RabbitMq 从零开始

作者：tll

原文链接：https://ld246.com/article/1614070241202

来源网站：链滴

yml

```yaml
rabbitmq:
    host: localhost
    port: 7672
    username: guest
    password: guest
    publisher-confirms: true   #开启发送确认
    publisher-returns: true  #开启发送失败回退

    #开启ack
    listener:
      direct:
        acknowledge-mode: manual
      simple:
        acknowledge-mode: manual #采取手动应答
        #concurrency: 1 # 指定最小的消费者数量
        #max-concurrency: 1 #指定最大的消费者数量
        retry:
          enabled: true # 是否支持重试
```

配置文件

```
package com.yss.screenmonitor.mq;

import org.springframework.amqp.core.Binding;
import org.springframework.amqp.core.BindingBuilder;
import org.springframework.amqp.core.DirectExchange;
import org.springframework.amqp.core.Queue;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * @author tanglonglong \(--)/
 * @date 2021/2/22 17:29
 */
@Configuration
public class MqConfig {
    @Bean
    public Queue helloQueue(){
        Queue hello = new Queue("hello");
        return hello;
    }

    @Bean(name="getDirectExchangeTx")
    public DirectExchange getDirectExchangeTx(){
        return new DirectExchange("directExchangeTx", true, false);
    }

    @Bean(name="getQueueTx")
    public Queue getQueueTx(){
        return new Queue("directQueueTx", true, false, false);
    }
```

```
//每个queue绑定exchange,routing key，然后发送消息时候指定routing key，指定exchange.
//三种类型的Exchange：direct ，fanout和topic
    @Bean
    public Binding getDirectExchangeQueueTx(
            @Qualifier(value="getDirectExchangeTx") DirectExchange getDirectExchangeTx,
            @org.springframework.beans.factory.annotation.Qualifier(value="getQueueTx") Queu
 getQueueTx){
        return BindingBuilder.bind(getQueueTx).to(getDirectExchangeTx).with("directQueueTxR
utingKey");
    }
}
```

消费者

```
/**
 * @author tanglonglong \(--)/
 * @date 2021/2/22 17:27
 */
@Component
@RabbitListener(queues = "hello")
public class WatchHello {
    @RabbitHandler
    public void get(String str, Channel chennel, Message message) throws InterruptedException
{
        try {
            chennel.basicNack(message.getMessageProperties().getDeliveryTag(),false,false);
//            chennel.basicAck(message.getMessageProperties().getDeliveryTag(),false);
        } catch (
    IOException e) {
            e.printStackTrace();
        }
//        Thread.sleep(10);
        System.out.println(str+"1");
    }
}
```

生产者

```
package com.yss.screenmonitor.mq;

import org.springframework.amqp.core.Message;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.amqp.rabbit.support.CorrelationData;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import javax.annotation.PostConstruct;
import java.util.Date;

/**
 * @author tanglonglong \(--)/
 * @date 2021/2/22 17:22
 */
@Service
```

```
public class MQUtil implements RabbitTemplate.ConfirmCallback, RabbitTemplate.ReturnCall
ack {
//    @Autowired
//    private AmqpTemplate rabbitmqTemplate;
    @Autowired
    RabbitTemplate rabbitTemplate;

    /**
     * PostConstruct: 用于在依赖关系注入完成之后需要执行的方法上，以执行任何初始化.
     */
    @PostConstruct
    public void init() {
        //指定 ConfirmCallback
        rabbitTemplate.setConfirmCallback(this);
        //指定 ReturnCallback
        rabbitTemplate.setReturnCallback(this);
    }
    public void send(String msg) throws InterruptedException {
        String content = msg+ new Date();
        for (int i = 0; i < 1000; i++) {
        Thread.sleep(20);
            this.rabbitTemplate.convertAndSend("hello", (Object)content,new CorrelationData(Int
ger.valueOf(i).toString()));
            this.rabbitTemplate.convertAndSend("directExchangeTx", "directQueueTxRoutingKey",
Object)content,new CorrelationData(Integer.valueOf(i).toString()));
        }
    }

    /**
     * Confirmation callback.
     * 有没有到交换机
     *
     * @param correlationData correlation data for the callback.
     * @param ack true for ack, false for nack
     * @param cause An optional cause, for nack, when available, otherwise null.
     */
    @Override
    public void confirm(CorrelationData correlationData, boolean ack, String cause) {
        String id = correlationData.getId();
        System.out.println(id+ack+cause);
    }

    /**
     * Returned message callback.
     * 有没有从交换机到queue，到了不调用
     * @param message the returned message.
     * @param replyCode the reply code.
     * @param replyText the reply text.
     * @param exchange the exchange.
     * @param routingKey the routing key.
     */
    @Override
    public void returnedMessage(Message message, int replyCode, String replyText, String exc
ange, String routingKey) {
```

```
        System.out.println(message.getMessageProperties().toString()+routingKey+replyText+e
change+routingKey);
    }
}
```