Hello Guava
码龄3年 暂无认证
46 原创 | 22万+ 周排名 | 16万+ 总排名 | 1万+ 访问 | 等级
643 积分 | 28 粉丝 | 31 获赞 | 8 评论 | 40 收藏
私信 关注

搜博主文章

热门文章
Flink进阶（二）：使用 ParameterTool 读取配置 2156
Flink进阶（三）：双流connect的用法 1955
Kafka整合Springboot的用法：配置文件和自定义的方式、以及遇到的问题 1566
Flink基础（九）：Checkpoint的说明和用法 1514
Flink基础（八）：流作业中的广播变量和BroadcastState 1007

分类专栏
Flink学习之路 18篇
Redis 1篇
ClickHouse踩坑之路 1篇
Mysql 4篇
spring 1篇
SpringCloud 6篇

最新评论
Flink基础（八）：流作业中的广播变量和…
阿新在路上: 这种方式有可能会导致业务流来的时候，广播规则实际并没有获取到，…
Flink基础（十）：Flink常用的Source和S…
cx-young: 很赞
Flink进阶（三）：双流connect的用法
木冬木秋: 有全部的代码吗?
Spring Cloud Alibaba (二)： Gateway
water___Wang: 学习了
Kafka入门教程
ctotalk: 学习

最新文章
Flink进阶（五）：天维度的统计中涉及的自定义触发器
ClickHouse SQL记录
Spring Cloud Alibaba (五)： Sentinel
2021年 2篇 2020年 44篇

Kafka整合Springboot的用法：配置文件和自定义的方式、以及遇到的问题

原创 Hello Guava 2020-04-28 14:01:36 1593 收藏 1 版权
分类专栏： Kafka 文章标签： kafka

版本问题

SpringBoot整合Kafka要主要版本依赖问题，包括kafka-server、kafka-client、spring-kafka、SpringBoot的版本

首先声明下版本依赖关系，来自于官网，截图如下

Spring for Apache Kafka Version	Spring Integration for Apache Kafka Version	kafka-clients	Spring Boot
2.5.x	3.3.x	2.5.0	2.3.x
2.4.x	3.2.x	2.4.1	2.2.x
2.3.x	3.2.x	2.3.1	2.2.x
2.2.x	3.1.x	2.0.1, 2.1.x, 2.2.x	2.1.x
1.3.x	2.3.x	0.11.0.x, 1.0.x	1.5.x (EOL)

官网链接如下

直接用yml配置文件 + 注解

利用SpringBoot的自动装配的功能，直接在yml配置文件里配置就行了。
我安装的kafka是0.11.0.2 按照官网的版本依赖关系，pom文件依赖如下：

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/200
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.16.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.msy</groupId>
    <artifactId>demo-kafka</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>demo-kafka</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-configuration-processor</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.kafka</groupId>
            <artifactId>spring-kafka</artifactId>
            <version>1.3.5.RELEASE</version>
            <!-- 此处要主要点进去spring-kafka看下里面的kafka-client的版本是多少，是否与kafk
            <!--<exclusions>
                <exclusion>
                    <groupId>org.apache.kafka</groupId>
                    <artifactId>kafka-clients</artifactId>
                </exclusion>
            </exclusions>-->
        </dependency>

        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
            <exclusions>
                <exclusion>
                    <groupId>org.junit.vintage</groupId>
                    <artifactId>junit-vintage-engine</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
    </dependencies>

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

</project>
```

yml配置文件如下：参数说明直接看注释，有些需要配置，有些不需要配置直接用默认值即可。
至于有哪些配置以及这些配置的默认值都是什么，可以看源码；
比如生产者的相关配置可以看ProducerConfig类；消费者的相关配置可以看ConsumerConfig类；
还有Listener相关的配置

```yaml
spring:
  kafka:
    bootstrap-servers: 192.168.10.45:9092
    producer:
      # 重试次数，默认Integer.MAX_VALUE
      retries: 1
      # 同一批次内存大小（默认16K）
      batch-size: 16384
      # 生产者内存缓存区大小(32M)
      buffer-memory: 33554432
      # key和value的序列化（默认，可以不设置）
      key-serializer: org.apache.kafka.common.serialization.StringSerializer
      value-serializer: org.apache.kafka.common.serialization.StringSerializer
      # ack应答机制，默认1，即只需要确认leader收到消息
      acks: 1
      # springboot1.5.16自动装配中不支持properties下的其他配置，不知道为啥。2.x版本可以
      #properties:
        # 使用自定义的分区选择器
        #{partitioner.class: com.msy.kafka.MyPartition, acks: all}
    consumer:
      group-id: test
      enable-auto-commit: false
      # earliest：从头开始消费    latest：从最新的开始消费    默认latest
      auto-offset-reset: latest
```

