DE JACKSON JSON (/JACKSON)
TUTORIAL DE HTTPCLIENT 4 (/HTTPCLIENT-GUIDE)
DESCANSO CON SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
TUTORIAL DE PERSISTENCIA DE PRIMAVERA (/PERSISTENCE-WITH-SPRING-SERIES)
SEGURIDAD CON SPRING (/SECURITY-SPRING)
TUTORIALES REACTIVOS DE PRIMAVERA (/SPRING-REACTIVE-GUIDE)

SOBRE

SOBRE BAELDUNG (/ABOUT)
EL ARCHIVO COMPLETO (/FULL_ARCHIVE)
ESCRIBE PARA BAELDUNG (/CONTRIBUTION-GUIDELINES)
EDITORES (/EDITORS)
TRABAJOS (/TAG/ACTIVE-JOB/)
NUESTROS COMPAÑEROS (/PARTNERS)
ANÚNCIESE EN BAELDUNG (/ADVERTISE)

TÉRMINOS DE SERVICIO (/TERMS-OF-SERVICE)
POLÍTICA DE PRIVACIDAD (/PRIVACY-POLICY)
INFORMACIÓN DE LA COMPAÑÍA (/BAELDUNG-COMPANY-INFO)
CONTACTO (/CONTACT)