RabbitMQ高级特性

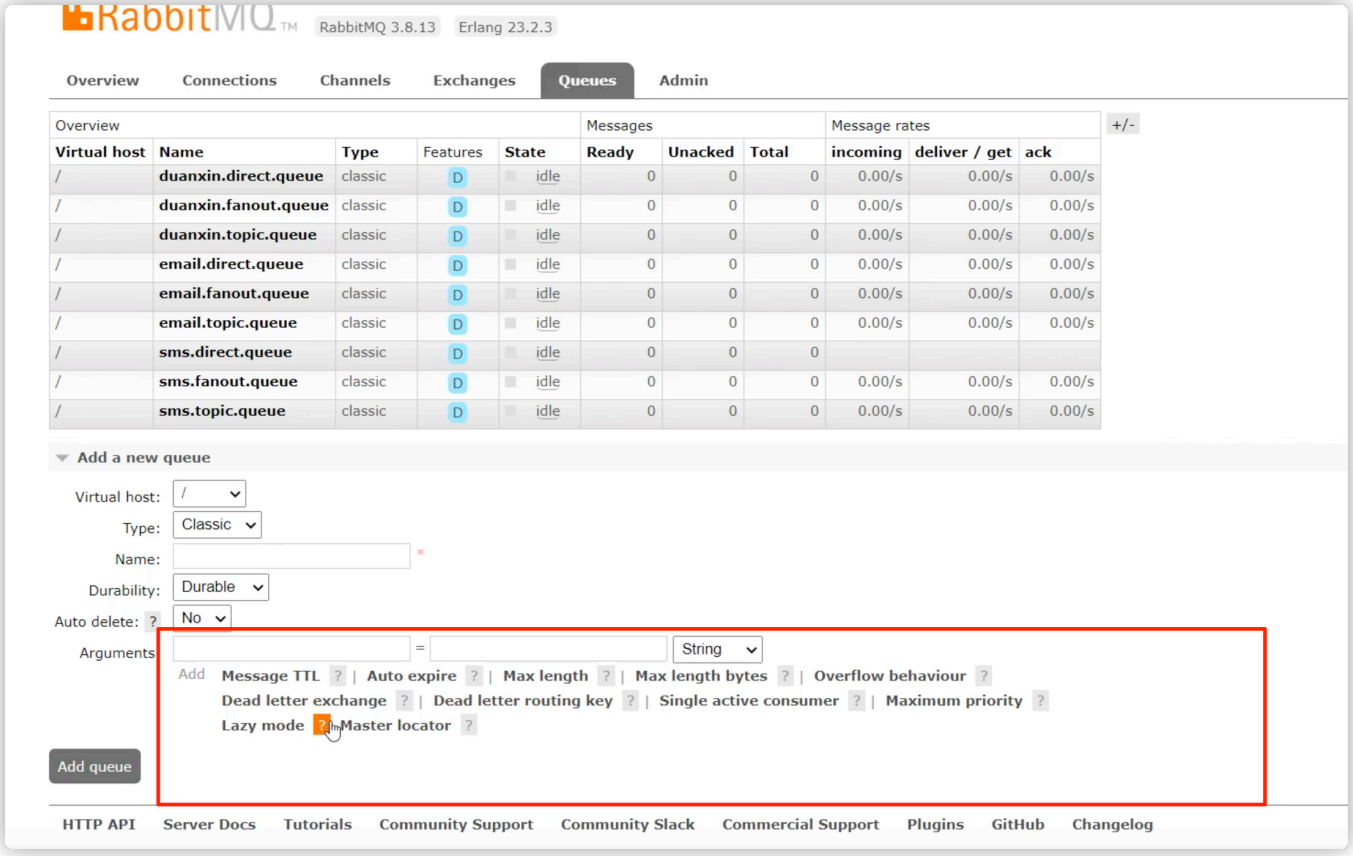
- 消息可靠性投递
- Consumer ACK
- 消费端限流
- TTL
- 死信队列
- 延迟队列
- 日志与监控
- 消息可靠性分析与追踪
- 管理

RabbitMQ应用问题

- 消息可靠性保障
- 消息幂等性处理

RabbitMQ集群搭建

- RabbitMQ高可用集群



Overview					Messages			Message rates			+/-
Virtual host	Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
/	duanxin.direct.queue	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	
/	duanxin.fanout.queue	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	
/	duanxin.topic.queue	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	
/	email.direct.queue	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	
/	email.fanout.queue	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	
/	email.topic.queue	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	
/	sms.direct.queue	classic	D	idle	0	0	0				
/	sms.fanout.queue	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	
/	sms.topic.queue	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s	

