RabbitMQ死信队列应用

1、什么是死信队列

DLX, Dead-Letter-Exchange。利用DLX, 当消息在一个队列中因为业务队列处理失败(比如抛异常并且达到了retry的上限)变成死信（dead message）之后，它能被重新publish到另一个Exchange，这个Exchange就是DLX。消息变成死信一般有一下几种情况：

```
1）消息被拒绝（basic.reject/ basic.nack）并且requeue=false；
2）消费消息时程序出现了异常；
3）消息过期(x-message-ttl);
4）队列中有消息数量超过了最大值(x-max-length);
5）队列中的消息容量超过了队列的最大空间(x-max-length-bytes);
```

PS：消息被拒绝：

consumer端，当消费者要过滤某些消息的时候，那部分被过滤掉的消息如果不设置退回，消息重回队列的话，这些消息就变成了死信，即在下面的代码中第三个参数设置成false即可

```
channel.basicNack(envelope.getDeliveryTag(),false,false);
```

DLX也是一个正常的Exchange，和一般的Exchange没有区别，它能在任何的队列上被指定，实际上就是设置某个队列的属性，当这个队列中有死信时，RabbitMQ就会自动的将这个消息重新发布到设置的Exchange上去，进而被路由到另一个队列，可以监听这个队列中消息做相应的处理，这个特性可以弥补RabbitMQ 3.0以前支持的immediate参数（可以参考RabbitMQ之mandatory和immediate）的功能。

注意应用场景：重要的业务队列如果失败，就需要重新将消息用另一种业务逻辑处理；如果是正常的业务逻辑故意让消息中不合法的值失败，就不需要死信；具体场景具体分析

2、死信队列应用场景

最经典的应用场景：超时未支付订单处理

需求分析：

超过60分钟未支付的订单，我们需要进行超时订单的处理：先调用微信支付api，查询该订单的支付状态。如果未支付调用关闭　订单的api，并修改订单状态为已关闭，并回滚库存数。如果该订单已经支付，做补偿操作（修改订单状态和记录）。

实现思路：

如何获取超过60分钟的订单？我们目前有两种实现方案

（1）定时任务轮询方案

编写定时任务，查询所有60分钟前创建的订单列表。 循环此订单列表，查询每个订单的支付状态。如果已支付进行状态补偿，如果未支付则 关闭订单。

这种实现方案缺点是时间精度不高，对系统压力比较大。 （为什么时间精度不高？因为时间短交叉了）

（2）使用延迟消息队列

所谓延迟消息队列，就是消息的生产者发送的消息并不会立刻被消费，而是在设定的时间之后才可以消费。

我们可以在订单创建时发送一个延迟消息，消息为订单号，系统会在60分钟后取出这个消息，然后查询订单的支付状态，根据结果做出相应的处理。

3、延迟消息概念

3.1 消息的TTL（Time To Live）

消息的TTL就是消息的存活时间。RabbitMQ可以对队列和消息分别设置TTL。对队列设置就是队列每个消息的保留时间，也可以对每一个单独的消息做单独的设置。如果同时使用这2种方法，那么以过期时间小的那个数值为准！超过 了这个时间，我们认为这个消息就死了，称之为死信。 我们创建一个队列queue.temp，在Arguments 中添加x-message-ttl 为5000 （单位是 毫秒），那所在这个队列的消息在5秒后会消失。

