链滴

SpringBoot 整合 RabbitMQ

作者：AutisticV5

原文链接：https://ld246.com/article/1573607081836

来源网站：链滴

direct模式，fanout模式，topic模式，死信队列，发送方消息确认机制实现

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
MLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/
aven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.0.4.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.wzx</groupId>
    <artifactId>springboot-mq</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>springboot-mq</name>
    <description>Demo project for Spring Boot mq</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-starter-amqp</artifactId>
        </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>
```

Consumer.java

```
package com.wzx.demo.conf;

import com.rabbitmq.client.Channel;
import org.springframework.amqp.core.Message;
import org.springframework.amqp.rabbit.annotation.RabbitHandler;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.amqp.support.AmqpHeaders;
import org.springframework.messaging.handler.annotation.Header;
import org.springframework.stereotype.Component;

import java.io.IOException;

/**
 * 消息消费者
 *
 * @author wangzhengxing
 * @version 1.0
 * @Date 2019-10-31 09:14
 */
@Component
public class Consumer {
```

```
    @RabbitHandler
    @RabbitListener(queues = "direct.queue")
    public void executeDirect(Message message, Channel channel, @Header(AmqpHeaders.DE
IVERY_TAG) long tag) throws IOException {
        System.out.println("监听到直连队列有消息进来");
        //确认消息
        channel.basicAck(tag, false);
        System.out.println("message: " + message + "; channel: " + channel);
    }

    @RabbitHandler
    @RabbitListener(queues = "dead.queue")
    public void executeDead(Message message, Channel channel) {
        System.out.println("监听到死信队列有消息进来");
        System.out.println("message: " + message + "; channel: " + channel);
    }

    @RabbitHandler
    @RabbitListener(queues = "fanout.queue")
    public void executeFanout(Message message, Channel channel) {
        System.out.println("监听到广播队列有消息进来");
        System.out.println("message: " + message + "; channel: " + channel);
    }

    @RabbitHandler
    @RabbitListener(queues = "topic.queue")
    public void executeTopic(Message message, Channel channel) {
        System.out.println("监听到主题队列有消息进来");
        System.out.println("message: " + message + "; channel: " + channel);
    }
}
```

ExchangeEnum.java

```
package com.wzx.demo.conf;

/**
 * RabbitMQ交换配置枚举
 * @Author Win10
 * @Date 2019年10月31日 08:54
 */
public enum ExchangeEnum {
    DIRECT_DEAD_EXCHANGE("direct.dead.exchange"),
    DIRECT_EXCHANGE("direct.exchange"),
    DEAD_EXCHANGE("dead.exchange"),
    FANOUT_EXCHANGE("fanout.exchange"),
    TOPIC_EXCHANGE("topic.exchange");

    private String exchangeName;

    ExchangeEnum(String exchangeName) {
        this.exchangeName = exchangeName;
```

```
    }

    public String getExchangeName() {
        return exchangeName;
    }

    public void setExchangeName(String exchangeName) {
        this.exchangeName = exchangeName;
    }
}
```

QueueEnum.java

```
package com.wzx.demo.conf;

/**
 * RabbitMQ队列配置枚举
 * @Author Win10
 * @Date 2019年10月31日 08:54
 */
public enum QueueEnum {
    DIRECT_DEAD_QUEUE("direct.dead.queue", "direct.dead.route"),
    DIRECT_QUEUE("direct.queue", "direct.route"),
    DEAD_QUEUE("dead.queue", "dead.route"),
    FANOUT_QUEUE("fanout.queue", "fanout.route"),
    TOPIC_QUEUE("topic.queue", "topic.*");

    /**
     * 队列名称
     */
    private String queueName;
    /**
     * 队列路由键
     */
    private String routingKey;

