大神，快来碗里
码龄9年 暂无认证

645	1万+	76	850万+	
原创	周排名	总排名	访问	等级
7万+	1001	907	156	1434
积分	粉丝	获赞	评论	收藏

私信 关注

搜博主文章

热门文章

Springboot2（29）集成zookeeper的增删改查、节点监听、分布式读写锁、分布式计数器 31700
Springboot2（27）集成netty实现反向代理（内网穿透） 30359
Springboot2（30）集成kafka--详细讲解@KafkaListener 30084
Springboot2（44）集成canal 29271
Springboot2（22）Mybatis拦截器实现 28994

分类专栏

android 7篇
tcp通讯原理 1篇
too many open file
并发编程 20篇
springboot2 49篇
JAVA基础知识 50篇

最新评论

netty报Too many open files了（必看）看… 大神，快来碗里: 如果再好
Springboot2(4)Controller控制层讲解 &Low_Key: 强！！！
netty报Too many open files了（必看）看… &Low_Key: 还可以再好点吗?
Springboot2（22）Mybatis拦截器实现 wocan23: ExecutorPluginUtils是哪个包下的?
Springboot2（30）集成kafka--详细讲解… Leon.Hopkins: 第一句话为什么那么说"如果该topic只有一个分区，实际上再启动一 …

最新文章

Android gradle3.x中 implementation 、compile、api区别
Android9网络请求
anaconda和tensorflow安装教程

2021年 8篇 2020年 7篇
2019年 574篇 2018年 44篇
2015年 7篇 2014年 3篇
2013年 8篇

目录

操作Topic
配置
Controller层
发送消息
消费消息
GenericMessageListener
@KafkaListener
启动关闭监听
配置消息过滤器
测试

Springboot2（30）集成kafka--详细讲解@KafkaListener

原创 置顶 大神，快来碗里 2018-12-30 20:19:16 30111 收藏 19 版权

分类专栏： springboot2 kafka springboot2 文章标签： Springboot2 集成kafka详细请解@KafkaListener @KafkaListener

> 源码地址
> springboot2教程系列

> 写性能非常高，因此，经常会碰到Kafka消息队列拥堵的情况 经测试，如果该topic只有一个分区，实际上再启动一个新的消费者，没有作用 。
> ConcurrentKafkaListenerContainerFactory并且设置了factory.setConcurrency(4); （我的topic有4个分区，为了加快消费将并发设置为4，也就是有4个KafkaMessageListenerContainer）

操作Topic

配置

```
@Component
public class PrividerKafkaConfig {

    @Value("${spring.kafka.bootstrap-servers}")
    private String bootStrapServer;

    @Bean
    public KafkaAdmin kafkaAdmin() {
        Map<String, Object> props = new HashMap<>();
        //配置Kafka实例的连接地址
        props.put(AdminClientConfig.BOOTSTRAP_SERVERS_CONFIG, bootStrapServer);
        KafkaAdmin admin = new KafkaAdmin(props);
        return admin;
    }

    @Bean
    public AdminClient adminClient() {
        return AdminClient.create(kafkaAdmin().getConfig());
    }

}
```

