

[Home](#)

[Java](#)

[Apache](#)

[Compress](#)

[HttpClient](#)

[Collections](#)

[Date and Time](#)

[PDF](#)

[Apache PdfBox](#)

[iText](#)

[Image](#)

[JAX-RS \(REST\)](#)

[JAX-WS \(SOAP\)](#)

[JSON](#)

[Google GSON](#)

[Mail](#)

[Servlet](#)

[Utilities](#)

[XML](#)

[JavaScript](#)

[Spring Framework](#)

[Spring Boot](#)

[Spring Core](#)

[Spring Cloud](#)

[Spring Data](#)

Spring JMS
Spring Kafka
Spring LDAP
Spring Mail
Spring Mobile
Spring MVC
Spring Security
Spring WS
Database
Hibernate
MySQL
Redis
JSF
Logging
Log4j
Logback
Testing
JUnit
Mockito
Selenium
Build Tools
Maven
Photoshop





Spring Kafka – JSON Serializer and Deserializer Example

BY [MEMORYNOTFOUND](#) · PUBLISHED MARCH 6, 2018 · UPDATED MARCH 6, 2018

DISCOVER MORE ARTICLES

[Unit Test Spring MVC Rest Service: MockMVC, JUnit, Mockito](#)

0



0

[Spring Boot ActiveMQ Publish Subscribe Topic Configuration Example](#)

1

[Spring Boot – @ConfigurationProperties Annotation Example](#)

[Spring Autowire By Name](#)

SHARE



Best IT HelpDesk Software

100,000+ Successful IT Help Desks use ServiceDesk Plus to Supercharge their IT Help Desk

Spring Boot – Create Executable using Maven with Parent Pom

The following tutorial demonstrates how to send and receive a Java Object as a JSON `byte[]` to and from Apache Kafka using Spring Kafka, Spring Boot and Maven. We'll send a Java Object as JSON `byte[]` to a Kafka Topic using a `JsonSerializer`. Afterwards we'll configure how to receive a JSON `byte[]` and automatically convert it to a Java Object using a `JsonDeserializer`.



Project Setup

Spring Kafka: 2.1.4.RELEASE

Spring Boot: 2.0.0.RELEASE

Apache Kafka: kafka_2.11-1.0.0

Maven: 3.5

Maven Dependencies

We use Apache Maven to manage our project dependencies. Make sure the following dependencies reside on the class-path. Since we are working with JSON, we need to include the Jackson JSON library

com.fasterxml.jackson.core:ackson-databind.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.
  xmlns:xsi="http://www.w3.org/2001/XMLSchema
  xsi:schemaLocation="http://maven.apache
    http://maven.apache

  <modelVersion>4.0.0</modelVersion>
  <groupId>com.memorynotfound.spring.kafka</gr
  <artifactId>message-conversion-json</artifac
  <version>1.0.0-SNAPSHOT</version>
```

```
<url>http://memorynotfound.com/spring-kafka-  
<description>Spring Kafka - JSON Serializer  
<name>Spring Kafka - ${project.artifactId}</
```

```
<parent>  
    <groupId>org.springframework.boot</group  
    <artifactId>spring-boot-starter-parent</  
    <version>2.0.0.RELEASE</version>  
</parent>
```

```
<properties>  
    <java.version>1.8</java.version>  
    <project.build.sourceEncoding>UTF-8</pro  
    <spring-kafka.version>2.1.4.RELEASE</spr  
</properties>
```

```
<dependencies>  
    <dependency>  
        <groupId>org.springframework.boot</g  
        <artifactId>spring-boot-starter</art  
    </dependency>  
    <dependency>  
        <groupId>org.springframework.kafka</
```

```
        <artifactId>spring-kafka</artifactId>
        <version>${spring-kafka.version}</ve
</dependency>
```

```
<!-- json support -->
<dependency>
    <groupId>com.fasterxml.jackson.core<
    <artifactId>jackson-databind</artifa
</dependency>
```

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

```
</dependencies>
```

```
<build>
```

```
  <plugins>
```

```
    <plugin>
```

```
      <groupId>org.springframework.bo
```

```
      <artifactId>spring-boot-maven-pl
```

```
    </plugin>
```

```
  </plugins>
```

```
</build>
```

```
</project>
```

Simple POJO to Serialize/Deserialize

In this example we'll send and receive a Foo object to and from a Kafka topic.

```
package com.memorynotfound.kafka;
```

```
public class Foo {
```

```
    private String name;
```



```
private String description;

public Foo() {
}

public Foo(String name, String description)
    this.name = name;
    this.description = description;
}

public String getName() {
    return name;
}

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

public String getDescription() {
    return description;
}

public void setDescription(String descriptio
```

```
        this.description = description;
    }

    @Override
    public String toString() {
        return "Foo{" +
            "name='" + name + '\'' +
            ", description='" + description
            + '\'';
    }
}
```

Apache Kafka stores and transports byte[].
There are a number of built in [serializers](#) and [deserializers](#) but it doesn't include any for JSON. Spring Kafka created a [JsonSerializer](#) and [JsonDeserializer](#) which we can use to convert Java Objects to and from JSON.

Producing JSON messages with Spring Kafka

Let's start by sending a Foo object to a Kafka Topic. Notice: we created a

KafkaTemplate<String, Foo> since we are sending Java Objects to the Kafka topic that'll automatically be transformed in a JSON byte []. In this example we created a Message<Foo> using the MessageBuilder. It's important to add the topic where we are going to send the message to.