    QueueEnum(String queueName, String routingKey) {
        this.queueName = queueName;
        this.routingKey = routingKey;
    }

    public String getQueueName() {
        return queueName;
    }

    public void setQueueName(String queueName) {
        this.queueName = queueName;
    }

    public String getRoutingKey() {
        return routingKey;
    }
```

```
    public void setRoutingKey(String routingKey) {
        this.routingKey = routingKey;
    }
}
```

QueueMessage.java

```
package com.wzx.demo.conf;

import org.springframework.amqp.rabbit.core.RabbitTemplate;

/**
 * 消息队列接口
 * @Author Win10
 * @Date 2019年10月31日 08:52
 */
public interface QueueMessage extends RabbitTemplate.ConfirmCallback, RabbitTemplate.Re
urnCallback {
}
```

QueueMessageService.java

```
package com.wzx.demo.conf;

import org.springframework.amqp.core.Message;
import org.springframework.amqp.core.MessagePostProcessor;
import org.springframework.amqp.core.MessageProperties;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.amqp.rabbit.support.CorrelationData;
import org.springframework.context.annotation.Configuration;

import javax.annotation.Resource;
import java.util.UUID;

/**
 * 消息队列业务具体逻辑实现
 *
 * @author wangzhengxing
 * @version 1.0
 * @Date 2019-10-31 09:00
 */
@Configuration
public class QueueMessageService implements QueueMessage {
    /**
     * 消息队列模板
     */
    @Resource
    private RabbitTemplate rabbitTemplate;

    public void sendDeadMsg(String msg) throws Exception {
        /**
         * 设置回调为当前类对象
```

```
         */
        rabbitTemplate.setConfirmCallback(this::confirm);
        rabbitTemplate.setReturnCallback(this::returnedMessage);
        //当参数为true时，交换器无法根据自身的类型和路由键找到一个符合条件的队列，那么RabbitMQ会调用Return命令将消息返回给生产者；否则消息直接被丢弃
        rabbitTemplate.setMandatory(true);
        /**
         * 构建回调ID为UUID
         */
        CorrelationData correlationData = new CorrelationData(UUID.randomUUID().toString());
        /**
         * 声明消息处理器
         */
        MessagePostProcessor messagePostProcessor = messageProcessor -> {
            MessageProperties messageProperties = messageProcessor.getMessageProperties();
            //设置编码
            messageProperties.setContentEncoding("utf-8");
            //设置过期时间,毫秒值
            messageProperties.setExpiration("2000");
            return messageProcessor;
        };
        /**
         * 发送消息到消息队列,10000毫秒后过期，成为死信
         */
        rabbitTemplate.convertAndSend(ExchangeEnum.DIRECT_DEAD_EXCHANGE.getExchangName(), QueueEnum.DIRECT_DEAD_QUEUE.getRoutingKey(), msg, messagePostProcessor, correlationData);
    }

    public void sendDirectMsg(String msg) {
        rabbitTemplate.convertAndSend(ExchangeEnum.DIRECT_EXCHANGE.getExchangeName() QueueEnum.DIRECT_QUEUE.getRoutingKey(), msg, new CorrelationData("2"));
        if (rabbitTemplate.waitForConfirms(1000)) {
            System.out.println("消息发送失败");
        }
    }

    public void sendFanoutMsg(String msg) {
        rabbitTemplate.convertAndSend(ExchangeEnum.FANOUT_EXCHANGE.getExchangeName(), "", msg, new CorrelationData("123213"));
    }

    public void sendTopicMsg(String msg) {
        rabbitTemplate.convertAndSend(ExchangeEnum.TOPIC_EXCHANGE.getExchangeName(), QueueEnum.TOPIC_QUEUE.getRoutingKey(), msg);
    }

    /**
     * 消息回调确认方法
     * @param correlationData   请求数据对象
     * @param ack               是否发送成功
     * @param cause             失败原因
     */
    @Override
```

```
    public void confirm(CorrelationData correlationData, boolean ack, String cause) {
        System.out.println("回调id: " + correlationData.getId());
        if (ack) {
            System.out.println("消息发送成功");
        } else {
            System.out.println("消息发送失败:  " + cause);
        }
    }

