02_kafka消费端配置+批量消费

时间：2023年1月11日17:02:32

推荐阅读：

- Kafka Consumer Configurations for Confluent Platform | Confluent Documentation
- Apache Kafka and Java – Getting Started Tutorial

一、Java消费者配置类

- 具体配置参数请参考上方【推荐阅读】

```
package org.apache.kafka.clients.consumer;

import ...


public class ConsumerConfig extends AbstractConfig {

    private static final ConfigDef CONFIG;

    public static final List<String> ASSIGN_FROM_SUBSCRIBED_ASSIGNORS = Collections.unmodifiableList(Arrays.asList("ran

    public static final String GROUP_ID_CONFIG = "group.id";

    private static final String GROUP_ID_DOC = "A unique string that identifies the consumer group this consumer belong

    public static final String GROUP_INSTANCE_ID_CONFIG = "group.instance.id";

    private static final String GROUP_INSTANCE_ID_DOC = "A unique identifier of the consumer instance provided by the e
```

二、SpringBoot消费者默认配置

```
2023-01-11 17:08:07.786 [main] INFO  o.a.k.c.consumer.ConsumerConfig - ConsumerConfig values:
        allow.auto.create.topics = true
        auto.commit.interval.ms = 5000
        auto.offset.reset = latest
        bootstrap.servers = [kafka-server-1:9092, kafka-server-2:9092, kafka-server-3:9092]
        check.crcs = true
        client.dns.lookup = use_all_dns_ips
        client.id = consumer-b-2
        client.rack =
        connections.max.idle.ms = 540000
        default.api.timeout.ms = 60000
        enable.auto.commit = false
        exclude.internal.topics = true
        fetch.max.bytes = 52428800
        fetch.max.wait.ms = 500
        fetch.min.bytes = 1
        group.id = b
        group.instance.id = null
        heartbeat.interval.ms = 3000
```

```
        interceptor.classes = []
        internal.leave.group.on.close = true
        internal.throw.on.fetch.stable.offset.unsupported = false
        isolation.level = read_uncommitted
        key.deserializer = class org.apache.kafka.common.serialization.StringDeserializer
        max.partition.fetch.bytes = 1048576
        max.poll.interval.ms = 300000
        max.poll.records = 500
        metadata.max.age.ms = 300000
        metric.reporters = []
        metrics.num.samples = 2
        metrics.recording.level = INFO
        metrics.sample.window.ms = 30000
        partition.assignment.strategy = [class org.apache.kafka.clients.consumer.RangeAssignor, class org.apache.kafka.
clients.consumer.CooperativeStickyAssignor]
        receive.buffer.bytes = 65536
        reconnect.backoff.max.ms = 1000
        reconnect.backoff.ms = 50
        request.timeout.ms = 30000
        retry.backoff.ms = 100
        sasl.client.callback.handler.class = null
        sasl.jaas.config = null
        sasl.kerberos.kinit.cmd = /usr/bin/kinit
        sasl.kerberos.min.time.before.relogin = 60000
        sasl.kerberos.service.name = null
        sasl.kerberos.ticket.renew.jitter = 0.05
        sasl.kerberos.ticket.renew.window.factor = 0.8
        sasl.login.callback.handler.class = null
        sasl.login.class = null
        sasl.login.connect.timeout.ms = null
        sasl.login.read.timeout.ms = null
        sasl.login.refresh.buffer.seconds = 300
        sasl.login.refresh.min.period.seconds = 60
        sasl.login.refresh.window.factor = 0.8
        sasl.login.refresh.window.jitter = 0.05
        sasl.login.retry.backoff.max.ms = 10000
        sasl.login.retry.backoff.ms = 100
        sasl.mechanism = GSSAPI
        sasl.oauthbearer.clock.skew.seconds = 30
        sasl.oauthbearer.expected.audience = null
        sasl.oauthbearer.expected.issuer = null
        sasl.oauthbearer.jwks.endpoint.refresh.ms = 3600000
        sasl.oauthbearer.jwks.endpoint.retry.backoff.max.ms = 10000
        sasl.oauthbearer.jwks.endpoint.retry.backoff.ms = 100
        sasl.oauthbearer.jwks.endpoint.url = null
        sasl.oauthbearer.scope.claim.name = scope
        sasl.oauthbearer.sub.claim.name = sub
        sasl.oauthbearer.token.endpoint.url = null
        security.protocol = PLAINTEXT
        security.providers = null
        send.buffer.bytes = 131072
        session.timeout.ms = 45000
        socket.connection.setup.timeout.max.ms = 30000
        socket.connection.setup.timeout.ms = 10000
        ssl.cipher.suites = null
        ssl.enabled.protocols = [TLSv1.2, TLSv1.3]
        ssl.endpoint.identification.algorithm = https
        ssl.engine.factory.class = null
```

```
        ssl.key.password = null
        ssl.keymanager.algorithm = SunX509
        ssl.keystore.certificate.chain = null
        ssl.keystore.key = null
        ssl.keystore.location = null
        ssl.keystore.password = null
        ssl.keystore.type = JKS
        ssl.protocol = TLSv1.3
        ssl.provider = null
        ssl.secure.random.implementation = null
        ssl.trustmanager.algorithm = PKIX
        ssl.truststore.certificates = null
        ssl.truststore.location = null
        ssl.truststore.password = null
        ssl.truststore.type = JKS
        value.deserializer = class org.apache.kafka.common.serialization.StringDeserializer

2023-01-11 17:08:07.791 [main] INFO  o.a.kafka.common.utils.AppInfoParser - Kafka version: 3.2.3
2023-01-11 17:08:07.791 [main] INFO  o.a.kafka.common.utils.AppInfoParser - Kafka commitId: 50029d3ed8ba576f
2023-01-11 17:08:07.791 [main] INFO  o.a.kafka.common.utils.AppInfoParser - Kafka startTimeMs: 1673428087789
```

