Spring Boot接入RocketMQ

Spring Boot对RocketMQ进行了简化封装，提供了rocketmq-spring-boot-starter组件对RocketMQ进行了直接继承，本讲我们快速上手Spring Boot如何开发RocketMQ生产者与消费者

生产者

第一步，新建Spring Boot工程，依赖rocketmq-spring-boot-starter

```
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-spring-boot-starter</artifactId>
    <version>2.2.1</version>
</dependency>
```

第二步，配置application.properties

```
# 应用名称
spring.application.name=spb-rocketmq
# 应用服务 WEB 访问端口
server.port=8000
# nameserver
#如果构建nameserver集群用;号分割多个nameserver，运行时Spring Boot从前向后依次尝试连接
#rocketmq.name-server=192.168.31.103:9876;192.168.31.113:9876
rocketmq.name-server=192.168.31.103:9876
# 生产者组
rocketmq.producer.group=producer-group
# 异步发送时，重试次数
rocketmq.producer.retry-times-when-send-async-failed=5
```

第三步，开发生产者服务

其中RocketMQTemplate对象是Spring Boot对RocketMQ整合对象，封装了asyncSend、convertAndSend等发送消息。

```
@Slf4j
@RestController
public class ProducerController {
    @Resource
    private RocketMQTemplate rocketMQTemplate;

    @GetMapping("/producer/send")
    public String testSendMessage(int num) throws InterruptedException {
        String ret = null;
        for(int i = 0 ; i < num ; i++) {
            //异步发送，Callback监听
            Message message = new Message("tax-data", "2022s1", "id-" + i, ("Tax-data-" + i).getBytes(StandardCharsets.UTF_8));
            rocketMQTemplate.asyncSend("tax-data", message, new SendCallback() {
                @Override
                public void onSuccess(SendResult sendResult) {
                    log.info("发送成功：{}", sendResult.toString());
                }

                @Override
                public void onException(Throwable throwable) {
                    log.info("发送异常：{}", throwable.toString());
                }
            });
            Thread.sleep(100);
        }
        return "OK";
    }
}
```

