链滴

spring-boot Kafka

作者：crick77

原文链接：https://ld246.com/article/1498353443246

来源网站：链滴

spring-boot Kafka

基于springboot和kafka开发消息通信系统，提供消息发送、接收功能。

本文需要对kafka有基础的了解，且存在可访问的kafka服务。如果缺乏相关知识，请先阅读kafka官文档

环境工具

- jdk 1.8
- maven 3
- apache-kafka 2.11-0.10.1.0

项目依赖

- spring-boot 1.4.3.RELEASE
- spring-kafka 1.1.1.RELEASE

代码

发送消息

利用**spring-kafka**提供的**KafkaTemplate**模板实现发送消息的功能，因为**KafkaTemplate**不是sprigbean，所以在Config中手动创建实例Bean。

看一下**KafkaTemplate**构造函数相关源码

```
/**
 * Create an instance using the supplied producer factory and autoFlush false.
 * @param producerFactory the producer factory.
 */
public KafkaTemplate(ProducerFactory<K, V> producerFactory) {
    this(producerFactory, false);
}

/**
 * Create an instance using the supplied producer factory and autoFlush setting.
 * Set autoFlush to true if you wish to synchronously interact with Kafaka, calling
 * {@link Future#get()} on the result.
 * @param producerFactory the producer factory.
 * @param autoFlush true to flush after each send.
 */
public KafkaTemplate(ProducerFactory<K, V> producerFactory, boolean autoFlush) {
    this.producerFactory = producerFactory;
    this.autoFlush = autoFlush;
}
```

构造函数需要参数**ProducerFactory**参数，**ProducerFactory**接口一个有一个默认实现**DefaultKafkProducerFactory**。再看一下**DefaultKafkaProducerFactory**构造函数相关源码

```
public DefaultKafkaProducerFactory(Map<String, Object> configs) {
    this(configs, null, null);
}

public DefaultKafkaProducerFactory(Map<String, Object> configs, Serializer<K> keySerializer,
        Serializer<V> valueSerializer) {
    this.configs = new HashMap<>(configs);
    this.keySerializer = keySerializer;
    this.valueSerializer = valueSerializer;
}
```

需要的configs参数，在创建**KafkaProducer**时用于初始化参数，所以可以再创建一个configMap用配制**KafkaProducer**相关配置项

```
protected KafkaProducer<K, V> createKafkaProducer() {
    return new KafkaProducer<K, V>(this.configs, this.keySerializer, this.valueSerializer);
}
```

相关参数项参考**org.apache.kafka.clients.producer.ProducerConfig**，这里只配置最必需参数，括kafka地址、keyValue序列化。

发送端配置类如下：

```
package pro.hemo.study.kafka;

import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.common.serialization.IntegerSerializer;
import org.apache.kafka.common.serialization.StringSerializer;
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 KafkaProducerConfig {

    private Map<String, Object> configs() {
        Map<String, Object> configMap = new HashMap<>();
        configMap.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
                "localhost:9092");
        configMap.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, IntegerSerializer.class);
        configMap.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class)

        return configMap;
    }

    private ProducerFactory producerFactory() {
        return new DefaultKafkaProducerFactory(configs());
    }

    @Bean
```

```
    public KafkaTemplate kafkaTemplate() {
        return new KafkaTemplate(producerFactory());
    }
}
```

发送消息工具类方法

```
package pro.hemo.study.kafka.producer;

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

@Component
public class KafkaSendService {

    @Autowired
    private KafkaTemplate kafkaTemplate;

    public void sendMessage(String topic, String message) {
        kafkaTemplate.send(topic, message);
    }
}
```

可以通过测试用例测试消息发送，通过命令行模式监听对应的topic，查看是否能够接收消息。

```
package pro.hemo.study.kafka;

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.test.context.junit4.SpringRunner;
import pro.hemo.study.kafka.producer.KafkaSendService;

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


    @Autowired
    private KafkaSendService kafkaSendService;

    @Test
    public void testSendMessage() throws Exception {
        kafkaSendService.sendMessage("foo", "Hello SpringBoot Kafka!");
    }
}
```