三、批量消费

- spring boot整合Kafka批量消费、并发消费


```
spring:
  kafka:
    bootstrap-servers: kafka-server-1:9092,kafka-server-2:9092,kafka-server-3:9092
    consumer:
      # 消费端限流（当服务端短时间生产105条数据，此时消费端每批会接受10条处理，当生产端产生消息减缓时，消费端仍然会消费，不是非要等到10条数据才处理）
      max-poll-records: 10
    # 生产者开启批量生产，此时，默认消息会被封装为逗号分隔的字符串
    # hello#kafka-21,hello#kafka-22,hello#kafka-23,hello#kafka-24,hello#kafka-25,hello#kafka-26,hello#kafka-27
    listener:
      type: batch
```

1. 生产者

```
import lombok.RequiredArgsConstructor;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Component;

@Component
@RequiredArgsConstructor
```

```
public class KafkaProducer {

    private final KafkaTemplate<String, String> kafkaTemplate;

    public void produce(String msg) {
        // auto-create topic
        kafkaTemplate.send("topic-foo", msg);
    }
}
```

2. 消费者


```
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Component;

import java.util.List;
import java.util.Optional;

@Component
public class KafkaConsumer {

//    @KafkaListener(topics = {"topic-foo"}, groupId = "group-a")
//    public void ProductInsertEvent2(List<String> records) {
//        System.out.println("批量消费数据开始...");
//        records.forEach(System.out::println);
//        System.out.println("批量消费数据结束...");
//    }


    @KafkaListener(topics = {"topic-foo"}, groupId = "group-b")
    public void ProductInsertEvent1(String record) {
        System.out.printf("KafkaConsumer1 kafka value:%s%n", record);
    }
}
```

四、只消费指定格式的消息（消费端定义value解析器）

- 消费端只处理JSON格式消息，并将其封装为JSONObject（fastjson2）


```
kafka:
  bootstrap-servers: kafka-server-1:9092,kafka-server-2:9092,kafka-server-3:9092
  consumer:
    # 消费端限流（当服务端短时间生产105条数据，此时消费端每批会接受10条处理，当生产端产生消息减缓时，消费端仍然会消费，不必非要等到10条数据才处理）
    max-poll-records: 10
    key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
    value-deserializer: com.xii.mp.kafka.mq.converter.JSONObjectDeserializer
    # 从何处开始消费: latest 表示消费最新消息,earliest 表示从头开始消费(默认latest)
    auto-offset-reset: latest
  # 生产者开启批量生产, 此时，默认消息会被封装为逗号分隔的字符串, 可在消费端指定value解析器 → value-deserializer
  # hello#kafka-21,hello#kafka-22,hello#kafka-23,hello#kafka-24,hello#kafka-25,hello#kafka-26,hello#kafka-27
  listener:
    type: batch
```