3.2 死信交换器

死信交换器 **Dead Letter Exchanges**

一个消息在满足如下条件下，会进死信路由，记住这里是路由而不是队列，一个路由可以对应很多队列。

（1） 一个消息被Consumer拒收了，并且reject方法的参数里requeue是false。也就是说不会被再次放在队列里，被其他消费者使用。

（2）上面的消息的**TTL**到了，消息过期了。

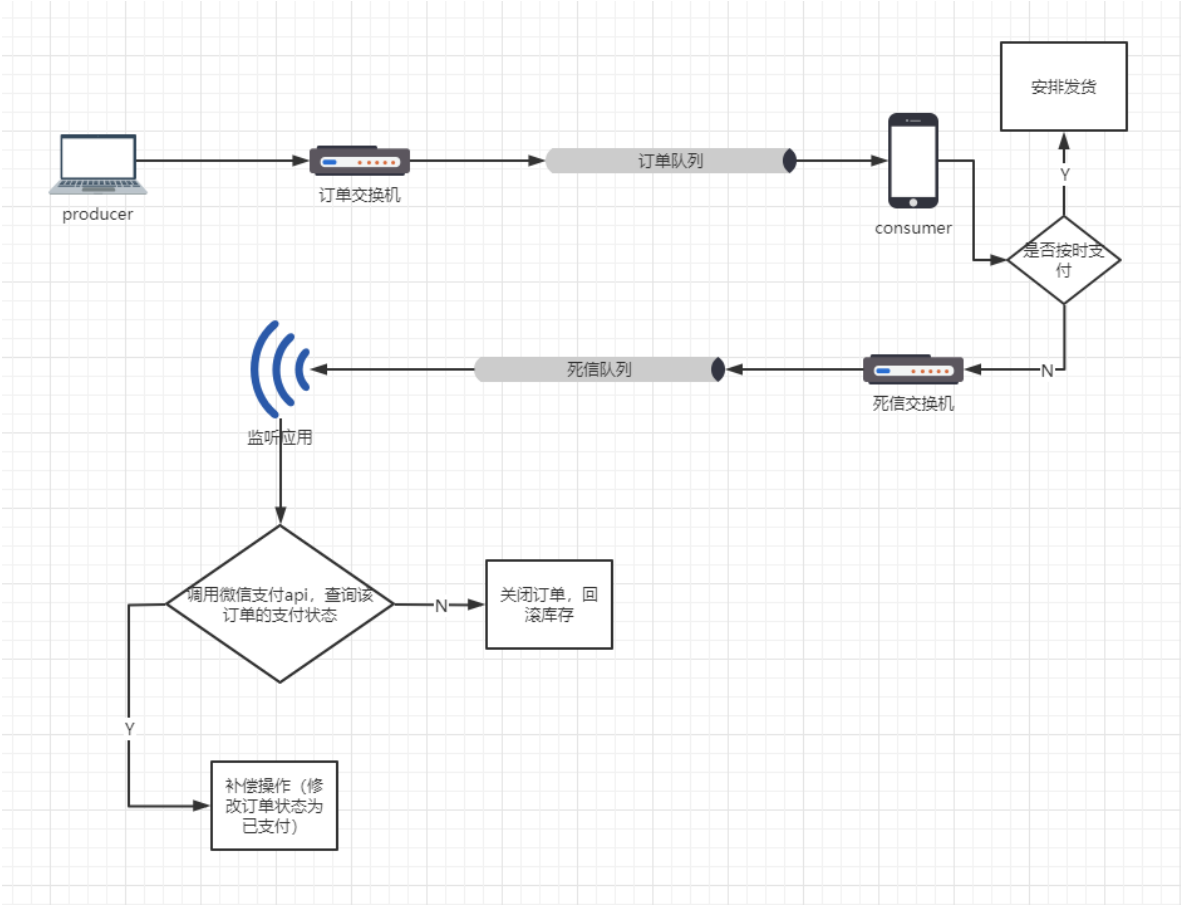
（3）队列的长度限制满了。排在前面的消息会被丢弃或者扔到死信路由上。

Dead Letter Exchange其实就是一种普通的exchange，和创建其他exchange没有两 样。只是在某一个设置Dead Letter Exchange的队列中有消息过期了，会自动触发消息的转发，发送到Dead Letter Exchange中去。

