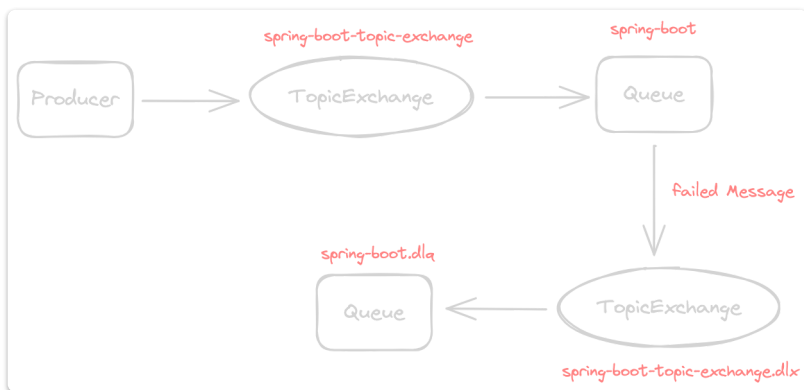


Dead Letter Queue는 **실패하거나 전달되지 못한 메시지를 보관**하는 queue이다.

## Dead Letter Exchange & Dead Letter Queue 생성



메시지가 전달에 실패하면, 실패한 메시지는 DLX(Dead Letter Exchange)로 라우팅된다.

```
public static final String topicExchangeName = "spring-boot-topic-exchange";
public static final String dlqExchangeName = topicExchangeName + ".dlx";

public static final String queueName = "spring-boot";
public static final String dlqQueueName = queueName + ".dlq";

/**
 * 큐 생성
 */
@Bean
Queue queue() {
    Map<String, Object> args = new HashMap<String, Object>();
    args.put("x-dead-letter-exchange", dlqExchangeName);
    return new Queue(queueName, true, false, false, args);
}

/**
 * DLQ 생성
 */
@Bean
Queue dlqQueue() {
    return new Queue(dlqQueueName);
}

/**
 * topic 타입으로 DLX 생성
 */
@Bean
TopicExchange dlqExchange() {
```

```

        return new TopicExchange(dlqExchangeName);
    }

```

위 코드에서는 DLX를 topic type의 exchange를 정의했다. 일반 큐는 consume에 실패한 메시지를 DLX로 보내야하기 때문에 `x-dead-letter-exchange` 속성 값에 정의한 exchange name을 입력했다.

topic type으로 DLX를 정의했기 때문에 DLQ로 라우팅하기 위한 routing-key와 함께 바운딩해주어야 한다.

```

@Bean
Binding dlqBinding(Queue dlqQueue, TopicExchange dlqExchange) {
    return
    BindingBuilder.bind(dlqQueue).to(dlqExchange).with("foo.bar.#");
}

```

## Consumer(Listener) 구현

```

@Slf4j
@Component
public class Consumer {

    @RabbitListener(queues = "#{queue.name}")
    public void consume(Channel channel, Message message,
    @Header(AmqpHeaders.DELIVERY_TAG) long tag) throws IOException {
        log.info("consume from normal queue");
        log.info("message: {}", message);
        channel.basicReject(tag, false);
    }

    @RabbitListener(queues = "#{dlqQueue.name}")
    public void dlqConsume(Channel channel, Message message,
    @Header(AmqpHeaders.DELIVERY_TAG) long tag) throws IOException {
        log.info("consume from dead letter queue");
        log.info("message: {}", message);
        channel.basicAck(tag, false);
    }
}

```

## Consume()

producer를 통해 메시지를 전송하면 `consume()` 메소드에서 메시지를 받아 추가 작업을 처리하게된다. 하지만 어떤 이유로 예외가 발생하거나 문제가 발생할 수 있다. 이럴 경우 `basicReject()` 메소드를 호출하여 메시지를 unacknowledge 처리해줄 수 있다. 처리된 메시지는 해당 큐에 지정된 `x-dead-letter-exchange` 로 전달된다.

## Note

`basicReject()` 가 아닌 `basicNack()` 메소드로도 가능하다.

## dlqConsume()

DLQ에 전달된 메시지들은 `dlqConsume()` 메소드에서 처리된다. 위 코드에서는 간단하게 `basicAck()` 를 호출하여 메시지를 처리해주었다.