```
      key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
      value-deserializer: org.apache.kafka.common.serialization.StringDeserializer
    listener:
      # 消费者并发能力
      concurrency: 6
      # 设置手动提交的时候，需要设置ackMode
      ack-mode: MANUAL
    topic: test5
```

如何发送消息，代码很简单

```
public class Producer {

    @Autowired
    private KafkaTemplate<String, String> kafkaTemplate;

    @Value("${spring.kafka.topic}")
    private String topic;

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

如何监听消息，代码也很简单，这里用了手动提交的方式，没有用自动提交，至于为什么，后面再说

```
@Component
@Slf4j
public class Consumer {

    @KafkaListener(topics = "test5")  // 支持监听多个topic的消息
    public void consumerMessage(ConsumerRecord<String, String> consumerRecord, Ackn
        try {
            String value = consumerRecord.value();
            log.info("监听到的消息为：{}", value);
            // 业务处理......
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            ack.acknowledge();
        }
    }
}
```

这里还用了自定义分区选择器，是因为有些消息设置了key，导致只往一个partition发，浪费了其他partition，所以重新写了个自定义分区选择器，写的也很简单，因为kafka默认用的是DefaultPartitioner，里面的partition(…)方法就是获取分区号，改下这个方法，取出里面轮询的代码即可。
如何使用自定义分区选择器，上面的yml配置中有。

```
public class MyPartition implements Partitioner {

    private final ConcurrentMap<String, AtomicInteger> topicCounterMap = new Concur

    @Override
    public int partition(String topic, Object key, byte[] keyBytes, Object value, b
        List<PartitionInfo> partitions = cluster.partitionsForTopic(topic);
        int numPartitions = partitions.size();
        int nextValue = nextValue(topic);
        List<PartitionInfo> availablePartitions = cluster.availablePartitionsForTop
        if (availablePartitions.size() > 0) {
            int part = Utils.toPositive(nextValue) % availablePartitions.size();
            return availablePartitions.get(part).partition();
        } else {
            // no partitions are available, give a non-available partition
            return Utils.toPositive(nextValue) % numPartitions;
        }
    }

    private int nextValue(String topic) {
        AtomicInteger counter = topicCounterMap.get(topic);
        if (null == counter) {
            counter = new AtomicInteger(ThreadLocalRandom.current().nextInt());
            AtomicInteger currentCounter = topicCounterMap.putIfAbsent(topic, count
            if (currentCounter != null) {
                counter = currentCounter;
            }
        }
        return counter.getAndIncrement();
    }

    @Override
    public void close() {

    }

    @Override
    public void configure(Map<String, ?> configs) {

    }
}
```

遇到的问题
如果遇到报错信息是什么java.lang.NoClassDefFoundError: org/springframework/kafka/listener
java.lang.NoSuchMethodError: org.apache.kafka.clients.consumer.Consumer.poll之类的话，基本就是版本问题了。

还有就是手动提交需要设置ackMode，不然报错
org.springframework.messaging.converter.MessageConversionException。

- RECORD :当listener一读到消息，就提交offset
- BATCH : poll() 函数读取到的所有消息,就提交offset
- TIME : 当超过设置的ackTime ， 即提交Offset
- COUNT ： 当超过设置的COUNT，即提交Offset
- COUNT_TIME ： TIME和COUNT两个条件都满足，提交offset
- MANUAL ： Acknowledgment.acknowledge()即提交Offset，和Batch类似
- MANUAL_IMMEDIATE： Acknowledgment.acknowledge()被调用即提交Offset

自定义配置

kafka-server版本装的是0.11.x，但是springboot又想用高版本的2.x怎么办？自己配置
首先是配置参数的获取，需要其他的设置自行添加

```
@ConfigurationProperties("kafka.prop")
public class KafkaProperties {

    /**
     * kafka生产者ack机制
     */
    private String acks = "1";

    /**
     * kafka并发数量
     */
    private String concurrency = "12";

    private String bootstrapServers;

    private String groupID;