Controller层

```
@RestController
@Slf4j
public class TopicController {

    @Autowired
    private AdminClient adminClient;

    @ApiOperation(value = "创建topic")
    @ApiImplicitParams({
           @ApiImplicitParam(name = "topicName", value = "topic名称",defaultValue =
                   required = true, dataType = "string", paramType = "query"),
            @ApiImplicitParam(name = "partitions", value = "分区数", defaultValue =
                   required = true, dataType = "int", paramType = "query"),
            @ApiImplicitParam(name = "replicationFactor", value = "副本数", defaultV
                   required = true, dataType = "int", paramType = "query")
    })
    @GetMapping("/createTopic")
    public String createTopic(String topicName,int partitions,int replicationFactor
        adminClient.createTopics(Arrays.asList(new NewTopic(topicName,partitions,(s
        return "create success";
    }

    @ApiOperation(value = "查看所有的topic")
    @GetMapping("/findAllTopic")
    public String findAllTopic() throws ExecutionException, InterruptedException {
        ListTopicsResult result = adminClient.listTopics();
        Collection<TopicListing> list = result.listings().get();
        List<String> resultList = new ArrayList<>();
        for(TopicListing topicListing : list){
            resultList.add(topicListing.name());
        }
        return JSON.toJSONString(resultList);
    }

    @ApiOperation(value = "查看topic详情")
    @ApiImplicitParams({
           @ApiImplicitParam(name = "topicName", value = "topic名称",defaultValue =
                   required = true, dataType = "string", paramType = "query")
    })
    @GetMapping("/info")
    public String topicInfo(String topicName) throws ExecutionException, Interrupte
        DescribeTopicsResult result = adminClient.describeTopics(Arrays.asList(topi
        Map<String,String> resultMap = new HashMap<>();
        result.all().get().forEach((k,v)->{
            log.info("k: "+k+" ,v: "+v.toString());
            resultMap.put(k,v.toString());
        });

        return JSON.toJSONString(resultMap);
    }

    @ApiOperation(value = "删除topic")
    @ApiImplicitParams({
           @ApiImplicitParam(name = "topicName", value = "topic名称",defaultValue =
                   required = true, dataType = "string", paramType = "query")
    })
    @GetMapping("/delete")
    public String deleteTopic(String topicName){
        DeleteTopicsResult  result = adminClient.deleteTopics(Arrays.asList(topicNa
        return  JSON.toJSONString(result.values());
    }

}
```

AdminClient常用方法还有

- 创建Topic：createTopics(Collection newTopics)
- 删除Topic：deleteTopics(Collection topics)
- 罗列所有Topic：listTopics()
- 增加分区：createPartitions(Map<String, NewPartitions> newPartitions)
- 查询Topic：describeTopics(Collection topicNames)
- 查询集群信息：describeCluster()
- 查询ACL信息：describeAcls(AclBindingFilter filter)
- 创建ACL信息：createAcls(Collection acls)
- 删除ACL信息：deleteAcls(Collection filters)
- 查询配置信息：describeConfigs(Collection resources)
- 修改配置信息：alterConfigs(Map<ConfigResource, Config> configs)
- 修改副本的日志目录：alterReplicaLogDirs(Map<TopicPartitionReplica, String> replicaAssignment)
- 查询节点的日志目录信息：describeLogDirs(Collection brokers)
- 查询副本的日志目录信息：describeReplicaLogDirs(Collection replicas)

发送消息

> KafkaTemplate发送消息是采取异步方式发送的

发送消息三种方式

```
//发送带有时间戳的消息
template.send(topic, 0, System.currentTimeMillis(), "0", msg);

//使用ProducerRecord发送消息
ProducerRecord record = new ProducerRecord(topic, msg);
template.send(record);

//使用Message发送消息
Map map = new HashMap();
map.put(KafkaHeaders.TOPIC, topic);
map.put(KafkaHeaders.PARTITION_ID, 0);
map.put(KafkaHeaders.MESSAGE_KEY, "0");
GenericMessage message = new GenericMessage(msg,new MessageHeaders(map));
template.send(message);
```

消息结果回调

```
@Component
@Slf4j
public class KafkaSendResultHandler implements ProducerListener {
    @Override
    public void onSuccess(ProducerRecord producerRecord, RecordMetadata recordMetad
        log.info("Message send success : " + producerRecord.toString());
    }

    @Override
    public void onError(ProducerRecord producerRecord, Exception exception) {
        log.info("Message send error : " + producerRecord.toString());
    }

}
```

发送同步消息

```
@GetMapping("/syncMsg")
public String syncMsg(@RequestParam String topic, @RequestParam String msg){
    try {
        template.send(topic, msg).get();
    } catch (InterruptedException e) {
        e.printStackTrace();
    } catch (ExecutionException e) {
        e.printStackTrace();
    }
    return "success";
}
```