过期时间TTL表示可以对消息设置预期的时间，在这个时间内都可以被消费者接收获取；过了之后消息将自动被删除。RabbitMQ可以对 **消息和队列** 设置TTL。目前有两种方法可以设置。

消息可靠投递

1.1 消息的可靠投递

在使用 RabbitMQ 的时候，作为消息发送方希望杜绝任何消息丢失或者投递失败场景。RabbitMQ 为我们提供了两种方式用来控制消息的投递可靠性模式。

- confirm 确认模式
- return 退回模式

rabbitmq 整个消息投递的路径为：

producer--->rabbitmq broker--->exchange--->queue--->consumer

- 消息从 producer 到 exchange 则会返回一个 confirmCallback 。
- 消息从 exchange-->queue 投递失败则会返回一个 returnCallback 。

我们将利用这两个 callback 控制消息的可靠性投递

代码

rabbitmq.properties

```
rabbitmq.host=172.16.98.133
rabbitmq.port=5672
rabbitmq.username=guest
rabbitmq.password=guest
rabbitmq.virtual-host=/
```

spring-rabbitmq-producer.xml

```
<!-- 定义rabbitmq connectionFactory -->
<rabbit:connection-factory id="connectionFactory" host="${rabbitmq.host}"
                           port="${rabbitmq.port}"
                           username="${rabbitmq.username}"
                           password="${rabbitmq.password}"
                           virtual-host="${rabbitmq.virtual-host}"
                           publisher-confirms="true"
                           publisher-returns="true"
/>
<!--定义管理交换机、队列-->
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:rabbit="http://www.springframework.org/schema/rabbit"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context
       https://www.springframework.org/schema/context/spring-context.xsd
       http://www.springframework.org/schema/rabbit
       http://www.springframework.org/schema/rabbit/spring-rabbit.xsd">
    <!--加载配置文件-->
    <context:property-placeholder location="classpath:rabbitmq.properties"/>

    <!-- 定义rabbitmq connectionFactory -->
    <rabbit:connection-factory id="connectionFactory" host="${rabbitmq.host}"
                               port="${rabbitmq.port}"
                               username="${rabbitmq.username}"
                               password="${rabbitmq.password}"
                               virtual-host="${rabbitmq.virtual-host}"
                               publisher-confirms="true"
                               publisher-returns="true"
    />
    <!--定义管理交换机、队列-->
    <rabbit:admin connection-factory="connectionFactory"/>

    <!--定义rabbitTemplate对象操作可以在代码中方便发送消息-->
    <rabbit:template id="rabbitTemplate" connection-factory="connectionFactory"/>



    <!--消息可靠性投递（生产端）-->
    <rabbit:queue id="test_queue_confirm" name="test_queue_confirm"></rabbit:queue>
    <rabbit:direct-exchange name="test_exchange_confirm">
        <rabbit:bindings>
            <rabbit:binding queue="test_queue_confirm" key="confirm"></rabbit:binding>
        </rabbit:bindings>
    </rabbit:direct-exchange>

</beans>
```

如果是springboot

```java
    @Bean
    public ConnectionFactory connectionFactory() {
        CachingConnectionFactory connectionFactory = new CachingConnectionFactory();
        connectionFactory.setHost("localhost");
        connectionFactory.setPort(5672);
        connectionFactory.setPassword("guest");
        connectionFactory.setUsername("guest");
```

```
 connectionFactory.setPublisherConfirmType(CachingConnectionFactory.ConfirmType.CORRELATED);
        connectionFactory.setPublisherReturns(true);
        connectionFactory.createConnection();
        return connectionFactory;
    }


    @Bean
    public RabbitListenerContainerFactory rabbitListenerContainerFactory(ConnectionFactory
connectionFactory){
        SimpleRabbitListenerContainerFactory factory = new
SimpleRabbitListenerContainerFactory();
        factory.setConnectionFactory(connectionFactory);
        return factory;
    }
    @Bean
    @Qualifier("rabbitTemplate")
    public RabbitTemplate rabbitTemplate(ConnectionFactory connectionFactory) {
        RabbitTemplate rabbitTemplate = new RabbitTemplate(connectionFactory);
        //开启mandatory模式（开启失败回调）
        rabbitTemplate.setMandatory(true);
        //添加失败回调方法
        rabbitTemplate.setReturnCallback((message, replyCode, replyText, exchange, routingKey) -
> {
            log.info("message:{}, replyCode:{}, replyText:{}, exchange:{}, routingKey:{}",
                    message, replyCode, replyText, exchange, routingKey);
        });
        // 添加发送方确认模式方法
        rabbitTemplate.setConfirmCallback((correlationData, ack, cause) ->
                log.info("correlationData:{}, ack:{}, cause:{}",
                        correlationData.getId(), ack, cause));
        return rabbitTemplate;
    }
```