```
package com.memorynotfound.kafka.producer;

import com.memorynotfound.kafka.Foo;
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 org.springframework.kafka.support.KafkaHeaders;
import org.springframework.messaging.Message;
import org.springframework.messaging.support.MessageBuilder;
import org.springframework.stereotype.Service;

@Service
public class FooSender {
```

```

private static final Logger LOG = LoggerFactory

@Autowired
private KafkaTemplate<String, Foo> kafkaTemp

@Value("${app.topic.example}")
private String topic;

public void send(Foo data){
    LOG.info("sending data='{}' to topic='{}'

    Message<Foo> message = MessageBuilder
        .withPayload(data)
        .setHeader(KafkaHeaders.TOPIC, t
        .build();

    kafkaTemplate.send(message);
}
}

```

Starting with version 2.1, type information can be conveyed in record Headers, allowing the

handling of multiple types. In addition, the serializer/deserializer can be configured using Kafka Properties.

`JsonSerializer.ADD_TYPE_INFO_HEADERS` (default true); set to false to disable this feature.

`JsonSerializer.DEFAULT_KEY_TYPE`; fallback type for deserialization of keys if no header information is present.

`JsonSerializer.DEFAULT_VALUE_TYPE`; fallback type for deserialization of values if no header information is present.

`JsonSerializer.TRUSTED_PACKAGES` (default `java.util, java.lang`); comma-delimited list of packages patterns allowed for deserialization; * means deserialize all.

We need to configure the correct Serializer to support JSON types. We can register this by setting the

`ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG` to the `JsonSerializer` class. Finally, we need to set the correct value type for our

ProducerFactory and KafkaTemplate to the
Foo object.

```
package com.memorynotfound.kafka.producer;

import com.memorynotfound.kafka.Foo;
import org.apache.kafka.clients.producer.Producer
import org.apache.kafka.common.serialization.Str
import org.springframework.beans.factory.annotat
import org.springframework.context.annotation.Be
import org.springframework.context.annotation.Co
import org.springframework.kafka.core.DefaultKaf
import org.springframework.kafka.core.KafkaTempl
import org.springframework.kafka.core.ProducerFa
import org.springframework.kafka.support.seriali

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

@Configuration
public class FooSenderConfig {

    @Value("${spring.kafka.bootstrap-servers}")
```

```
private String bootstrapServers;
```

```
@Bean
```

```
public Map<String, Object> producerConfigs()  
    Map<String, Object> props = new HashMap<  
        props.put(ProducerConfig.BOOTSTRAP_SERVERS,  
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,  
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,  
    return props;  
}
```

```
@Bean
```

```
public ProducerFactory<String, Foo> producerFactory()  
    return new DefaultKafkaProducerFactory<>(props);  
}
```

```
@Bean
```

```
public KafkaTemplate<String, Foo> kafkaTemplate()  
    return new KafkaTemplate<>(producerFactory());  
}
```

```
}
```

Consuming JSON Messages with Spring Kafka

Next, we'll look at how we can receive JSON messages. In the `FooListener` we simply need to add the `Foo` Java Object as a parameter in our method.

```
package com.memorynotfound.kafka.consumer;

import com.memorynotfound.kafka.Foo;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.messaging.MessageHeader;
import org.springframework.messaging.handler.annotation.MessageHeader;
import org.springframework.messaging.handler.annotation.Payload;
import org.springframework.stereotype.Service;

@Service
public class FooListener {

    private static final Logger LOG = LoggerFactory.getLogger(FooListener.class);
```



```

    @KafkaListener(topics = "${app.topic.example}
    public void receive(@Payload Foo data,
                        @Headers MessageHeaders
                        LOG.info("received data='{ }'", data);

    headers.keySet().forEach(key -> {
        LOG.info("{ }: { }", key, headers.get(
    });
}
}

```

The FooListenerConfig is a bit more complex. First we need to add the appropriate Deserializer which can convert JSON byte [] into a Java Object. To do this, we need to set the

ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG with the JsonSerializer class. Next we need to create a ConsumerFactory and pass the consumer configuration, the key deserializer and the typed JsonSerializer<>(Foo.class). Finally,

we need to make sure the ConsumerFactory
and the
ConcurrentKafkaListenerContainerFactory
all have the correct value type of Foo.