3.3 延时队列

在rabbitmq中不存在延时队列，但是我们可以通过设置消息的过期时间和死信队列来模拟出延时队列。消费者监听死信交换器绑定的队列，而不要监听消息发送的队列。

4、死信队列使用流程图



我们以订单支付作为例子来画图理解：服务器产生订单，通过订单交换机和routing key将订单消息发送到死信队列

5、死信队列应用（管理控制台）

需要创建四个角色：

order_exchange：订单交换机

order_queue：订单队列

dlx.exchange：死信交换机

dlx.queue：死信队列（存放无法消费的消息）

绑定关系：

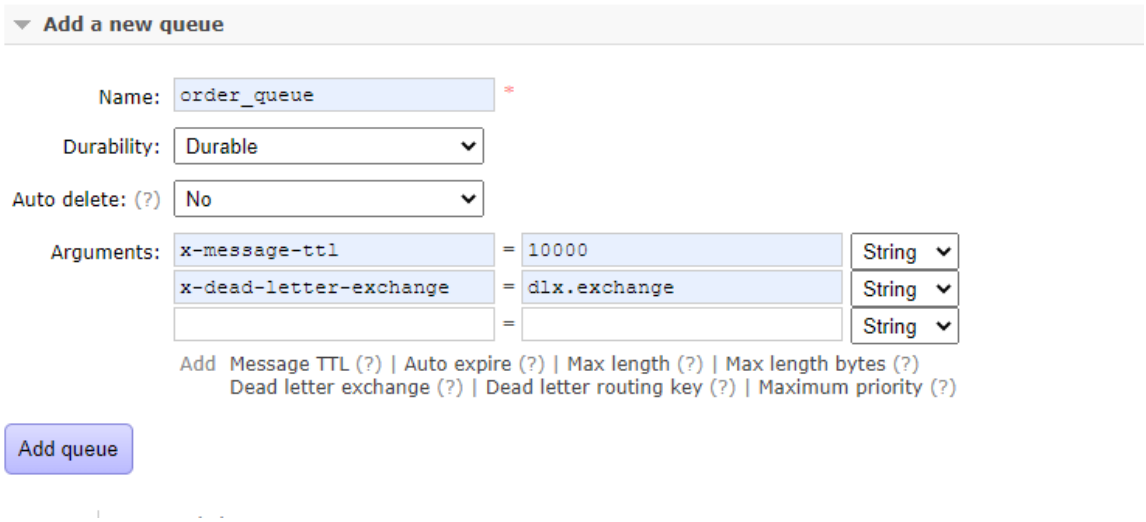
order_exchange和order_queue通过order.#路由key绑定；

dlx.exchange和dlx.queue通过#路由key绑定

唯一的区别在于order_queue的创建：

设置消息过期参数 x-message-ttl：10000（10秒）；

设置死信交换机 x-dead-letter-exchange：dlx.exchange



测试：通过order_exchange交换机发送一个消息，等待10S，发现一开始消息进入了order_queue；

10S后消息进入死信队列dlx.queue

6、死信队列应用（Java代码）

下面我们以两种方式模拟出死信队列（消息被拒绝）

6.1 RabbitMQ原生API

6.1.1 生产者Producer

```
package com.ydt.rabbitmq.dlx;

import com.rabbitmq.client.AMQP;
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.ydt.rabbitmq.util.ConnectionUtil;

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

/**
 * 生产者
 * 死信队列使用
 */
public class Producer {

    public static void main(String[] args) throws Exception{

        //得到连接
        Connection connection= ConnectionUtil.getConnection();
        //创建通道
        Channel channel=connection.createChannel();
        String exchangeName = "order_exchange";
        String routingKey = "order.save";

        //通过在properties设置来标识消息的相关属性
        for(int i=0;i<5;i++){
            Map<String, Object> headers = new HashMap<String, Object>();
            headers.put("num",i);
```

```
            AMQP.BasicProperties properties = new
AMQP.BasicProperties().builder()
                    .deliveryMode(2)                    // 传送方式 2:持久化投递
                    .contentEncoding("UTF-8")           // 编码方式
                    //.expiration("10000")              // 过期时间
                    .headers(headers)                   //自定义属性
                    .build();
            String message = "hello this is ack message ....."  + i;
            System.out.println(message);

 channel.basicPublish(exchangeName,routingKey,true,properties,message.getBytes()
);
        }

    }
}
```

