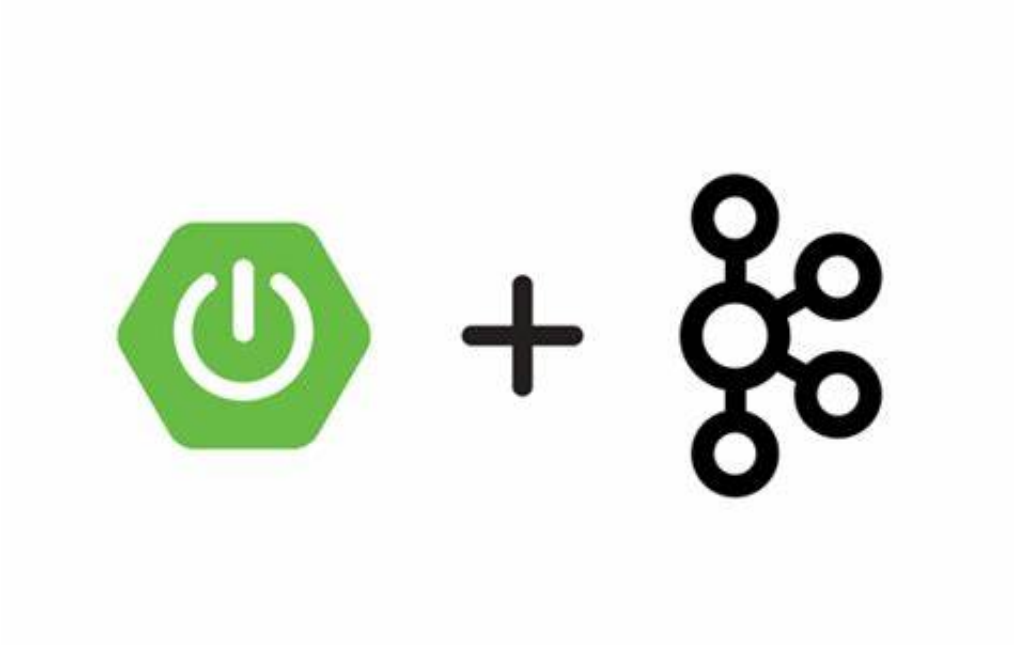
Spring Boot 中读写 Kafka header 信息

Apache Kafka 从 0.11.0.0 版本开始支持在消息中添加 header 信息，具体参见 KAFKA-4208。



如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：iteblog_hadoop

本文将介绍如何使用 spring-kafka 在 Kafka Message 中添加或者读取自定义 headers。本文使用各个系统的版本为：

- Spring Kafka: 2.1.4.RELEASE
- Spring Boot: 2.0.0.RELEASE
- Apache Kafka: kafka_2.11-1.0.0
- Maven: 3.5

我们使用 Maven 来管理项目，整个 pom.xml 文件如下：

```
<?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.iteblog.spring.kafka</groupId>
    <artifactId>message-headers</artifactId>
```

```
<version>1.0.0-SNAPSHOT</version>
<url>http://iteblog.com</url>
<name>Spring Kafka - ${project.artifactId}</name>

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

<properties>
   <java.version>1.8</java.version>
   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
   <spring-kafka.version>2.1.4.RELEASE</spring-kafka.version>
   <spring-boot.version>2.0.0.RELEASE</spring-boot.version>
</properties>

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

   <!-- for serializing/deserializing complex headers -->
   <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-databind</artifactId>
   </dependency>

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

```
    </dependencies>

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

</project>
```

使用 spring-kafka 在 Kafka Message 添加自定义 headers

我们使用 Message< ?> 或者 ProducerRecord< string , String> 往 Kafka Message 里面添加自定义 headers，具体如下：

```
package com.iteblog.kafka.producer;

import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.common.header.Header;
import org.apache.kafka.common.header.internals.RecordHeader;
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;

import java.util.ArrayList;
import java.util.List;