## Publish

간단하게 String 형식의 메시지를 전송하는 코드를 구현한다.

```
@Slf4j
@Component
@RequiredArgsConstructor
public class Runner implements CommandLineRunner {


    private final RabbitTemplate rabbitTemplate;

    @Override
    public void run(String... args) {
        log.info("Sending message...");
        String message = "Hello from RabbitMQ!";
        rabbitTemplate.convertAndSend(RabbitMQConfig.topicExchangeName,
            "foo.bar.baz", message);
    }
}
```

## 결과

```
[          main] joo.example.messagewithrabbitmq.Runner    : Sending
message...
[ntContainer#0-1] j.example.messagewithrabbitmq.Consumer    : consume from
normal queue
[ntContainer#0-1] j.example.messagewithrabbitmq.Consumer    : message:
(Body:'"Hello from RabbitMQ!'" MessageProperties [headers=
{spring_listener_return_correlation=b5e46e09-8bf3-4792-b6b9-b737e6a2925e,
__TypeId__=java.lang.String}, contentType=application/json,
contentEncoding=UTF-8, contentLength=0, receivedDeliveryMode=PERSISTENT,
priority=0, redelivered=false, receivedExchange=spring-boot-topic-
exchange, receivedRoutingKey=foo.bar.baz, deliveryTag=1,
consumerTag=amq.ctag-UdqZiDZz3q9rz1kzqiGgkg, consumerQueue=spring-boot])
```

`channel.basicReject()` 를 통해 `reject` 된 메시지는 DLQ에 전달된다.

 RabbitMQ 3.13.3 Erlang 26.2.5

OverviewConnectionsChannelsExchangesQueues and StreamsAdmin

## Queues

► All queues (2)

Overview					Messages			Message rates			+/-
Virtual host	Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
/	spring-boot	classic	<span>D</span> <span>DLX</span>	<span>running</span>	0	0	0	0.00/s	0.00/s	0.00/s	
/	spring-boot.dlq	classic	<span>D</span>	<span>running</span>	1	0	1		0.00/s	0.00/s	

► Add a new queue

DLQ에 전달된 메시지는 `dlqConsume()` 메소드에의해서 처리된다.

```
[ntContainer#1-1] j.example.messagewithrabbitmq.Consumer : consume from
dead letter queue
[ntContainer#1-1] j.example.messagewithrabbitmq.Consumer : message:
(Body:"Hello from RabbitMQ!" MessageProperties [headers=
{spring_listener_return_correlation=b5e46e09-8bf3-4792-b6b9-b737e6a2925e,
x-first-death-exchange=spring-boot-topic-exchange, x-last-death-
reason=rejected, x-death=[{reason=rejected, count=1, exchange=spring-boot-
topic-exchange, time=Thu Jun 13 16:54:20 KST 2024, routing-keys=
[foo.bar.baz], queue=spring-boot}], x-first-death-reason=rejected, x-
first-death-queue=spring-boot, x-last-death-queue=spring-boot, x-last-
death-exchange=spring-boot-topic-exchange, __TypeId__=java.lang.String},
contentType=application/json, contentEncoding=UTF-8, contentLength=0,
receivedDeliveryMode=PERSISTENT, priority=0, redelivered=false,
receivedExchange=spring-boot-topic-exchange.dlx,
receivedRoutingKey=foo.bar.baz, deliveryTag=1, consumerTag=amq.ctag-
0Sf1NtAUieGF_mQQdk7u4A, consumerQueue=spring-boot.dlq])
```

### Note

큐의 arguments에 `x-dead-letter-routing-key` 를 추가해주면 DLX로 전달될 때 추가한 routing key를 사용하여 라우팅된다. 추가하지 않으면 실패한 큐에서 사용한 routing key가 그대로 사용된다.

## Retry

기본 retry policy와 다를 것 없는 방식으로 다음과 같이 실패한 메시지를 retry 해줄 수 있다. 이때 DLQ를 통해서 메시지를 처리해줄 때 `x-retries-count` 라는 이름의 헤더를 추가하여 retry 횟수를 저장한다. 그리고 retry할 때마다 이 횟수를 1씩 증가시킨다.

```

@RabbitListener(queues = "#{dlqQueue.name}")
public void dlqConsume(Channel channel, Message message,
@Header(AmqpHeaders.DELIVERY_TAG) long tag) throws IOException {
    log.info("consume from dead letter queue");
    log.info("message: {}", message);

    String HEADER_X_RETRIES_COUNT = "x-retries-count";
    Integer retriesCnt = (Integer)
message.getMessageProperties().getHeaders().get(HEADER_X_RETRIES_COUNT);

    if (retriesCnt == null) retriesCnt = 1;
    else retriesCnt++;

    // retry 횟수가 3회 넘어가면 취소
    if (retriesCnt > 3) {
        log.info("Discarding message");
        channel.basicAck(tag, false);
        return;
    }

    log.info("Retrying message for the {} time", retriesCnt);
    // 증가시킨 count 지정

    message.getMessageProperties().getHeaders().put(HEADER_X_RETRIES_COUNT,
retriesCnt);
    // 다시 실패한 큐에게 전달
    rabbitTemplate.send(RabbitMQConfig.topicExchangeName,
message.getMessageProperties().getReceivedRoutingKey(), message);
    channel.basicAck(tag, false);
}