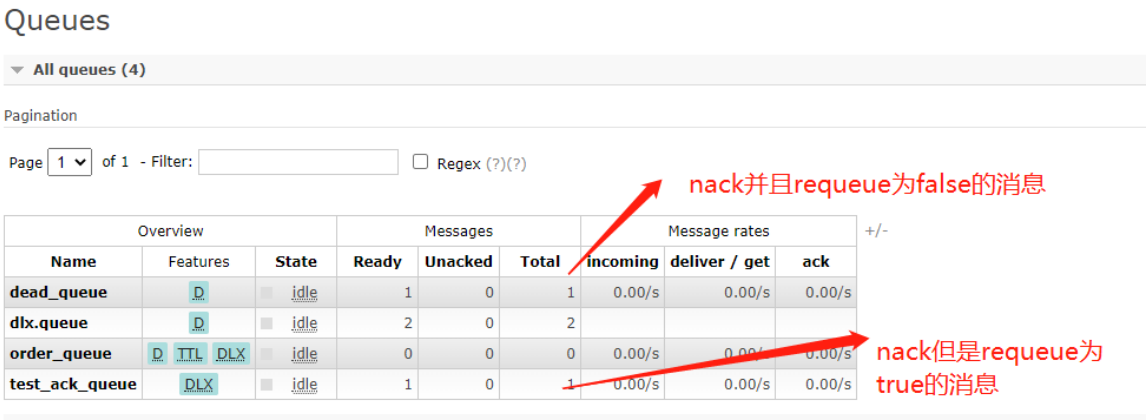
6.1.2 消费者Consumer

```
package com.ydt.rabbitmq.dlx;

import com.rabbitmq.client.*;
import com.ydt.rabbitmq.util.ConnectionUtil;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

public class Consumer {

    public static void main(String[] args) throws Exception{

        //得到连接
        Connection connection= ConnectionUtil.getConnection();
        //创建通道
        final Channel channel=connection.createChannel();
        //业务交换机和业务队列
        String exchangeName = "order_exchange";
        String exchangeType="topic";
        final String queueName = "order_queue";
        String routingKey = "order.#";

        //死信队列配置  ----------------
        String deadExchangeName = "dlx.exchange";
        String deadQueueName = "dlx.queue";
        String deadRoutingKey = "#";
        //死信队列配置  ----------------

        //如果需要将死信消息路由
        Map<String,Object> arguments = new HashMap<String, Object>();
        arguments.put("x-dead-letter-exchange",deadExchangeName);

        //创建业务交换机

 channel.exchangeDeclare(exchangeName,exchangeType,true,false,false,null);
        //创建业务队列
        channel.queueDeclare(queueName,false,false,false,arguments);
```

```
        //通过routing key绑定队列和交换机
        channel.queueBind(queueName,exchangeName,routingKey);

        //定义死信交换机

 channel.exchangeDeclare(deadExchangeName,exchangeType,true,false,false,null);
        //定义死信队列
        channel.queueDeclare(deadQueueName,true,false,false,null);
        //绑定交换机和队列
        channel.queueBind(deadQueueName,deadExchangeName,deadRoutingKey);

        System.out.println("consumer启动 .....");

        com.rabbitmq.client.Consumer consumer = new DefaultConsumer(channel){
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
AMQP.BasicProperties properties, byte[] body) throws IOException {
                try{
                    Thread.sleep(2000);
                }catch (Exception e){

                }
                Integer num = (Integer)properties.getHeaders().get("num");
                //通过消费者回调，根据判断进行消息确认或者拒绝
                if(num==0){
                    //未被ack的消息，并且requeue=false。即nack的 消息不再被退回队列而成
为死信队列
                    channel.basicNack(envelope.getDeliveryTag(),false,false);
                    String message = new String(body, "UTF-8");
                    System.out.println("consumer端的Nack消息是： " + message);
                }else {
                    channel.basicAck(envelope.getDeliveryTag(),false);
                    String message = new String(body, "UTF-8");
                    System.out.println("consumer端的ack消息是： " + message);
                }
            }
        };
        //消息要能重回队列，需要设置autoAck的属性为false，即在回调函数中进行手动签收
        channel.basicConsume(queueName,false,consumer);
    }
}
```

先启动消费者创建队列和交换机，然后通过生产者发送消息，可以看到，未被ACK并且并且requeue=false的消息,消息不再被退回队列而

成为了死信队列消息!