1. value解析器

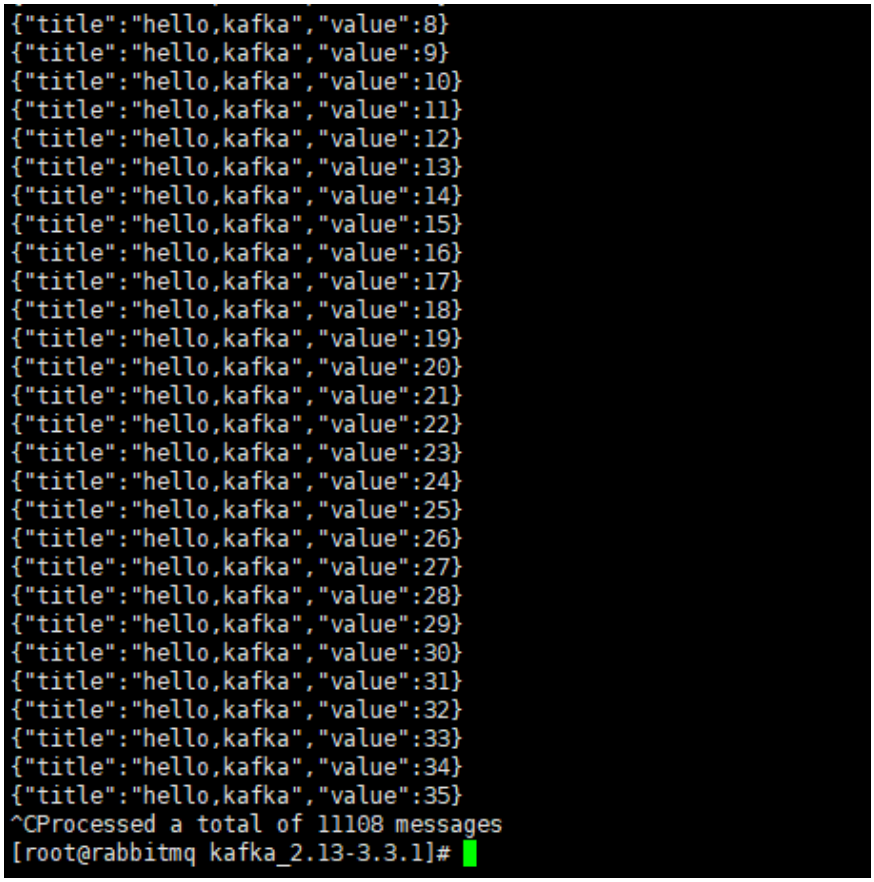
- 默认的是String解析器

```
package com.xii.mp.kafka.converter;

import com.alibaba.fastjson2.JSONObject;
import org.apache.kafka.common.errors.SerializationException;
import org.apache.kafka.common.serialization.Deserializer;
import org.springframework.stereotype.Component;

import java.io.UnsupportedEncodingException;
import java.nio.charset.StandardCharsets;
import java.util.Map;

@Component
public class JSONObjectDeserializer implements Deserializer<JSONObject> {
    private String encoding = StandardCharsets.UTF_8.name();

    @Override
    public void configure(Map<String, ?> configs, boolean isKey) {
        String propertyName = isKey ? "key.deserializer.encoding" : "value.deserializer.encoding";
        Object encodingValue = configs.get(propertyName);
        if (encodingValue == null)
            encodingValue = configs.get("deserializer.encoding");
        if (encodingValue instanceof String)
            encoding = (String) encodingValue;
    }

    @Override
```

```
    public JSONObject deserialize(String topic, byte[] data) {
        try {
            if (data != null) {
                String r = new String(data, encoding);
                if (r.startsWith("{"))
                    return JSONObject.parseObject(r);
            }
            return null;
        } catch (UnsupportedEncodingException e) {
            throw new SerializationException("Error when deserializing byte[] to string → JSONObject(fastjson2) due to unsupported encoding " + encoding);
        }
    }
}
```