```

지정한 횟수(위 예시에서는 3)만큼 retry 되었다면 다시 전달하지 않고 해당 메시지를 취소한다.

다른 방법으로 `x-message-ttl` 헤더를 추가하여 시간을 기준으로 만료된 메시지를 취소할 수 있다.

## Parking Lot Queue

하지만 은행처럼 거래에 대한 메시지를 취소시키는 것은 문제가 될 수 있다. 따라서 이러한 상황에 대해 `Parking Lot Queue` 개념이 있다.

실패한 메시지를 모두 DLQ로 보내지고, 지정된 횟수만큼 실패한 후 `Parking Lot Queue` 로 전달되어 추가로 진행된다.

## Exchange & Queue 생성

```

public static final String parkingLotExchangeName = topicExchangeName +
".parking-lot";
public static final String parkingLotQueueName = queueName + ".parking-

```

```

lot";

/**
 * Parking Lot Queue 생성
 */
@Bean
Queue parkingLotQueue() {
    return new Queue(parkingLotQueueName);
}

/**
 * ParkingLotExchange 생성
 */
@Bean
TopicExchange parkingLotExchange() {
    return new TopicExchange(parkingLotExchangeName);
}

@Bean
Binding parkingLotBinding(Queue parkingLotQueue, TopicExchange
parkingLotExchange) {
    return
BindingBuilder.bind(parkingLotQueue).to(parkingLotExchange).with("foo.bar.
#");
}

```

## Consumer

그리고 consume 할 때 취소하는 대신 `Parking Lot Queue` 로 전달해주면 된다.

```

@RabbitListener(queues = "${dlqQueue.name}")
public void dlqConsume(Channel channel, Message message,
@Header(AmqpHeaders.DELIVERY_TAG) long tag) throws IOException {
    log.info("consume from dead letter queue");
    log.info("message: {}", message);

    String HEADER_X_RETRIES_COUNT = "x-retries-count";
    Integer retriesCnt = (Integer)
message.getMessageProperties().getHeaders().get(HEADER_X_RETRIES_COUNT);

    if (retriesCnt == null) retriesCnt = 1;
    else retriesCnt++;

    // retry 횟수가 3회 넘어가면
    if (retriesCnt > 3) {
        log.info("Sending message to the parking lot queue");
        // parking lot queue 로 전달
        rabbitTemplate.send(RabbitMQConfig.parkingLotExchangeName,
message.getMessageProperties().getReceivedRoutingKey(), message);
    }
}

```

```

        channel.basicAck(tag, false);
        return;
    }

    log.info("Retrying message for the {} time", retriesCnt);
    // 증가시킨 count 지정

    message.getMessageProperties().getHeaders().put(HEADER_X_RETRIES_COUNT,
retriesCnt);
    // 다시 실패한 큐에게 전달
    rabbitTemplate.send(RabbitMQConfig.topicExchangeName,
message.getMessageProperties().getReceivedRoutingKey(), message);
    channel.basicAck(tag, false);
}

```

## Parking Lot Consume

parking lot queue에 전달된 메시지는 DB에 저장하거나 이메일로 알림을 보내는 등 추가적인 처리를 해 줄 수 있다.

```

@RabbitListener(queues = "#{parkingLotQueue.name}")
public void parkingLotConsume(Channel channel, Message message,
@Header(AmqpHeaders.DELIVERY_TAG) long tag) throws IOException {
    log.info("Received message in parking lot queue");
    // Save to DB or send a email notification.
    channel.basicAck(tag, false);
}

```

전체 코드는 [MessageWithRabbitMQ](#) 에서 확인할 수 있다.

[reference]

- <https://www.baeldung.com/spring-amqp-error-handling>
- <https://www.rabbitmq.com/docs/dlx>