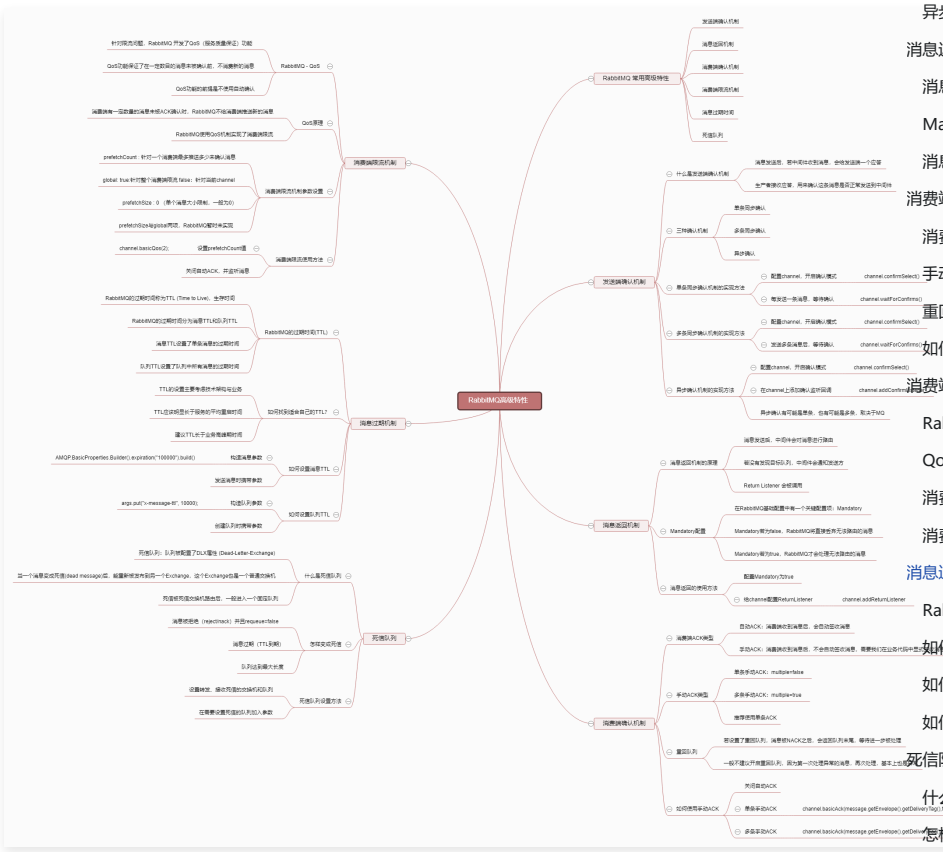
Moody • 更新于 2020-10-10

上一节 第三章RabbitM... 第五章RabbitM... 下一节



RabbitMQ 常用高级特性

- 发送端确认机制
- 消息返回机制
- 消费端确认机制
- 消费端限流机制
- 消息过期时间
- 死信队列

发送端确认机制

什么是发送端确认机制

- 消息发送后，若中间件收到消息，会给发送端一个应答
- 生产者接收应答，用来确认这条消息是否正常发送到中间件

三种确认机制

- 单条同步确认

- 异步确认

索引目录

单条同步确认机制的实现方法

- 配置channel，开启确认模式：channel.confirmSelect()

代码块

```
channel.confirmSelect();
```

- 每发送一条消息，调用channel.waitForConfirms()方法，等待确认

代码块

```
if(channel.waitForConfirms()){
    //确认成功逻辑
}else {
    //确认失败逻辑
}
```

多条同步确认机制的实现方法

- 配置channel，开启确认模式：channel.confirmSelect()

代码块

```
channel.confirmSelect();
```

- 发送多条消息后，调用channel.waitForConfirms()方法，等待确认

代码块

```
if(channel.waitForConfirms()){
    //确认成功逻辑
}else {
    //确认失败逻辑
}
```

异步确认机制的实现方法

- 配置channel，开启确认模式：channel.confirmSelect()

代码块

```
channel.confirmSelect();
```

- 在channel上添加监听：addConfirmListener，发送消息后，会回调此方法，通知是否发送成功

代码块

```
channel.addConfirmListener(new ConfirmListener() {
    public void handleAck(long deliveryTag, boolean multiple) throws IOException {
        log.info("Ack, deliveryTag: {}, multiple: {}",  deliveryTag, multiple);
        //确认成功逻辑，其中multiple指示单条/多条
    }
    public void handleNack(long deliveryTag, boolean multiple) throws IOException {
```

```
8           //确认失败逻辑，其中multiple指示单条/多条
9       }
10  });
```

