

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
sas.client.callback.handler.class = null
sas.jaas.config = null
sas.kerberos.kinit.cmd = /usr/bin/kinit
sas.kerberos.min.time.before.relogin = 60000
sas.kerberos.service.name = null
sas.kerberos.ticket.renew.jitter = 0.05
sas.kerberos.ticket.renew.window.factor = 0.8
sas.login.callback.handler.class = null
sas.login.class = null
sas.login.connect.timeout.ms = null
sas.login.read.timeout.ms = null
sas.login.refresh.buffer.seconds = 300
sas.login.refresh.min.period.seconds = 60
sas.login.refresh.window.factor = 0.8
sas.login.refresh.window.jitter = 0.05
sas.login.retry.backoff.max.ms = 10000
sas.login.retry.backoff.ms = 100
sas.mechanism = GSSAPI
sas.oauthbearer.clock.skew.seconds = 30
sas.oauthbearer.expected.audience = null
sas.oauthbearer.expected.issuer = null
sas.oauthbearer.jwks.endpoint.refresh.ms = 3600000
sas.oauthbearer.jwks.endpoint.retry.backoff.max.ms = 10000
sas.oauthbearer.jwks.endpoint.retry.backoff.ms = 100
sas.oauthbearer.jwks.endpoint.url = null
sas.oauthbearer.scope.claim.name = scope
sas.oauthbearer.sub.claim.name = sub
sas.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
```

```
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...");
    // }

    // 自动按照逗号(默认)拆分存放在List
    // → 问题: 如果消息中有逗号呢? (默认消息会被按照逗号异常拆分)

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

    // 逗号分隔的批量数据字符串 ↓
    hello#kafka-21,hello#kafka-22,hello#kafka-23,hello#kafka-24,hello#kafka-25,hello#kafka-26,hello#kafka-27
}

```

```

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慢时，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
    # 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;

@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 = "group-b")
    public void ProductInsertEvent2(List<JSONObject> 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...");
    }
}
```

```

{"title":"hello,kafka","value":8}
{"title":"hello,kafka","value":9}
{"title":"hello,kafka","value":10}
{"title":"hello,kafka","value":11}
{"title":"hello,kafka","value":12}
{"title":"hello,kafka","value":13}
{"title":"hello,kafka","value":14}
{"title":"hello,kafka","value":15}
{"title":"hello,kafka","value":16}
{"title":"hello,kafka","value":17}
{"title":"hello,kafka","value":18}
{"title":"hello,kafka","value":19}
{"title":"hello,kafka","value":20}
{"title":"hello,kafka","value":21}
{"title":"hello,kafka","value":22}
{"title":"hello,kafka","value":23}
{"title":"hello,kafka","value":24}
{"title":"hello,kafka","value":25}
{"title":"hello,kafka","value":26}
{"title":"hello,kafka","value":27}
{"title":"hello,kafka","value":28}
{"title":"hello,kafka","value":29}
{"title":"hello,kafka","value":30}
{"title":"hello,kafka","value":31}
{"title":"hello,kafka","value":32}
{"title":"hello,kafka","value":33}
{"title":"hello,kafka","value":34}
{"title":"hello,kafka","value":35}
^CProcessed a total of 11108 messages
[root@rabbitmq kafka_2.13-3.3.1]#

```

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);

}

}

}