```
package com.memorynotfound.kafka.consumer;

import com.memorynotfound.kafka.Foo;
import org.apache.kafka.clients.consumer.Consume
import org.apache.kafka.common.serialization.Str
import org.springframework.beans.factory.annotat
import org.springframework.context.annotation.Be
import org.springframework.context.annotation.Co
import org.springframework.kafka.annotation.Enab
import org.springframework.kafka.config.Concurre
import org.springframework.kafka.core.ConsumerFa
import org.springframework.kafka.core.DefaultKaf
import org.springframework.kafka.support.seriali

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

@Configuration
```

@EnableKafka

```
public class FooListenerConfig {
```

```
    @Value("${spring.kafka.bootstrap-servers}")
```

```
    private String bootstrapServers;
```

```
    @Bean
```

```
    public Map<String, Object> consumerConfigs()
```

```
        Map<String, Object> props = new HashMap<
```

```
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS,
```

```
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_NAME,
```

```
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_NAME,
```

```
        props.put(ConsumerConfig.GROUP_ID_CONFIG, "foo-group-id");
```

```
        props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
```

```
        return props;
```

```
    }
```

```
    @Bean
```

```
    public ConsumerFactory<String, Foo> consumerFactory()
```

```
        return new DefaultKafkaConsumerFactory<>
```

```
            consumerConfigs(),
```

```
            new StringDeserializer(),
```

```
            new JsonSerializer<>(Foo.class)
```

```

    }

    @Bean
    public ConcurrentKafkaListenerContainerFactory<
        ConcurrentKafkaListenerContainerFactory<
            new ConcurrentKafkaListenerConta
            factory.setConsumerFactory(consumerFacto
            return factory;
        }
    }
}

```

Configure with application.yml

We also create a application.yml properties file which is located in the src/main/resources folder. These properties are injected in the configuration classes by spring boot.

```

spring:
  kafka:
    bootstrap-servers: localhost:9092

```

```
app:
  topic:
    example: example.t

logging:
  level:
    root: WARN
    org.springframework.web: INFO
    com.memorynotfound: DEBUG
```

Running with Spring Boot

Finally, we wrote a simple Spring Boot application to demonstrate the application. In order for this demo to work, we need a Kafka Server running on localhost on port 9092, which is the default configuration of Kafka.

```
package com.memorynotfound.kafka;

import com.memorynotfound.kafka.producer.FooSend
import org.springframework.beans.factory.annotation
import org.springframework.boot.CommandLineRunner
```

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.Sp
```

```
@SpringBootApplication
```

```
public class SpringKafkaApplication implements C
```

```
    public static void main(String[] args) {
        SpringApplication.run(SpringKafkaApplica
    }
```

```
@Autowired
```

```
private FooSender sender;
```

```
@Override
```

```
public void run(String... strings) throws Ex
    Foo foo = new Foo("Spring Kafka", "sendi
    sender.send(foo);
}
```

```
}
```

Output

When we run the application we receive the following output.

```

      .      _____      -      _____      -
/\ \ /  ___ '  _ _ _ _ _ ( _ ) _ _ _ _ _ \ \ \ \ \
( ( ) \___ | ' _ | ' _ | | ' _ \ / _ ` | \ \ \ \ \
\ \ /  ___ ) | | _ ) | | | | | | | ( _ | | ) ) ) )
'   | ___ | . _ | | | _ | | \_, | / / / /
=====|_|=====|___/=/_/_/_/_/
:: Spring Boot ::          (v2.0.0.RELEASE)
```

```
Running with Spring Boot v2.0.0.RELEASE, Spring
No active profile set, falling back to default p
sending data='Foo{name='Spring Kafka', descripti
received data='Foo{name='Spring Kafka', descript
kafka_offset: 18
kafka_consumer: org.apache.kafka.clients.consume
kafka_timestampType: CREATE_TIME
kafka_receivedMessageKey: null
kafka_receivedPartitionId: 0
kafka_receivedTopic: example.t
```

```
kafka_receivedTimestamp: 1520332684097  
__TypeId__: [99, 111, 109, 46, 109, 101, 109, 11
```

References

[Apache Kafka Official Website](#)

[Spring Kafka Client Compatibility](#)

[Spring Kafka Documentation](#)

[Spring Kafka JavaDoc](#)

[Spring Kafka Serialize Deserialize
Documentation](#)

Download

Download it – [spring-kafka-jsonserializer-
jsondeserializer-example](#)



You're not connected

And the web just isn't the same without

Let's get you back online

Tags: [Apache Kafka](#)[Consumer](#)[JSON](#)[Kafka](#)[Producer](#)[Spring Kafka](#)

You may also like...

 [spring-boot-activemq-logo](#)

Setting and Reading Spring JMS Message Header Properties Example

October 11, 2017



Spring WS-Addressing Delegate Response

March 24, 2016



Spring Boot – Create Executable using Maven without Parent Pom

May 29, 2017

Leave a Reply

☐ avatar

b

i

link

b-quote

U

ul

ol

li

code

spoiler

Start the discussion...

Subscribe

Subscribe to our mailing list

Subscribe

[Privacy policy](#)

[Cookie Policy](#)

[Terms of use](#)



© Copyright Memorynotfound.com 2015-2018