ProducerTest

```
  /**
     * 确认模式：
     * 步骤：
     * 1. 确认模式开启：ConnectionFactory中开启publisher-confirms="true"
     * 2. 在rabbitTemplate定义ConfirmCallBack回调函数
     */
    @Test
    public void testConfirm() {

        //2. 定义回调
        rabbitTemplate.setConfirmCallback(new RabbitTemplate.ConfirmCallback() {
            /**
             *
```

```
             * @param correlationData 相关配置信息
             * @param ack   exchange交换机 是否成功收到了消息。true 成功，false代表失败
             * @param cause 失败原因
             */
            @Override
            public void confirm(CorrelationData correlationData, boolean ack, String cause) {
                System.out.println("confirm方法被执行了....");

                if (ack) {
                    //接收成功
                    System.out.println("接收成功消息" + cause);
                } else {
                    //接收失败
                    System.out.println("接收失败消息" + cause);
                    //做一些处理，让消息再次发送。
                }
            }
        });

        //3. 发送消息
        rabbitTemplate.convertAndSend("test_exchange_confirm111", "confirm", "message
confirm....");
    }


 /**
     * 回退模式： 当消息发送给Exchange后，Exchange路由到Queue失败是 才会执行 ReturnCallBack
     * 步骤：
     * 1. 开启回退模式:publisher-returns="true"
     * 2. 设置ReturnCallBack
     * 3. 设置Exchange处理消息的模式：
     * 1. 如果消息没有路由到Queue，则丢弃消息（默认）
     * 2. 如果消息没有路由到Queue，返回给消息发送方ReturnCallBack
     */

    @Test
    public void testReturn() {

        //设置交换机处理失败消息的模式
        rabbitTemplate.setMandatory(true);

        //2.设置ReturnCallBack
        rabbitTemplate.setReturnCallback(new RabbitTemplate.ReturnCallback() {
            /**
             *
             * @param message   消息对象
             * @param replyCode 错误码
             * @param replyText 错误信息
             * @param exchange  交换机
             * @param routingKey 路由键
             */
            @Override
```

```
            public void returnedMessage(Message message, int replyCode, String replyText, String
exchange, String routingKey) {
                System.out.println("return 执行了....");

                System.out.println(message);
                System.out.println(replyCode);
                System.out.println(replyText);
                System.out.println(exchange);
                System.out.println(routingKey);

                //处理
            }
        });


        //3. 发送消息
        rabbitTemplate.convertAndSend("test_exchange_confirm", "confirm", "message
confirm....");
    }
```

1.1 消息的可靠投递小结

- 设置ConnectionFactory的publisher-confirms="true" 开启 确认模式。
- 使用rabbitTemplate.setConfirmCallback设置回调函数。当消息发送到exchange后回调confirm方法。在方法中判断ack，如果为true，则发送成功，如果为false，则发送失败，需要处理。

1.1 消息的可靠投递小结

- 设置ConnectionFactory的publisher-confirms="true" 开启 确认模式。
- 使用rabbitTemplate.setConfirmCallback设置回调函数。当消息发送到exchange后回调confirm方法。在方法中判断ack，如果为true，则发送成功，如果为false，则发送失败，需要处理。
- 设置ConnectionFactory的publisher-returns="true" 开启 退回模式。
- 使用rabbitTemplate.setReturnCallback设置退回函数，当消息从exchange路由到queue失败后，如果设置了rabbitTemplate.setMandatory(true)参数，则会将消息退回给producer。并执行回调函数returnedMessage。
- 在RabbitMQ中也提供了事务机制，但是性能较差，此处不做讲解。

使用channel下列方法，完成事务控制：

txSelect(), 用于将当前channel设置成transaction模式
txCommit()，用于提交事务
txRollback(),用于回滚事务