@Service
public class Sender {

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

    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;
```

```
    @Value("${app.topic.foo}")
    private String topicFoo;

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

    public void sendFoo(String data){

        Message<String> message = MessageBuilder
                .withPayload(data)
                .setHeader(KafkaHeaders.TOPIC, topicFoo)
                .setHeader(KafkaHeaders.MESSAGE_KEY, "999")
                .setHeader(KafkaHeaders.PARTITION_ID, 0)
                .setHeader("X-Custom-Header", "Sending Custom Header with Spring Kafka")
                .build();

        LOG.info("sending message='{}' to topic='{}'", data, topicFoo);
        kafkaTemplate.send(message);
    }

    public void sendBar(String data){

        List<Header> headers = new ArrayList<>();
        headers.add(new RecordHeader("X-Custom-
Header", "Sending Custom Header with Spring Kafka".getBytes()));

        ProducerRecord<String, String> bar = new ProducerRecord<>(topicBar, 0, "111", data, hea
ders);
        LOG.info("sending message='{}' to topic='{}'", data, topicBar);

        kafkaTemplate.send(bar);
    }
}
```

我们在 SenderConfig 类里面配置 KafkaTemplate，为了简单起见，我们使用 StringSerializer 来序列号 key 和 value：

```
package com.iteblog.kafka.producer;

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

@Configuration
public class SenderConfig {

    @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_CONFIG, bootstrapServers);
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
        return props;
    }

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

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

}
```

使用 spring-kafka 读取 Kafka Message 中自定义的 headers

前面我们已经学习到如何通过 spring-kafka 在 Kafka Message 中自定义的 headers，现在我们来看看如何获取这些 headers 信息：

```
package com.iteblog.kafka.consumer;
```

```
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.kafka.support.KafkaHeaders;
import org.springframework.messaging.MessageHeaders;
import org.springframework.messaging.handler.annotation.Header;
import org.springframework.messaging.handler.annotation.Headers;
import org.springframework.messaging.handler.annotation.Payload;
import org.springframework.stereotype.Service;

@Service
public class Listener {

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

    @KafkaListener(topics = "${app.topic.foo}")
    public void receive(@Payload String data,
                        @Header(KafkaHeaders.OFFSET) Long offset,
                        @Header(KafkaHeaders.CONSUMER) KafkaConsumer<String, String> consumer,
                        @Header(KafkaHeaders.TIMESTAMP_TYPE) String timestampType,
                        @Header(KafkaHeaders.RECEIVED_TOPIC) String topic,
                        @Header(KafkaHeaders.RECEIVED_PARTITION_ID) Integer partitionId,
                        @Header(KafkaHeaders.RECEIVED_MESSAGE_KEY) String messageKey,
                        @Header(KafkaHeaders.RECEIVED_TIMESTAMP) Long timestamp,
                        @Header("X-Custom-Header") String customHeader) {

        LOG.info("- - - - - - - - - - - - - - -");
        LOG.info("received message='{}'", data);
        LOG.info("consumer: {}", consumer);
        LOG.info("topic: {}", topic);
        LOG.info("message key: {}", messageKey);
        LOG.info("partition id: {}", partitionId);
        LOG.info("offset: {}", offset);
        LOG.info("timestamp type: {}", timestampType);
        LOG.info("timestamp: {}", timestamp);
        LOG.info("custom header: {}", customHeader);
    }

    @KafkaListener(topics = "${app.topic.bar}")
    public void receive(@Payload String data,
                        @Headers MessageHeaders messageHeaders) {

        LOG.info("- - - - - - - - - - - - - - -");
        LOG.info("received message='{}'", data);
        messageHeaders.keySet().forEach(key -> {
```

```
            Object value = messageHeaders.get(key);
            if (key.equals("X-Custom-Header")){
                LOG.info("{}: {}", key, new String((byte[])value));
            } else {
                LOG.info("{}: {}", key, value);
            }
        });