```

```
    private int bufferMemory = 1024 * 1024 * 32;

    private int lingerMS = 5;

    private boolean enableAutoCommit = true;

    private int batchSize = 1024 * 16;

    /**
     * Key序列化
     */
    private String keySerializer = "org.apache.kafka.common.serialization.StringSer

    /**
     * Value序列化
     */
    private String valueSerializer = "org.apache.kafka.common.serialization.StringS

    /**
     * Key反序列化
     */
    private String keyDeserializer = "org.apache.kafka.common.serialization.StringD

    /**
     * Value反序列化
     */
    private String valueDeserializer = "org.apache.kafka.common.serialization.Strin

    public String getBootstrapServers() {
        return bootstrapServers;
    }

    public void setBootstrapServers(String bootstrapServers) {
        this.bootstrapServers = bootstrapServers;
    }

    public String getAcks() {
        return acks;
    }

    public void setAcks(String acks) {
        this.acks = acks;
    }

    public int getRetries() {
        return retries;
    }

    public void setRetries(int retries) {
        this.retries = retries;
    }

    public int getBufferMemory() {
        return bufferMemory;
    }

    public void setBufferMemory(int bufferMemory) {
        this.bufferMemory = bufferMemory;
    }

    public int getLingerMS() {
        return lingerMS;
    }

    public void setLingerMS(int lingerMS) {
        this.lingerMS = lingerMS;
    }

    public int getBatchSize() {
        return batchSize;
    }

    public void setBatchSize(int batchSize) {
        this.batchSize = batchSize;
    }

    public String getKeyDeserializer() {
        return keyDeserializer;
    }

    public void setKeyDeserializer(String keyDeserializer) {
        this.keyDeserializer = keyDeserializer;
    }

    public String getValueDeserializer() {
        return valueDeserializer;
    }

    public void setValueDeserializer(String valueDeserializer) {
        this.valueDeserializer = valueDeserializer;
    }

    public String getKeySerializer() {
        return keySerializer;
    }

    public void setKeySerializer(String keySerializer) {
        this.keySerializer = keySerializer;
    }

    public String getValueSerializer() {
        return valueSerializer;
    }

    public void setValueSerializer(String valueSerializer) {
        this.valueSerializer = valueSerializer;
    }

    public boolean getEnableAutoCommit() {
        return enableAutoCommit;
    }

    public void setEnableAutoCommit(boolean enableAutoCommit) {
        this.enableAutoCommit = enableAutoCommit;
    }

    public String getGroupID() {
        return groupID;
    }

    public void setGroupID(String groupID) {
        this.groupID = groupID;
    }

    public String getTopics() {
        return topics;
    }

    public void setTopics(String topics) {
        this.topics = topics;
    }

    public String getTopicForAll() {
        return topicForAll;
    }

    public void setTopicForAll(String topicForAll) {
        this.topicForAll = topicForAll;
    }
```

```
        return concurrency;
    }

    public void setConcurrency(String concurrency) {
        this.concurrency = concurrency;
    }
}
```