接收消息

接收端的配置项代码和发送端类似，不同的事，接收端需要指定一个**KafkaListenerContainerFactory**，用于注册监听。先看代码

```
package pro.hemo.study.kafka;

import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.common.serialization.IntegerDeserializer;
import org.apache.kafka.common.serialization.StringDeserializer;
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 java.util.HashMap;
import java.util.Map;

@Configuration
@EnableKafka
public class KafkaConsumerConfig {

    private Map<String, Object> configs() {
        Map<String, Object> configMap = new HashMap<>();
        configMap.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
                "localhost:9092");
        configMap.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
                IntegerDeserializer.class);
        configMap.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
                StringDeserializer.class);
        configMap.put(ConsumerConfig.GROUP_ID_CONFIG, "groupTest");

        return configMap;
    }

    private ConsumerFactory consumerFactory() {
        return new DefaultKafkaConsumerFactory<>(configs());
    }

    @Bean
    public KafkaListenerContainerFactory kafkaListenerContainerFactory() {
        ConcurrentKafkaListenerContainerFactory factory = new ConcurrentKafkaListenerContai
erFactory<>();
        factory.setConsumerFactory(consumerFactory());
        return factory;
    }
}
```

结合接收消息工具类一起看

```
package pro.hemo.study.kafka.consumer;

import org.springframework.kafka.annotation.KafkaHandler;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Component;
```

```
@KafkaListener(topics = "topicTest")
@Component
public class KafkaReceiveService {

    @KafkaHandler
    public void receiveMessage(String message) {
        System.out.println("receive:" + message);
    }
}
```

消息接收端代码核心在于注解**@KafkaListener**，查看源码，重点为两个属性**containerFactory**和**opics**。

```
package org.springframework.kafka.annotation;

import java.lang.annotation.Documented;
import java.lang.annotation.ElementType;
import java.lang.annotation.Repeatable;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
import org.springframework.kafka.listener.MessageListener;
import org.springframework.messaging.handler.annotation.MessageMapping;

@Target({ ElementType.TYPE, ElementType.METHOD, ElementType.ANNOTATION_TYPE })
@Retention(RetentionPolicy.RUNTIME)
@MessageMapping
@Documented
@Repeatable(KafkaListeners.class)
public @interface KafkaListener {

    /**
     * The unique identifier of the container managing for this endpoint.
     * <p>If none is specified an auto-generated one is provided.
     * @return the {@code id} for the container managing for this endpoint.
     * @see org.springframework.kafka.config.KafkaListenerEndpointRegistry#getListenerConta
ner(String)
     */
    String id() default "";

    /**
     * The bean name of the {@link org.springframework.kafka.config.KafkaListenerContainerFa
tory}
     * to use to create the message listener container responsible to serve this endpoint.
     * <p>If not specified, the default container factory is used, if any.
     * @return the container factory bean name.
     */
    String containerFactory() default "";

    /**
     * The topics for this listener.
     * The entries can be 'topic name', 'property-placeholder keys' or 'expressions'.
     * Expression must be resolved to the topic name.
```

```
     * Mutually exclusive with {@link #topicPattern()} and {@link #topicPartitions()}.
     * @return the topic names or expressions (SpEL) to listen to.
     */
    String[] topics() default {};

    /**
     * The topic pattern for this listener.
     * The entries can be 'topic name', 'property-placeholder keys' or 'expressions'.
     * Expression must be resolved to the topic pattern.
     * Mutually exclusive with {@link #topics()} and {@link #topicPartitions()}.
     * @return the topic pattern or expression (SpEL).
     */
    String topicPattern() default "";

    /**
     * The topicPartitions for this listener.
     * Mutually exclusive with {@link #topicPattern()} and {@link #topics()}.
     * @return the topic names or expressions (SpEL) to listen to.
     */
    TopicPartition[] topicPartitions() default {};

    /**
     * If provided, the listener container for this listener will be added to a bean
     * with this value as its name, of type {@code Collection<MessageListenerContainer>}.
     * This allows, for example, iteration over the collection to start/stop a subset
     * of containers.
     * @return the bean name for the group.
     */
    String group() default "";

}
```

topics、topicPattern、topicPartitions都是用于指定监听的topic，而**containerFactory**需要指明听仓库**KafkaListenerContainerFactory**，所以在**KafkaConsumerConfig**配置类中创建对应的Bea对象，并根据构造函数添加相应的参数配置。

@EnableKafka

用于扫描对应的listener。

@KafkaHandler

指明消息接收处理方法。