消费消息

> Spring-Kafka中消息监听大致分为两种类型，一种是单条数据消费，一种是批量消费；

GenericMessageListener

```
@Bean
public KafkaMessageListenerContainer demoListenerContainer(ConsumerFactory consumer
    ContainerProperties properties = new ContainerProperties("topic3");

    properties.setGroupId("group1");

    /*
      //批量消费
      properties.setMessageListener(new MessageListener<Integer,String>() {
            @Override
            public void onMessage(ConsumerRecord<Integer, String> record) {
                log.info("topic3: " + record.toString());
            }
        });*/

    //批量消费
    properties.setMessageListener(
        new BatchAcknowledgingConsumerAwareMessageListener<String,String>(){
            @Override
            public void onMessage(List<ConsumerRecord<String, String>> list,
                                  Acknowledgment acknowledgment, Consumer<?, ?> con
                log.info("size:{}",list.size());
            }
        });
    return new KafkaMessageListenerContainer(consumerFactory, properties);
}
```

其它MessageListener，BatchAcknowledgingConsumerAwareMessageListener为GenericMessageListener的实现类

@KafkaListener

```
@Component
@Slf4j
public class KafkaConsumer {
    //单条消息
      @KafkaListener(topics = {"first_top2"})
      public void consumer(ConsumerRecord<?, ?> record){
          Optional<?> kafkaMessage = Optional.ofNullable(record.value());
          if (kafkaMessage.isPresent()) {
              Object message = kafkaMessage.get();
              log.info("record =" + record);
              log.info(" message =" + message);
          }
      }

      //批量消息
    @KafkaListener(topics = {"first_top"},containerFactory="batchFactory")
    public void consumerBatch(List<ConsumerRecord<?, ?>> record){
        log.info("接收到消息数量：{}",record.size());
    }
}
```

```
@Bean
public KafkaListenerContainerFactory<?> batchFactory(ConsumerFactory consumerFactor
    ConcurrentKafkaListenerContainerFactory<Integer,String> factory =
    new ConcurrentKafkaListenerContainerFactory<>();
    factory.setConsumerFactory(consumerFactory);
    factory.setConcurrency(10);
    factory.getContainerProperties().setPollTimeout(1500);
    factory.setBatchListener(true);//设置为批量消费，每个批次数量在Kafka配置参数中设置
    return factory;
}
```

application.yml

```
 messages:
    basename: i18n/Messages,i18n/Pages
  kafka:
    #bootstrap-servers: 10.10.2.138:9092,10.10.2.138:9093,10.10.2.138:9094
```

点赞7 评论5 分享 收藏19 打赏 关注 一键三连

举报

```
    default-topic: self-topic0
  consumer:
        key-deserializer: org.apache.kafka.common.serialization.StringDeserialize
        value-deserializer: org.apache.kafka.common.serialization.StringDeseriali
        group-id: myGroup998
        # 最早未被消费的offset
        auto-offset-reset: earliest
        # 批量一次最大拉取数据量
        max-poll-records: 1000
        # 自动提交
        enable-auto-commit: true
  consumer-extra:
         # 是否批量处理
         batch-listener: true
```

@KafkaListener 属性

- id：消费者的id，当GroupId没有被配置的时候，默认id为GroupId
- containerFactory：上面提到了@KafkaListener区分单数据还是多数据消费只需要配置一下注解的containerFactory属性就可以了，这里面配置的是监听容器工厂，也就是ConcurrentKafkaListenerContainerFactory，配置BeanName
- topics：需要监听的Topic，可监听多个
- topicPartitions：可配置更加详细的监听信息，必须监听某个Topic中的指定分区，或者从offset为200的偏移量开始监听
- errorHandler：监听异常处理器，配置BeanName
- groupId：消费组ID
- idIsGroup：id是否为GroupId
- clientIdPrefix：消费者Id前缀
- beanRef：真实监听容器的BeanName，需要在 BeanName前加 “__”

