Spring Kafka - JSON Serializer Deserializer Example

6 minute read

JSON (http://www.json.org/) (JavaScript Object Notation) is a lightweight data-interchange format that uses human-readable text to transmit data objects. It is built on two structures: a collection of name/value pairs and an ordered list of values.

The following tutorial illustrates how to send/receive a Java object as a JSON `byte[]` array to/from Apache Kafka using Spring Kafka, Spring Boot and Maven.

If you want to learn more about Spring Kafka - head on over to the Spring Kafka tutorials page (https://codenotfound.com/spring-kafka/).

General Project Setup

Tools used:

- Spring Kafka 1.2
- Spring Boot 1.5
- Maven 3.5

Apache Kafka (https://kafka.apache.org/) stores and transports `Byte` arrays in its topics. It ships with a number of built in (de)serializers (https://kafka.apache.org/0100/javadoc/org/apache/kafka/common/serialization/Serializer.html) but a JSON one is not included. Luckily, the Spring K framework (https://projects.spring.io/spring-kafka/) includes a support package that contains a JSON (de)serializer (https://github.com/spring-projects/spring-kafka/tree/master/spring-kafka/src/main/java/org/springframework/kafka/support/serializer) that uses a Jackson (https://github.com/FasterXML/jackson) `ObjectMapper` under the covers.

We base the below example on a previous Spring Kafka example (https://codenotfound.com/spring-kafka-consumer-producer-example.html). Th only thing that needs to be added to the Maven POM file for working with JSON is the `spring-boot-starter-web` dependency which indirectly include the needed `jackson-*` JAR dependencies.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.codenotfound</groupId>
  <artifactId>spring-kafka-json</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <name>spring-kafka-json</name>
  <description>Spring Kafka - JSON Serializer Deserializer Example</description>
  <url>https://www.codenotfound.com/spring-kafka-json-serializer-deserializer-example.html</url>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.4.RELEASE</version>
  </parent>

  <properties>
    <java.version>1.8</java.version>

    <spring-kafka.version>1.2.2.RELEASE</spring-kafka.version>
  </properties>

  <dependencies>
    <!-- spring-boot -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
```

```
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
    <!-- spring-kafka -->
    <dependency>
      <groupId>org.springframework.kafka</groupId>
      <artifactId>spring-kafka</artifactId>
      <version>${spring-kafka.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework.kafka</groupId>
      <artifactId>spring-kafka-test</artifactId>
      <version>${spring-kafka.version}</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <!-- spring-boot-maven-plugin -->
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```


Object Model to Serialize/Deserialize

To illustrate the example we will send a `Car` object to a `'json.t'` topic. Let's use following class representing a car with a basic structure.

```
package com.codenotfound.model;

public class Car {

  private String make;
  private String manufacturer;
  private String id;

  public Car() {
    super();
  }

  public Car(String make, String manufacturer, String id) {
    super();
    this.make = make;
    this.manufacturer = manufacturer;
    this.id = id;
  }

  public String getMake() {
    return make;
  }

  public void setMake(String make) {
    this.make = make;
  }

  public String getManufacturer() {
    return manufacturer;
  }


  public void setManufacturer(String manufacturer) {
    this.manufacturer = manufacturer;
```

```
    }

    public String getId() {
      return id;
    }


    public void setId(String id) {
      this.id = id;
    }

    @Override
    public String toString() {
      return "Car [make=" + make + ", manufacturer=" + manufacturer + ", id=" + id + "]";
    }
}
```


Producing JSON Messages to a Kafka Topic

In order to use the `JsonSerializer`, shipped with Spring Kafka, we need to set the value of the producer's `'VALUE_SERIALIZER_CLASS_CONFIG'` configuration property to the `JsonSerializer` class. In addition, we change the `ProducerFactory` and `KafkaTemplate` generic type so that it specifies `Car` instead of `String`. This will result in the `Car` object to be serialized in a JSON `byte[]` message.

```
package com.codenotfound.kafka.producer;

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.codenotfound.model.Car;

@Configuration
public class SenderConfig {

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

  @Bean
  public Map<String, Object> producerConfigs() {
    Map<String, Object> props = new HashMap<>();
    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
    props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
    props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, JsonSerializer.class);

    return props;
  }

  @Bean
```

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

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

  @Bean
  public Sender sender() {
    return new Sender();
  }
}
```


The `Sender` class is updated accordingly so that it's `send()` method accepts a `Car` object as input. We also update the `KafkaTempla` generic type from `String` to `Car`.