6.2 SpringBoot集成RabbitMQ

我们该章节通过x-dead-letter-routing-key参数来实现一个需求：消费者通过重试消费消息队列三次，如果都没有成功，则跳转到重定向队列消费者处理

6.2.1 pom依赖坐标

```
<dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <scope>test</scope>
        </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>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-amqp</artifactId>
        </dependency>

        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
        </dependency>
    </dependencies>
```

6.2.2 RabbitMQ连接信息配置

```
spring.rabbitmq.host=192.168.223.128
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest

# 允许消息消费失败的重试
spring.rabbitmq.listener.simple.retry.enabled=true
# 消息最多消费次数3次
spring.rabbitmq.listener.simple.retry.max-attempts=3
# 消息多次消费的间隔1秒
spring.rabbitmq.listener.simple.retry.initial-interval=1000
#  设置为false，会丢弃消息或者重新发布到死信队列
spring.rabbitmq.listener.simple.default-requeue-rejected=false
```

6.2.3 业务队列配置类

```
package com.ydt.springboot.rabbitmq.deadletter.config;

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;

/**
 * 交换机队列的配置
 */
@Configuration
public class RabbitDeadLetterConfig {

    public static final String MESSAGE_EXCHANGE = "TDL_EXCHANGE";//消息接收的交换机
    public static final String MESSAGE_ROUTING_KEY = "TDL_KEY";//消息routing key
    public static final String MESSAGE_QUEUE = "TDL_QUEUE"; //消息存储queue

    public static final String REDIRECT_EXCHANGE = "TREDIRECT_EXCHANGE";//重定向
交换机
    public static final String REDIRECT_ROUTING_KEY = "TTREDIRECT_KEY_R"; //重定
向队列routing key
    public static final String REDIRECT_QUEUE = "TREDIRECT_QUEUE"; //重定向消息存
储queue

    /**
     * 定义消息接收交换机
     * 交换机类型没有关系 不一定为directExchange  不影响该类型交换机的特性.
     */
    @Bean("messageExchange")
    public Exchange messageExchange() {
        return
ExchangeBuilder.directExchange(MESSAGE_EXCHANGE).durable(true).build();
    }

    /**
     * 定义消息存储queue，设置死信交换机和死信路由key,本例子当异常时将消息
```

```
     * 发送到x-dead-letter-exchange，然后x-dead-letter-exchange根据x-dead-letter-routing-key将消息路由到REDIRECT_QUEUE
     * @return
     */
    @Bean("messageQueue")
    public Queue messageQueue() {
        Map<String, Object> args = new HashMap<>(2);
//       x-dead-letter-exchange    声明  死信队列Exchange
        args.put("x-dead-letter-exchange", REDIRECT_EXCHANGE);
//       x-dead-letter-routing-key    声明  队列抛出异常并重试无效后进入重定向队列的routingKey(TKEY_R)
        args.put("x-dead-letter-routing-key", REDIRECT_ROUTING_KEY);
        return QueueBuilder.durable(MESSAGE_QUEUE).withArguments(args).build();
    }

    /**
     * 定义重定向交换机
     * @return
     */
    @Bean("redirectExchange")
    public Exchange redirectExchange() {
        return ExchangeBuilder.directExchange(REDIRECT_EXCHANGE).durable(true).build();
    }

    /**
     * 定义重定向消息存储queue
     * @return
     */
    @Bean("redirectQueue")
    public Queue redirectQueue() {
        return QueueBuilder.durable(REDIRECT_QUEUE).build();
    }

    /**
     * 消息队列绑定到消息交换器上.,第二个参数需要指定类型为队列来区分，因为交换机可以绑定交换机
     *
     * @return the binding
     */
    @Bean
    public Binding messageBinding() {
        return new Binding(MESSAGE_QUEUE, Binding.DestinationType.QUEUE,MESSAGE_EXCHANGE
                , MESSAGE_ROUTING_KEY, null);

    }

    /**
     * 将重定向队列通过重定向routingKey(TKEY_R)绑定到消息交换机上
     * @return the binding
     */
    @Bean
    public Binding redirectBinding() {
        return new Binding(REDIRECT_QUEUE, Binding.DestinationType.QUEUE, REDIRECT_EXCHANGE
                , REDIRECT_ROUTING_KEY, null);
    }
}
```