监听Topic中指定的分区

```
@KafkaListener(id = "id0", containerFactory="batchFactory",
               topicPartitions = { @TopicPartition(topic = TPOIC, partitions = { "0
public void listenPartition0(List<ConsumerRecord<?, ?>> records) {
    log.info("Id0 Listener, Thread ID: " + Thread.currentThread().getId());
    log.info("Id0 records size " +  records.size());

    for (ConsumerRecord<?, ?> record : records) {
        Optional<?> kafkaMessage = Optional.ofNullable(record.value());
        log.info("Received: " + record);
        if (kafkaMessage.isPresent()) {
            Object message = record.value();
            String topic = record.topic();
            log.info("p0 Received message={}",  message);
        }
    }
}
```

注解方式获取消息头及消息体

```
@KafkaListener(id = "group3", topics = "first_top3")
    public void annoListener(@Payload String data,
                             @Header(KafkaHeaders.RECEIVED_MESSAGE_KEY) String key,
                             @Header(KafkaHeaders.RECEIVED_PARTITION_ID) String par
                             @Header(KafkaHeaders.RECEIVED_TOPIC) String topic,
                             @Header(KafkaHeaders.RECEIVED_TIMESTAMP) String ts) {
        log.info(" receive : \n"+
                "data : "+data+"\n"+
                "key : "+key+"\n"+
                "partitionId : "+partition+"\n"+
                "topic : "+topic+"\n"+
                "timestamp : "+ts+"\n"
        );
    }
```

测试方法：http://127.0.0.1:8080/send?topic=first_top3&msg=message

使用Ack机制确认消费

> RabbitMQ的消费可以说是一次性的，也就是你确认消费后就**立刻从硬盘或内存中删除**，而且RabbitMQ粗糙点来说是顺序消费，像排队一样，一个个顺序消费，未被确认的消息则会重新回到队列中，等待监听器再次消费。

> 但Kafka不同，Kafka是通过最新保存偏移量进行消息消费的，而且确认消费的消息并不会立刻删除，所以我们可以重复的消费未被删除的数据，当第一条消息未被确认，而第二条消息被确认的时候，Kafka会保存第二条消息的偏移量，也就是说第一条消息再也不会被监听器所获取，除非是根据第一条消息的偏移量手动获取

把application.yml中的 `enable-auto-commit` 设置为 `false` ,设置为不自动提交

```
@Bean("ackContainerFactory")
public ConcurrentKafkaListenerContainerFactory ackContainerFactory(
    ConsumerFactory consumerFactory) {
    ConcurrentKafkaListenerContainerFactory factory =
        new ConcurrentKafkaListenerContainerFactory();
    factory.getContainerProperties().setAckMode(ContainerProperties.AckMode.MANUAL_
    factory.setConsumerFactory(consumerFactory);
    return factory;
}
```

```
@KafkaListener(id = "ack", topics = "ack",containerFactory = "ackContainerFactory")
public void ackListener(ConsumerRecord record, Acknowledgment ack) {
    log.info("receive : " + record.value());
    //手动提交
    //  ack.acknowledge();
}
```

实现消息转发

```
@KafkaListener(id = "forward", topics = "first_top4")
@SendTo("first_top2")
public String forward(String data) {
    log.info("接收到消息数量：{}",data);
    return "send msg : " + data;
}
```

启动关闭监听

```
@RestController
public class ConsumerContoller {

    @Autowired
    private KafkaListenerEndpointRegistry registry;

    @Autowired
    private ConsumerFactory consumerFactory;

    @GetMapping("/stop")
    public String stop(){
        registry.getListenerContainer("forward").pause();
        return "success";
    }
```

点赞7 评论5 分享 收藏19 打赏 关注 一键三连

举报

```
    @GetMapping("/start")
    public String start(){
       //判断监听容器是否启动，未启动则将其启动
        if (!registry.getListenerContainer("forward").isRunning()) {
            registry.getListenerContainer("forward").start();
        }
        registry.getListenerContainer("forward").resume();
        return "success";
    }

}
```