2. 消费者

```
import com.alibaba.fastjson2.JSONObject;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Component;

import java.util.List;
import java.util.Objects;
import java.util.stream.Collectors;

@Component
public class KafkaConsumer {

//    // 生产者没有指定 listener.type.batch=true
//    //   - 可以使用ConsumerRecord<String, String>
//    //   - 可以使用String
//    @KafkaListener(topics = "topic-foo", groupId = "group-a")
//    public void ProductInsertEvent1(ConsumerRecord<String, String> record) {
//        Optional<String> kafkaMessage = Optional.ofNullable(record.value());
//        kafkaMessage.ifPresent(msg -> {
//            System.out.printf("KafkaConsumer1 kafka value:%s%n", msg);
//        });
//    }


    // listener.type.batch=true
    // consumer.max-poll-records=10
    // consumer.value-deserializer=com.xii.mp.kafka.converter.JSONObjectDeserializer
    @KafkaListener(topics = {"topic-foo"}, groupId = "xxxx")
    public void ProductInsertEvent2(List<JSONObject> records) {
        records = records.stream().filter(Objects::nonNull).collect(Collectors.toList());
        if(records.size()>0) {
            System.out.println("批量消费数据开始...");
            records.forEach(System.out::println);
            System.out.println("批量消费数据结束...");
        }
    }

}
```



3. 生产者

```
import lombok.RequiredArgsConstructor;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Component;

@Component
@RequiredArgsConstructor
public class KafkaProducer {

    private final KafkaTemplate<String, String> kafkaTemplate;

    public void produce(String msg) {
        // auto-create topic
        kafkaTemplate.send(”topic-foo”, msg);
    }
}
```

```
import com.alibaba.fastjson2.JSONObject;
import com.xii.mp.kafka.KafkaProducer;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import java.util.concurrent.TimeUnit;

@SpringBootTest
public class AppTest {
```

```
    @Autowired
    private KafkaProducer kafkaProducer;

    @Test
    public void test01() {
        for (int i = 0; i < 35; i++) {
            JSONObject jsonObject = new JSONObject();
            jsonObject.put("title", "hello,kafka");
            jsonObject.put("value", i+1);
            kafkaProducer.produce(jsonObject.toJSONString());
            System.out.println("第" + (i + 1) + "条消息发送成功！");
        }

        try {
            TimeUnit.SECONDS.sleep(1000);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
}
```

五、从头开始消费

```
### kafka服务地址
kafka:
  bootstrap-servers: kafka-server-1:9092,kafka-server-2:9092,kafka-server-3:9092
  consumer:
    # 消费端限流（当服务端短时间生产105条数据，此时消费端每批会接受10条处理，当生产端产生消息减缓时，消费端仍然会消费，不是非要等到10条数据才处理）
    max-poll-records: 10
    value-deserializer: com.xii.mp.kafka.converter.JSONObjectDeserializer
    # 从何处开始消费：latest 表示消费最新消息,earliest 表示从头开始消费(默认latest)
    auto-offset-reset: earliest
  # 生产者开启批量生产，此时，默认消息会被封装为逗号分隔的字符串，可在消费端指定value解析器 → value-deserializer
  # hello#kafka-21,hello#kafka-22,hello#kafka-23,hello#kafka-24,hello#kafka-25,hello#kafka-26,hello#kafka-27
  listener:
    type: batch
```

```
// listener.type.batch=true
// consumer.max-poll-records=10
// consumer.value-deserializer=com.xii.mp.kafka.converter.JSONObjectDeserializer
0 个用法
@KafkaListener(topics = {"topic-foo"}, groupId = "group-a")    使用一个没有使用过的groupId
public void ProductInsertEvent2(List<JSONObject> records) {
    records = records.stream().filter(Objects::nonNull).collect(Collectors.toList());
    if(records.size()>0) {
        System.out.println("批量消费数据开始...");
        records.forEach(System.out::println);
        System.out.println("批量消费数据结束...");
    }
}
```