    }
}
```

DefaultKafkaHeaderMapper 将 key 映射到 MessageHeaders header name。

```
package com.iteblog.kafka.consumer;

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.config.KafkaListenerContainerFactory;
import org.springframework.kafka.core.ConsumerFactory;
import org.springframework.kafka.core.DefaultKafkaConsumerFactory;
import org.springframework.kafka.listener.ConcurrentMessageListenerContainer;
import org.springframework.kafka.support.DefaultKafkaHeaderMapper;

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

@EnableKafka
@Configuration
public class ListenerConfig {

    @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_CONFIG, bootstrapServers);
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
```

```
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class)
;
        props.put(ConsumerConfig.GROUP_ID_CONFIG, "foo");
        props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
        return props;
    }

    @Bean
    public ConsumerFactory<String, String> consumerFactory() {
        return new DefaultKafkaConsumerFactory<>(consumerConfigs());
    }

    @Bean
    public KafkaListenerContainerFactory<ConcurrentMessageListenerContainer<String, String>
> kafkaListenerContainerFactory() {
        ConcurrentKafkaListenerContainerFactory<String, String> factory = new ConcurrentKafkaL
istenerContainerFactory<>();
        factory.setConsumerFactory(consumerFactory());
        return factory;
    }

    @Bean
    public DefaultKafkaHeaderMapper headerMapper(){
        return new DefaultKafkaHeaderMapper();
    }

}
```

application.yml 文件配置

代码写完了，我们在 src/main/resources 目录下创建 application.yml 文件配置，内容如下：

```
spring:
  kafka:
    bootstrap-servers: localhost:9092

app:
  topic:
    foo: foo.t
    bar: bar.t

logging:
  level:
```

```
root: WARN
org.springframework.web: INFO
com.iteblog: DEBUG
```

使用 Spring Boot 运行 demo

最后，我们编写了一个简单的 Spring Boot 应用程序来运行我们上面的例子。为了让上面的 demo 运行成功，我们需要在本地的 9092 端口上运行 Kafka Server，其他 Kafka 配置使用默认的即可。

```
package com.iteblog.kafka;

import com.iteblog.kafka.producer.Sender;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringKafkaApplication implements CommandLineRunner {

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

    @Autowired
    private Sender sender;

    @Override
    public void run(String... strings) throws Exception {
        String data = "Spring Kafka Custom Header Example";
        sender.sendFoo(data);
        sender.sendBar(data);
    }
}
```

输出

```
  .   ____          _            __ _ _
 /₩₩ / ___'_ __ _ _(_)_ __  __ _ ₩ ₩ ₩ ₩
```

```
( ( )₩___ | '_ | '_| | '_ ₩/ _` | ₩ ₩ ₩ ₩
 ₩₩/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_₩__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.0.0.RELEASE)

Running with Spring Boot v2.0.0.RELEASE, Spring v5.0.4.RELEASE
No active profile set, falling back to default profiles: default
sending message='Spring Kafka Custom Header Example' to topic='foo.t'
sending message='Spring Kafka Custom Header Example' to topic='bar.t'
- - - - - - - - - - - - - - -
received message='Spring Kafka Custom Header Example'
X-Custom-Header: Sending Custom Header with Spring Kafka
kafka_offset: 49
kafka_consumer: org.apache.kafka.clients.consumer.KafkaConsumer@1c151fa8
kafka_timestampType: CREATE_TIME
kafka_receivedMessageKey: 111
kafka_receivedPartitionId: 0
kafka_receivedTopic: bar.t
kafka_receivedTimestamp: 1520322866725
- - - - - - - - - - - - - - -
received message='Spring Kafka Custom Header Example'
consumer: org.apache.kafka.clients.consumer.KafkaConsumer@7e4dcc64
topic: foo.t
message key: 999
partition id: 0
offset: 137
timestamp type: CREATE_TIME
timestamp: 1520322866701
custom header: Sending Custom Header with Spring Kafka
```