第四步，启动应用，访问http://localhost:8000/producer/send?num=10，向Broker发送十条数据，主题为tax-data

消息会随机投放到不同队列中。

运行结果如下：

```
INFO 19628 --- [ublicExecutor_7] c.i.r.s.controller.ProducerController    : 发送成功：
SendResult [sendStatus=SEND_OK,
msgId=7F0000014CAC18B4AAC28424C413001E,
offsetMsgId=C0A81F6900002A9F0000000000404A14,
messageQueue=MessageQueue [topic=tax-data, brokerName=broker-a,
queueId=3], queueOffset=4058]
INFO 19628 --- [ublicExecutor_8] c.i.r.s.controller.ProducerController    : 发送成功：
SendResult [sendStatus=SEND_OK,
msgId=7F0000014CAC18B4AAC28424C486001F,
offsetMsgId=C0A81F6900002A9F0000000000404BF1,
messageQueue=MessageQueue [topic=tax-data, brokerName=broker-a,
queueId=0], queueOffset=4171]
INFO 19628 --- [ublicExecutor_9] c.i.r.s.controller.ProducerController    : 发送成功：
SendResult [sendStatus=SEND_OK,
msgId=7F0000014CAC18B4AAC28424C4F30020,
offsetMsgId=C0A81F6900002A9F0000000000404DCE,
messageQueue=MessageQueue [topic=tax-data, brokerName=broker-a,
queueId=0], queueOffset=4172]
INFO 19628 --- [blicExecutor_10] c.i.r.s.controller.ProducerController    : 发送成功：
SendResult [sendStatus=SEND_OK,
msgId=7F0000014CAC18B4AAC28424C55F0021,
offsetMsgId=C0A81F6900002A9F0000000000404FAB,
messageQueue=MessageQueue [topic=tax-data, brokerName=broker-a,
queueId=1], queueOffset=4085]
INFO 19628 --- [blicExecutor_11] c.i.r.s.controller.ProducerController    : 发送成功：
SendResult [sendStatus=SEND_OK,
msgId=7F0000014CAC18B4AAC28424C5CB0022,
offsetMsgId=C0A81F6900002A9F0000000000405188,
messageQueue=MessageQueue [topic=tax-data, brokerName=broker-a,
queueId=3], queueOffset=4059]
INFO 19628 --- [blicExecutor_12] c.i.r.s.controller.ProducerController    : 发送成功:
SendResult [sendStatus=SEND_OK,
msgId=7F0000014CAC18B4AAC28424C6360023,
offsetMsgId=C0A81F6900002A9F0000000000405365,
messageQueue=MessageQueue [topic=tax-data, brokerName=broker-a,
queueId=0], queueOffset=4173]
INFO 19628 --- [ublicExecutor_1] c.i.r.s.controller.ProducerController    : 发送成功:
SendResult [sendStatus=SEND_OK,
msgId=7F0000014CAC18B4AAC28424C6A10024,
offsetMsgId=C0A81F6900002A9F0000000000405542,
messageQueue=MessageQueue [topic=tax-data, brokerName=broker-a,
queueId=2], queueOffset=4171]
INFO 19628 --- [ublicExecutor_2] c.i.r.s.controller.ProducerController    : 发送成功:
SendResult [sendStatus=SEND_OK,
msgId=7F0000014CAC18B4AAC28424C70B0025,
offsetMsgId=C0A81F6900002A9F000000000040571F,
messageQueue=MessageQueue [topic=tax-data, brokerName=broker-a,
queueId=3], queueOffset=4060]
INFO 19628 --- [ublicExecutor_3] c.i.r.s.controller.ProducerController    : 发送成功:
SendResult [sendStatus=SEND_OK,
msgId=7F0000014CAC18B4AAC28424C7760026,
offsetMsgId=C0A81F6900002A9F00000000004058FC,
messageQueue=MessageQueue [topic=tax-data, brokerName=broker-a,
queueId=2], queueOffset=4172]
INFO 19628 --- [ublicExecutor_4] c.i.r.s.controller.ProducerController    : 发送成功:
SendResult [sendStatus=SEND_OK,
msgId=7F0000014CAC18B4AAC28424C7E20027,
offsetMsgId=C0A81F6900002A9F0000000000405AD9,
messageQueue=MessageQueue [topic=tax-data, brokerName=broker-a,
queueId=2], queueOffset=4173]
```

消费者

第一步，新建Spring Boot工程，依赖rocketmq-spring-boot-starter

```
<dependency>
   <groupId>org.apache.rocketmq</groupId>
   <artifactId>rocketmq-spring-boot-starter</artifactId>
   <version>2.2.1</version>
</dependency>
```

第二步，配置application.properties

配置消费者组、消息模式等与消费者相关设置

```
spring.application.name=spb-rocketmq-consumer
rocketmq.name-server=192.168.31.103:9876
#消费者组
rocketmq.consumer.group=consumer-group
#消息模式CLUSTERING-集群模式
rocketmq.consumer.message-model=CLUSTERING
```

第三步，开发消费者监听器

@RocketMQMessageListener监听器用于监听Broker队列，默认推送Push，实现RocketMQListener获取发送的数据结果。

onMessage方法执行成功自动ack确认接收

onMessage方法抛出异常自动nack拒绝接收，broker下次会重新发送

```
@RocketMQMessageListener(topic = "tax-data",consumerGroup =
"${rocketmq.consumer.group}",selectorExpression = "*")
@Slf4j
@Component
public class Consumer implements RocketMQListener<MessageExt>{
    @Override
    public void onMessage(MessageExt s) {
        log.info("接收到消息:{}",s);
    }
}
```

第四步，启动应用生产者发送数据会自动被消费者消费。