6.2.4业务队列生产者

```
package com.ydt.springboot.rabbitmq.deadletter.producer;

import com.ydt.springboot.rabbitmq.deadletter.config.RabbitDeadLetterConfig;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class DeadLetterProducer {

    @Autowired
    private RabbitTemplate rabbitTemplate;

    public void send(int number) {
        System.out.println("DeadLetterSender : "+ number);
        // 发送消息并制定routing key
        rabbitTemplate.convertAndSend(
                RabbitDeadLetterConfig.MESSAGE_EXCHANGE,
                RabbitDeadLetterConfig.MESSAGE_ROUTING_KEY,
                number);
    }
}
```

6.2.5 业务队列消费者

```
package com.ydt.springboot.rabbitmq.deadletter.consumer;

import com.ydt.springboot.rabbitmq.deadletter.config.RabbitDeadLetterConfig;
import org.springframework.amqp.rabbit.annotation.RabbitHandler;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;

/**
 * 消息异常类（人为监听制造异常）
 */
@Component
@RabbitListener(queues = RabbitDeadLetterConfig.MESSAGE_QUEUE)
public class DeadLetterConsumer {

    @RabbitHandler
    public void testDeadLetterQueueAndThrowsException(Integer number){
        System.out.println("DeadLetterConsumer :" + number +"/0 ");
        int i = number / 0;
    }
}
```

6.2.6 重定向队列消费者

队列"死信"后，会将消息投递到Dead Letter Exchanges，然后该Exchange会将消息投递到重定向队列。此时，在重定向队列中，做对应的业务操作

```
package com.ydt.springboot.rabbitmq.deadletter.consumer;
```

```java
import com.ydt.springboot.rabbitmq.deadletter.config.RabbitDeadLetterConfig;
import org.springframework.amqp.rabbit.annotation.RabbitHandler;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.stereotype.Component;

/**
 * 重定向队列监听类
 */
@RabbitListener(queues = RabbitDeadLetterConfig.REDIRECT_QUEUE)
@Component
public class RedirectQueueConsumer {

    /**
     * 重定向队列和死信队列形参一致Integer number
     * @param number
     */
    @RabbitHandler
    public void fromDeadLetter(Integer number){
        System.out.println("RedirectQueueConsumer : " + number);
        // 对应的业务操作。。。。。
    }
}
```

6.2.7 测试

先启动项目，保证没有异常并且加载了rabbitmq信息

```
] c.y.s.r.d.DeadletterApplication          : Starting DeadletterApplication on DESKTOP-S4SQ9G3 with PID 29436 (D:\works
] c.y.s.r.d.DeadletterApplication          : No active profile set, falling back to default profiles: default
] o.s.a.r.c.CachingConnectionFactory       : Attempting to connect to: [192.168.223.128:5672]
] o.s.a.r.c.CachingConnectionFactory       : Created new connection: rabbitConnectionFactory#732f29af:0/SimpleConnectio
] c.y.s.r.d.DeadletterApplication          : Started DeadletterApplication in 1.866 seconds (JVM running for 3.041)
```

利用测试类发送一条消息

```java
package com.ydt.springboot.rabbitmq.deadletter;

import com.ydt.springboot.rabbitmq.deadletter.producer.DeadLetterProducer;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
class DeadletterApplicationTests {

    @Autowired
    private DeadLetterProducer deadLetterProducer;

    @Test
    public void test1() {
        deadLetterProducer.send(15);
    }

}
```

重试三次

```
2020-07-21 14:45:45.928  WARN 29436 --- [ntContainer#0-1] c.y.s.r.d.consumer.DeadLetterConsumer    : DeadLetterConsumer :15/0
2020-07-21 14:45:46.928  WARN 29436 --- [ntContainer#0-1] c.y.s.r.d.consumer.DeadLetterConsumer    : DeadLetterConsumer :15/0
2020-07-21 14:45:47.928  WARN 29436 --- [ntContainer#0-1] c.y.s.r.d.consumer.DeadLetterConsumer    : DeadLetterConsumer :15/0
2020-07-21 14:45:47.929  WARN 29436 --- [ntContainer#0-1] o.s.a.r.r.RejectAndDontRequeueRecoverer  : Retries exhausted for message
```

最终到达重定向对列，开发人员可以针对这些死信消息进行业务处理

```
c.y.s.r.d.c.RedirectQueueConsumer          : RedirectQueueConsumer : 15
```