启动类要添加 `@EnableKafka`

配置消息过滤器

> 消息过滤器可以在消息抵达监听容器前被拦截，过滤器根据系统业务逻辑去筛选出需要的数据再交由KafkaListener处理。

```
/**
     * 消息过滤
     * @return
     */
    @Bean
public ConcurrentKafkaListenerContainerFactory filterContainerFactory(
    ConsumerFactory consumerFactory) {
    ConcurrentKafkaListenerContainerFactory factory =
        new ConcurrentKafkaListenerContainerFactory();
    factory.setConsumerFactory(consumerFactory);
    //配合RecordFilterStrategy使用，被过滤的信息将被丢弃
    factory.setAckDiscarded(true);
    factory.setRecordFilterStrategy(new RecordFilterStrategy() {
        @Override
        public boolean filter(ConsumerRecord consumerRecord) {
            String msg = (String) consumerRecord.value();
            if(msg.contains("abc")){
                return false;
            }
            log.info("filterContainerFactory filter : "+msg);
            //返回true将会被丢弃
            return true;
        }
    });
    return factory;
}
```

```
public class FilterListener {

    @KafkaListener(topics = {"filter_topic"},containerFactory="filterContainerFacto
    public void consumerBatch(ConsumerRecord<?, ?> record){
        Optional<?> kafkaMessage = Optional.ofNullable(record.value());
        if (kafkaMessage.isPresent()) {
            Object message = kafkaMessage.get();
            log.info("record =" + record);
            log.info("接收到消息数量：{}",message);
        }

    }
}
```

测试

生产者地址：http://127.0.0.1:8080/swagger-ui.html

“500元卡时GPU资源”，限时免费申领中！ 04-21
海量GPU计算资源，预装AI框架和开发环境，开机即用；7*24小时专家团队提供多元服务，让计算科研省时、省力、省心！

springboot 集成kafka 实现多个customer不同group caijiapeng0102的博客 1万+
springboot正常集成kafka这个网上很多资料都有些集成，我就不浪费太多篇幅和时间了，笔者找了篇还算很容易理解的博客…

优质评论可以帮助作者获得更高权重 评论

码哥 Leon.Hopkins： 第一句话为什么那么说"如果该topic只有一个分区，实际上再启动一个新的消费者，没有作用 " 难道不会随机消费么 11 天前 回复

码哥 wo41chuan_luan_ma： 大佬 受教了，帮助很大 ，在此谢过 9 月前 回复

强风吹拂： 有个事情教教，能加个qq么，1174158844 2 年前 回复

大脸的猫 回复： y 7 月前 回复

大脸的猫 回复： h 7 月前 回复

相关推荐

Spring Boot整合Kafka的简单用例(@KafkaListener注解... 8-29
第七步、启动程序、调用接口 消息监听器只监听订阅的topic的特定分区的消息 源码:https://github.com/NapWells/java_fram…

springBoot2.x集成kafka_歪歪梯的博客 4-10
以下是外部生产代码,外部生产者创建的主题,springBoot集成的kafka并不能立即分区并消费其消息。springBoot默认是5分钟…

Springboot2整合kafka的两种使用方式 冲动的仔bb博客 1万+
Springboot2整合kafkakafkadocker上安装环境Springboot2引入kafka基于注解基于客户端 kafka是一个分布式消息队列。在…

springboot+@KafkaListener 消费者参数详解 asd5629626的博客 1万+
1.1 consumer参数详解 BOOTSTRAP_SERVERS_CONFIG kafka ip+port REQUEST_TIMEOUT_MS_CONFIG 请求超时…

spring-boot 2.3.x 整合kafka_鸭鸭的博客 4-4
《spring官网 kafka》Spring for Apache Kafka(Spring Kafka)项目将核心的Spring概念应用到基于Kafka的消息传递解决方案…

SpringBoot笔记:SpringBoot2.3集成Kafka组件配置_u0110... 4-12
packagecom.demo.kafka;importlombok.extern.slf4j.Slf4j;importorg.apache.kafka.clients.consumer.ConsumerRecord;impor…

Springboot2整合kafka的两种使用方式 weixin_39249427的博客 1768
1、Springboot2整合kafka 原文链接：https://blog.csdn.net/victoylin/article/details/93409055 kafka docker上安装环境 Sprin…

spring boot 集成kafka (多线程,消费者使用kafka的原生api实现,因为@KakfkaListe… qq_40633152的博客 7514
原文链接：http://www.mamicode.com/info-detail-2078498.html 1 #kafka 2 kafka.consumer.zookeeper.connect=*:2181 3 k…

Spring Boot 中使用@KafkaListener并发批量接收消息 russle的专栏 10万+
kakfa是 我们在项目开发中经常使用的消息中间件。由于它的写性能非常高，因此，经常会碰到Kafka消息队列拥堵的情况…

Spring Boot 中使用@KafkaListener批量接收消息ack weixin_34347651的博客 2978
2019独角兽企业重金招聘Python工程师标准>>> ...