```
package com.codenotfound.kafka.producer;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.kafka.core.KafkaTemplate;

import com.codenotfound.model.Car;

public class Sender {

  private static final Logger LOGGER = LoggerFactory.getLogger(Sender.class);

  @Value("${kafka.topic.json}")
  private String jsonTopic;

  @Autowired
  private KafkaTemplate<String, Car> kafkaTemplate;

  public void send(Car car) {
    LOGGER.info("sending car='{}'", car.toString());
    kafkaTemplate.send(jsonTopic, car);
  }
}
```


Consuming JSON Messages from a Kafka Topic

To receive the JSON serialized message we need to update the value of the `'VALUE_DESERIALIZER_CLASS_CONFIG'` property so that it points to the `JsonDeserializer` class. The `ConsumerFactory` and `ConcurrentKafkaListenerContainerFactory` generic type needs to be changed so that it specifies `Car` instead of `String`.

> *Note that the* `JsonDeserializer` *requires a* `Class<?>` *argument to allow the deserialization of a consumed* `byte[]` *to the proper target object (in this example the* `Car` *class).*

```
package com.codenotfound.kafka.consumer;

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.codenotfound.model.Car;

@Configuration
@EnableKafka
public class ReceiverConfig {

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

  @Bean
  public Map<String, Object> consumerConfigs() {
    Map<String, Object> props = new HashMap<>();
    props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
    props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
    props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, JsonDeserializer.class);
    props.put(ConsumerConfig.GROUP_ID_CONFIG, "json");

    return props;
```

```
  }

  @Bean
  public ConsumerFactory<String, Car> consumerFactory() {
    return new DefaultKafkaConsumerFactory<>(consumerConfigs(), new StringDeserializer(),
        new JsonDeserializer<>(Car.class));
  }

  @Bean
  public ConcurrentKafkaListenerContainerFactory<String, Car> kafkaListenerContainerFactory() {
    ConcurrentKafkaListenerContainerFactory<String, Car> factory =
        new ConcurrentKafkaListenerContainerFactory<>();
    factory.setConsumerFactory(consumerFactory());

    return factory;
  }

  @Bean
  public Receiver receiver() {
    return new Receiver();
  }
}
```


Identical to the updated `Sender` class, the argument of the `receive()` method of the `Receiver` class needs to be changed to the `Car` type.

```
package com.codenotfound.kafka.consumer;

import java.util.concurrent.CountDownLatch;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.kafka.annotation.KafkaListener;

import com.codenotfound.model.Car;

public class Receiver {

  private static final Logger LOGGER = LoggerFactory.getLogger(Receiver.class);

  private CountDownLatch latch = new CountDownLatch(1);

  public CountDownLatch getLatch() {
    return latch;
  }

  @KafkaListener(topics = "${kakfa.topic.json}")
  public void receive(Car car) {
    LOGGER.info("received car='{}'", car.toString());
    latch.countDown();
  }
}
```


Test Sending and Receiving JSON Messages on Kafka

The Maven project contains a `SpringKafkaApplicationTest` test case to demonstrate the above sample code. A JUnit ClassRule starts an embedded Kafka and ZooKeeper server (https://codenotfound.com/spring-kafka-embedded-unit-test-example.html).

Using `@Before` we wait until all the partitions are assigned to our `Receiver` by looping over the available `ConcurrentMessageListenerContainer` (if we don't do this the message will already be sent before the listeners are assigned to the topic).

In the `testReceiver()` test case we create a `Car` object and send it to the `'json.t'` topic. Finally the `CountDownLatch` from the `Receiver` is used to verify that a message was successfully received.

```
package com.codenotfound.kafka;

import static org.assertj.core.api.Assertions.assertThat;

import java.util.concurrent.TimeUnit;

import org.junit.Before;
import org.junit.ClassRule;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.kafka.config.KafkaListenerEndpointRegistry;
import org.springframework.kafka.listener.MessageListenerContainer;
import org.springframework.kafka.test.rule.KafkaEmbedded;
import org.springframework.kafka.test.utils.ContainerTestUtils;
import org.springframework.test.context.junit4.SpringRunner;

import com.codenotfound.kafka.consumer.Receiver;
import com.codenotfound.kafka.producer.Sender;
import com.codenotfound.model.Car;

@RunWith(SpringRunner.class)
@SpringBootTest
public class SpringKafkaApplicationTest {

  @Autowired
  private Sender sender;

  @Autowired
  private Receiver receiver;

  @Autowired
  private KafkaListenerEndpointRegistry kafkaListenerEndpointRegistry;
```

```
  @ClassRule
  public static KafkaEmbedded embeddedKafka = new KafkaEmbedded(1, true, "json.t");

  @Before
  public void setUp() throws Exception {
    // wait until the partitions are assigned
    for (MessageListenerContainer messageListenerContainer : kafkaListenerEndpointRegistry
        .getListenerContainers()) {
      ContainerTestUtils.waitForAssignment(messageListenerContainer,
          embeddedKafka.getPartitionsPerTopic());
    }
  }

  @Test
  public void testReceive() throws Exception {
    Car car = new Car("Passat", "Volkswagen", "ABC-123");
    sender.send(car);

    receiver.getLatch().await(10000, TimeUnit.MILLISECONDS);
    assertThat(receiver.getLatch().getCount()).isEqualTo(0);
  }
}
```


In order to run the above example open a command prompt and execute following Maven command:

```
mvn test
```

Maven will download the needed dependencies, compile the code and run the unit test case. The result should be a successful build during which following logs are generated:

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.5.4.RELEASE)

16:38:11.745 [main] INFO  c.c.kafka.SpringKafkaApplicationTest - Starting SpringKafkaApplicationTest on cnf-pc with PID 6116 (started by CodeNotFound in c:\codenotfound\code\spring-kafka\spring-kafka-json)
16:38:11.745 [main] INFO  c.c.kafka.SpringKafkaApplicationTest - No active profile set, falling back to default profiles: default
16:38:13.633 [main] INFO  c.c.kafka.SpringKafkaApplicationTest - Started SpringKafkaApplicationTest in 2.184 seconds (JVM running for 6.4
16:38:15.021 [main] INFO  c.codenotfound.kafka.producer.Sender - sending car='Car [make=Passat, manufacturer=Volkswagen, id=ABC-123]'
16:38:15.115 [org.springframework.kafka.KafkaListenerEndpointContainer#0-0-C-1] INFO  c.c.kafka.consumer.Receiver - received car='Car [make=Passat, manufacturer=Volkswagen, id=ABC-123]'
16:38:18.391 [main] ERROR o.a.zookeeper.server.ZooKeeperServer - ZKShutdownHandler is not registered, so ZooKeeper server won't take any action on ERROR or SHUTDOWN server state changes
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 11.622 sec - in com.codenotfound.kafka.SpringKafkaApplicationTest

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 14.714 s
[INFO] Finished at: 2017-08-02T16:38:19+02:00
[INFO] Final Memory: 29M/214M
[INFO] ------------------------------------------------------------------------
```

If you would like to run the above code sample you can get the full source code here (https://github.com/code-not-found/spring-kafka/tree/master/spring-kafka-json).

This concludes the example on how to use the Spring Kafka JsonSerializer/JsonDeserializer in combination with Apache Kafka.

If you have some questions or remarks, drop me a line below.

Tags: Apache Kafka | Deserializer | Example | JSON | Maven | Serializer | Spring Boot | Spring Kafka | Spring | Tutorial

Categories: Spring Kafka

Updated: March 21, 2017

LEAVE A COMMENT

Angga Raditya • 4 months ago

Hey i got "org.apache.kafka.common.errors.SerializationException: Can't convert value of class com.example.Foo to class org.apache.kafka.common.serialization.StringSerializer specified in value.serializer"

Reply • Share ›

Sthitaprajna(Pintu) Jena • a year ago

https://stackoverflow.com/q...

Can anyone help me on this ?

Reply • Share ›

codenotfound Mod ➚ Sthitaprajna(Pintu) Jena • a year ago

Hi,

Did you check the latest answer of Gary Russell?

Reply • Share ›

Elvis Orji • a year ago

Nice one.
My question is ... How do you handle exceptions when for instance a string is sent to the topic but the consumer is expecting a json.

Reply • Share ›

codenotfound Mod ➚ Elvis Orji • a year ago

Hi,

Good question. In this scenario I think you have more or less two options:

1) You could customize the *JsonDeserializer* to catch the exception and return something that indicates the failure to the *receive()* method (like a special type of *Car*).

2) Instead of using *JsonDeserializer*, use a *StringDeserializer* and then a separate JSON to Object transformer in the *receive()* method.

Reply • Share ›

Praneeth Reddy • 2 years ago

This is really good and thank you so much for the explanation. I just have one question, how to handle avro unions with help from spring kafka JsonDeserializer.

Reply • Share ›

codenotfound Mod ➚ Praneeth Reddy • 2 years ago

Hi,

Thanks for the nice comment.

The Avro union datatype should be automatically handled if you implement an Avro serializer/deserializer: https://www.codenotfound.co...

Or you could use a framework like Bijection: https://www.codenotfound.co... to handle the Avro message.

Reply • Share ›

Ravindar • 2 years ago

Nice Article clear and easy to understand.
I have one question....
Here you are creating kafkaListenerContainerFactory using the below code

@Bean
public ConcurrentKafkaListenerContainerFactory<string, car=""> kafkaListenerContainerFactory() {
ConcurrentKafkaListenerContainerFactory<string, car=""> factory =
new ConcurrentKafkaListenerContainerFactory<>();
factory.setConsumerFactory(consumerFactory());

return factory;
}

My question is if i have another class called Bike how to create the factory for both Car and Bike
i Tried to create the two ConcurrentKafkaListenerContainerFactory iin two different class by annotating with enableKafka
But always only one factory is creating.
how can we create multiple ConcurrentKafkaListenerContainerFactory for different object type with in the same application.

Thanks,
Reply • Share ›

codenotfound Mod Ravindar • 2 years ago

Hi,

Thanks for the nice comment.

Check following Github project I made a while back: https://github.com/code-not...

It shows how to receive different types (Foo and Bar) on different topics using a single *ConcurrentKafkaListenerContainerFactory*.

Or do you really need two different factories?

1 Reply • Share ›

Maxim Webster codenotfound • 3 months ago

What is required to receive different types on the same topic? Or is this considered to be bad practive?

Reply • Share ›

Show more replies