    /**
     * 当消息从交换机到队列失败时，该方法被调用。若成功，则不调用。注意：当方法调用后，confi
m方法也会被调用，且ack为true
     * @param message       发送的消息
     * @param replyCode     返回码
     * @param replyText     返回消息
     * @param exchange      交换机
     * @param routingKey    路由键
     */
    @Override
    public void returnedMessage(Message message, int replyCode, String replyText, String exc
ange, String routingKey) {
        System.out.println("message: " + message + "; replyCode: " + replyCode + "; exchange: "
+ exchange + "; routingKey: " + routingKey);
    }
}
```

RabbitConfiguration.java

```
package com.wzx.demo.conf;

import org.springframework.amqp.core.*;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.HashMap;
import java.util.Map;

/**
 * 用户配置
 *
 * @author wangzhengxing
 * @version 1.0
 * @Date 2019-10-31 09:07
 */
@Configuration
public class RabbitConfiguration {

    //=============================================Direct模式=======
==============================
    /**
     * 配置路由交换对象实例
     * @return
     */
```

```
    @Bean
    public DirectExchange directExchange() {
        return new DirectExchange((ExchangeEnum.DIRECT_EXCHANGE.getExchangeName()));
    }

    /**
     * 配置注册队列实例并设置持久化队列
     * @return
     */
    @Bean
    public Queue directQueue() {
        return new Queue(QueueEnum.DIRECT_QUEUE.getQueueName(), true);
    }

    /**
     * 注册队列绑定到路由交换配置上并设置指定路由键进行转发
     * @return
     */
    @Bean
    public Binding directBinding() {
        return BindingBuilder.bind(directQueue()).to(directExchange()).with(QueueEnum.DIRECT
QUEUE.getRoutingKey());
    }

    //=================================================死信队列=====
=========================================
    /**
     * 所谓死信：即（1）消息被拒绝（basic.reject 或者 basic.nack），并且requeue=false;（2）消
的过期时间到期了;
     *     * （3）队列长度限制超过了 等三个因素造成。
     *     * 我们会将以上原因造成的队列存入死信队列，死信队列其实也是一个普通的队列，我们可
根据自身需要，可以对死信进行操作。
     *     * 以下为死信队列的演示（将正常队列监听关闭并设置超时）：首先声明一个正常的队列，
设置死信队列的相关声明【死信交换器（与正常队列一致即可），死信路由Key等】
     *     * 设置完后，准备一个新的队列，此队列用于接收上一个正常队列发生死信后，将由此队列
替（即候补队列），然后将新队列通过上一个交换器以及正常队列中声明的死信路由Key进行绑定
     *     * 该操作与正常声明一致（声明交换器（可使用正常队列的交换器，无需另外声明），队列
将队列绑定到交换器）
     */
    /**
     * 直连死信交换器，正常与死信交换器一致
     * @return
     */
    @Bean
    public DirectExchange directDeadExchange() {
        return new DirectExchange(ExchangeEnum.DIRECT_DEAD_EXCHANGE.getExchangeName
), true, false);
    }

    /**
     * 声明一个正常的队列，并设置死信相关信息（交换器，路由Key），确保发生死信后会将死信存
交换器
     * @return
     */
```

```
    @Bean
    public Queue directDeadQueue() {
        Map<String, Object> args = new HashMap<>(4);
        // x-dead-letter-exchange 声明 死信交换机
        args.put("x-dead-letter-exchange", ExchangeEnum.DIRECT_DEAD_EXCHANGE.getExchan
eName());
        // x-dead-letter-routing-key 声明死信路由键
        args.put("x-dead-letter-routing-key", QueueEnum.DEAD_QUEUE.getRoutingKey());
        return new Queue(QueueEnum.DIRECT_DEAD_QUEUE.getQueueName(), true, false, false,
args);
    }