然后是kafka配置类，主要就是生产者消费者还有KafkaTemplate的一些配置

```
@Configuration
@ConditionalOnProperty(prefix = "kafka.prop", value = "enabled", havingValue = "tru
@EnableConfigurationProperties(KafkaProperties.class)
@EnableKafka
public class KafkaConfigurer {

    @Autowired
    private KafkaProperties kafkaProperties;

    @Bean
    public DefaultKafkaProducerFactory<String, String> kafkaProducerFactory() {
        Map<String, Object> paras = new HashMap<>();
        paras.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getBoots
        paras.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, kafkaProperties.getKe
        paras.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, kafkaProperties.get
        paras.put(ProducerConfig.ACKS_CONFIG,kafkaProperties.getAcks());
        return new DefaultKafkaProducerFactory<>(paras);
    }

    @Bean
    @ConditionalOnBean(DefaultKafkaProducerFactory.class)
    public KafkaTemplate<String, String> kafkaTemplate(DefaultKafkaProducerFactory<
        return new KafkaTemplate<>(kafkaProducerFactory, false);
    }


    public DefaultKafkaConsumerFactory<String, String> kafkaConsumerFactory() {
        Map<String, Object> paraMap = new HashMap<>();
        paraMap.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaProperties.getBoo
        paraMap.put(ConsumerConfig.GROUP_ID_CONFIG, kafkaProperties.getGroupID());
        paraMap.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, kafkaProperties.g
        paraMap.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, kafkaProperties
        paraMap.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, kafkaProperties.getEn

        return new DefaultKafkaConsumerFactory<>(paraMap);
    }


    @Bean
    public KafkaListenerContainerFactory<ConcurrentMessageListenerContainer<String,
        ConcurrentKafkaListenerContainerFactory<String, String> factory = new Concu
        factory.setConsumerFactory(kafkaConsumerFactory());
        if (StringUtils.isEmpty(kafkaProperties.getConcurrency())) {
            factory.setConcurrency(30);
        } else {
            factory.setConcurrency(Integer.parseInt(kafkaProperties.getConcurrency(
        }
        factory.getContainerProperties().setPollTimeout(1000);
        // 手动提交的时候需要设置AckMode为 MANUAL或MANUAL_IMMEDIATE
        factory.getContainerProperties().setAckMode(AbstractMessageListenerContaine

        return factory;
    }
}
```

yml文件如下，只列了部分

```
kafka:
  prop:
    enabled: true
    enable-auto-commit: false
    bootstrapServers: 192.168.10.45:9092
    groupID: test
    topics: test5
    saslMechanism: PLAIN
    concurrency: 6
```

生产者发送者代码还是一样的。

选择哪种方式看实际情况，kafka还是需要在实际生产中去修改它的一些参数达到高可用高吞吐的地方，后面再纪录，目前我们的kafka发送消费消息的量并不是特别大。

Windows下kafka的安装参考文章：https://blog.csdn.net/github_38482082/article/details/82112641 maven依赖： <depend…

kafka学习笔记(四)spring boot整合kafka0.10.0.1（使用配置类） 拾荒者 3395
配置类，配置属性，以及实现收发消息的实现类都与笔记(三)的代码相同，不同的是引入的kafka的依赖, 继承kafka0.10.0.1…

kafka学习笔记(三)spring boot整合kafka0.9.0.1（使用配置类） 拾荒者 3655
spring boot 版本：1.5.6引入关于kafka的相关jar <dependency> <groupId>org.springframework.kafka&…

kafka-高级应用-SpringBoot整合kafka-小白笔记（八） 码灵的博客 184
1 SpringBoot整合kafka SpringBoot整合kafka简单方便。后文介绍了生产者、消费者、事务代码实现。 1.1 添加依赖 <depe…

Kafka的ACK机制有三种，是哪三种 linux下安装配置 MySQL5.6 1万+
Kafka producer有三种ack机制 初始化producer时在config中进行配置 0 意味着producer不等待broker同步完成的确认，继…

Java Spring Boot 2.1.3 实战Kafka配置文件server.properties参数解析 ... weixin_33948416的博客 587
Apache Kafka是开源分布式高并发消息中间件，支持每秒百万级消息并发，在互联网高并发架构：双11、电商秒杀抢购、…

Spring Kafka中关于Kafka的配置参数 冯立彬的博客 4万+
SpringKafka文档地址：https://docs.spring.io/spring-kafka/reference/htmlsingle kafka文档地址：http://kafka.apache.org/do…

kafka springboot 集成配置+测试 yang_zzu的博客 690
kafka在代码中的的使用主要是对配置文件的配置，与服务器上配置文件的配置， 实际 生产、消费 的代码很简单。 ----------…

auto.offset.reset设置 u012636124的博客 204
auto.offset.reset earliest 当各分区下有已提交的offset时，从提交的offset开始消费；无提交的offset时，从头开始消费 latest…

spring boot集成kafka（简易配置） u014535689的博客 403
maven依赖 <dependency> <groupId>org.springframework.kafka</groupId> <artifactId>spring-kafka</artifactId> </depend…

springboot之kafka配置参数 LoveMyLife 4521
kafka使用 @

springboot配置kafka生产者和消费者详解 Java_自由人 1万+
在原有pom.xml依赖下新添加一下kafka依赖ar包 <!--kafka--> <dependency> <groupId>org.springframework.kafka</groupI…