【spring-kafka】@KafkaListener详解与使用 石臻臻 2982
说明 从2.2.4版开始，您可以直接在注释上指定Kafka使用者属性，这些属性将覆盖在使用者工厂中配置的具有相同名称的所…

注解@KafkaListener批量接收消息 Asa_Prince的博客 488
之前介绍了如何在SpringBoot中集成Kafka,但是默认情况下，@KafkaListener都是一条一条消费，如果想要一次消费一个批…

使用@KafkaListener配置两个不同的Consumer监听不同kafka集群的消息 weixin_40910372的博客 1113
我们的项目中很多系统交互使用的kafka，最近遇到一个问题，原来我们的kafka需要监听消费我们自己的kafka生产的消息…

第三集 Spring for Apache Kafka 接受消息 技术宅星云-CSDN博客 3205
我们可以接受消息通过配置一个MessageListenerContainer 和提供一个消息监听或者通过使用@KafkaListener 注解 3.1 Me…

SpringBoot中@KafkaListener原理 雪落南城的博客 8370
在我们的SpringBoot工程中，只需在方法中注解 @KafkaListener(topics = {”demo_topic_01“}) 即可实现对该topic的监听 我…

KafkaListener的各种操作 my_momo_csdn的博客 8571
@KafkaListener的各种操作 通过KafkaListener可以自定义批量消费和多线程消费，通过自定义创建消费容器的工厂类，来…

点赞7 评论5 分享 收藏19 打赏 关注 一键三连 …e Blog of Forward 2万+

举报

springboot+kafka中@KafkaListener如何动态指定多个topic 本项目为springboot+kafak的整合项目，故其用了springboot中…

Spring Boot 整合 Kafka 并使用 @KafkaListener 并发批量接收消息 storm_fury 2073

注册 KafkaListenerContainerFactory import org.apache.kafka.clients.CommonClientConfigs; import org.apache.kafka.clien…

SpringKafka动态指定@KafkaListener的topics和groupId u014259503的博客 2461

1.@KafkaListener @KafkaListener是kafka的消费者，topics是其主题名，groupId是组名； 属性值一般只支持常量，再集群…

spring-kafka通过@KafkaListener实现消费者监听流程分析 自律使我自由 3983

首先通过@EnableKafka注解，注入KafkaBootstrapConfiguration类 @Target(ElementType.TYPE) @Retention(RetentionP…

SpringBoot -- Kafka(二) Demo 代码行间的无聊生活的博客 9253

前置工作 Kafka 、zookeeper环境已经完成 JDK完成安装（kafka环境依赖jvm） 了解kafka、zookeeper各种的作用 Demo …

@KafkaListener通过配置加载多个topic mrxiky的专栏 1万+

接到领导的一个需求，希望封装一下kafka的消费者，可以从配置读取topic进行消费；一开始首先想到的是用java kafka…

©2020 CSDN 皮肤主题: 猿与汪的秘密 设计师:白松林 返回首页

关于我们 招贤纳士 广告服务 开发助手 400-660-0108 kefu@csdn.net 在线客服 工作时间 8:30-22:00

公安备案号11010502030143 京ICP备19004658号 京网文〔2020〕1039-165号 经营性网站备案信息 北京互联网违法和不良信息举报中心
网络110报警服务 中国互联网举报中心 家长监护 Chrome商店下载 ©1999-2021北京创新乐知网络技术有限公司 版权与免责声明 版权申诉
出版物许可证 营业执照

点赞7 评论5 分享 收藏19 打赏 关注 一键三连

举报