    /**
     * 将队列绑定到指定交换器并设置路由
     */
    @Bean
    public Binding directDeadBinding() {
        return BindingBuilder.bind(directDeadQueue()).to(directDeadExchange()).with(QueueEn
m.DIRECT_DEAD_QUEUE.getRoutingKey());
    }

    /**
     * 死信队列（候补队列） 若上面的正常队列发生死信时，需将发生死信的队列信息路由到此队列中
     * 路由过程：正常队列发送->信息到交换器->交换器路由到正常队列->监听,发生死信->死信回到
定的交换器->再由交换器路由到死信队列->死信监听
     */
    @Bean
    public Queue deadQueue() {
        return new Queue(QueueEnum.DEAD_QUEUE.getQueueName(), true, false, false);
    }

    /**
     * 绑定死信的队列到候补队列
     */
    @Bean
    public Binding deadBinding() {
        return BindingBuilder.bind(deadQueue()).to(directDeadExchange()).with(QueueEnum.DE
D_QUEUE.getRoutingKey());
    }


    //=================================================Fano
t模式=====================================
    /**
     * 广播路由
     * @return
     */
    @Bean
    public FanoutExchange fanoutExchange() {
        return new FanoutExchange(ExchangeEnum.FANOUT_EXCHANGE.getExchangeName());
    }

    /**
     * 广播队列
```

```
     * @return
     */
    @Bean
    public Queue fanoutQueue1() {
        return new Queue(QueueEnum.FANOUT_QUEUE.getQueueName(), true);
    }

    /**
     * 绑定广播队列到广播路由
     * @return
     */
    @Bean
    public Binding fanoutBinding() {
        return BindingBuilder.bind(fanoutQueue1()).to(fanoutExchange());
    }

    //=================================================Topic模式
================================
    /**
     * 主题路由
     * @return
     */
    @Bean
    public TopicExchange topicExchange() {
        return new TopicExchange(ExchangeEnum.TOPIC_EXCHANGE.getExchangeName());
    }

    /**
     * 主题队列
     * @return
     */
    @Bean
    public Queue topicQueue() {
        return new Queue(QueueEnum.TOPIC_QUEUE.getQueueName(), true);
    }

    /**
     * 绑定主题队列到路由
     * @return
     */
    @Bean
    public Binding topicBinding() {
        return BindingBuilder.bind(topicQueue()).to(topicExchange()).with(QueueEnum.TOPIC_Q
EUE.getRoutingKey());
    }
}
```

测试类SpringbootMqApplicationTests.java

```
package com.wzx.demo;

import com.wzx.demo.conf.QueueMessageService;
import org.junit.Before;
```

```
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.setup.MockMvcBuilders;
import org.springframework.web.context.WebApplicationContext;

import javax.annotation.Resource;

@RunWith(SpringRunner.class)
@SpringBootTest(classes = SpringbootMqApplication.class)
public class SpringbootMqApplicationTests {

    /**
     * 模拟mvc测试对象
     */
    private MockMvc mockMvc;

    /**
     * web项目上下文
     */
    @Autowired
    private WebApplicationContext webApplicationContext;

    @Resource
    private QueueMessageService queueMessageService;

    /**
     * 所有测试方法执行之前执行该方法
     */
    @Before
    public void before() {
        //获取mockmvc对象实例
        mockMvc = MockMvcBuilders.webAppContextSetup(webApplicationContext).build();
    }

    /**
     * 测试添加用户
     * @throws Exception
     */
    @Test
    public void testDeadMsg() throws Exception
    {
        //发送直连
        queueMessageService.sendDeadMsg("发来一则直连消息，测试死信队列");
    }

    @Test
    public void testDirectMsg() {
        //发送直连消息
        queueMessageService.sendDirectMsg("发来一则直连消息");
    }
```

```
    @Test
    public void testFanoutMsg() {
        //发送广播消息
        queueMessageService.sendFanoutMsg("发来一则广播消息");
    }

    @Test
    public void testTopicMsg() {
        queueMessageService.sendTopicMsg("发来一则主题消息");
    }

}
```