- 异步确认有可能是单条，也有可能是多条，取决于MQ

消息返回机制

消息返回机制的原理

- 消息发送后，中间件会对消息进行路由
- 若没有发现目标队列，中间件会通知发送方
- Return Listener 会被调用

Mandatory配置

- 在RabbitMQ基础配置中有一个关键配置项：Mandatory
- Mandatory若为false，RabbitMQ将直接丢弃无法路由的消息
- Mandatory若为true，RabbitMQ才会处理无法路由的消息

消息返回的使用方法

- 配置Mandatory为true
- 给channel配置ReturnListener

代码块

```
1       channel.addReturnListener(new ReturnCallback() {
2           @Override
3           public void handle(Return returnMessage) {
4               log.info("Message Return: returnMessage:{}", returnMessage);
5               //消息返回后的业务逻辑
6           }
7       });
```

消费端确认机制

消费端ACK类型

- 自动ACK：消费端收到消息后，会自动签收消息
- 手动ACK：消费端收到消息后，不会自动签收消息，需要我们在业务代码中显式签收消息

手动ACK类型

- 单条手动ACK：multiple=false
- 多条手动ACK：multiple=true
- 推荐使用单条ACK

重回队列

- 一般不建议开启重回队列，因为第一次处理异常的消息，再次处理，基本上也是异常

如何使用手动ACK

- 关闭自动ACK

代码块

```
channel.basicConsume("queue.name", false, deliverCallback, consumerTag -> {
            });
```

- 单条手动ACK

代码块

```
channel.basicAck(message.getEnvelope().getDeliveryTag(),false);
```

- 多条手动ACK

代码块

```
channel.basicAck(message.getEnvelope().getDeliveryTag(),true);
```

消费端限流机制

RabbitMQ - QoS

- 针对限流问题，RabbitMQ 开发了QoS（服务质量保证）功能
- QoS功能保证了在一定数目的消息未被确认前，不消费新的消息
- QoS功能的前提是不使用自动确认

QoS原理

- QoS原理是当消费端有一定数量的消息未被ACK确认时，RabbitMQ不给消费端推送新的消息
- RabbitMQ使用QoS机制实现了消费端限流

消费端限流机制参数设置

- prefetchCount : 针对一个消费端最多推送多少未确认消息
- global: true:针对整个消费端限流 false：针对当前channel
- prefetchSize : 0 （单个消息大小限制，一般为0）
- prefetchSize与global两项，RabbitMQ暂时未实现

消费端限流使用方法

- 设置prefetchCount值

代码块

```
channel.basicQos(2);//参数为prefetchCount值
```

代码块

```
channel.basicConsume("queue.name", false, deliverCallback, consumerTag -> {
            });
```

消息过期机制

RabbitMQ的过期时间(TTL)

- RabbitMQ的过期时间称为TTL (Time to Live)，生存时间
- RabbitMQ的过期时间分为消息TTL和队列TTL
- 消息TTL设置了单条消息的过期时间
- 队列TTL设置了队列中所有消息的过期时间

如何找到适合自己的TTL?

- TTL的设置主要考虑技术架构与业务
- TTL应该明显长于服务的平均重启时间
- 建议TTL长于业务高峰期时间

如何设置消息TTL

- 构造消息参数

代码块

```
AMQP.BasicProperties properties = new AMQP.BasicProperties.Builder().expiration("10000").build();
```

- 发送消息时携带参数

代码块

```
channel.basicPublish("exchange.name", "key.name", properties, messageToSend.getBytes());
```

如何设置队列TTL

- 构造队列参数

代码块

```
Map<String, Object> args = new HashMap<String, Object>();
args.put("x-message-ttl", 10000);
```

- 创建队列时携带参数

代码块

```
channel.queueDeclare(
                    "queue.name",
                    true,
                    false,
                    false,
```

死信队列

什么是死信队列

- 死信队列：队列被配置了DLX属性 (Dead-Letter-Exchange)
- 当一个消息变成死信(dead message)后，能重新被发布到另一个Exchange，这个Exchange也是一个普通交换机
- 死信被死信交换机路由后，一般进入一个固定队列

怎样变成死信

- 消息被拒绝（reject/nack）并且requeue=false
- 消息过期（TTL到期）
- 队列达到最大长度

死信队列设置方法

- 设置转发、接收死信的交换机和队列：

代码块

```
channel.exchangeDeclare(
        "exchange.dlx",
        BuiltinExchangeType.TOPIC,
        true,
        false,
        null);

channel.queueDeclare(
        "queue.dlx",
        true,
        false,
        false,
        null);

channel.queueBind(
        "queue.dlx",
        "exchange.dlx",
        "#");
```

- 在需要设置死信的队列加入参数：

代码块

```
Map<String, Object> args = new HashMap<>(16);
args.put("x-dead-letter-exchange", "exchange.dlx");
args.put("x-max-length", 10);
```

第三章RabbitMQ基础总结 上一节 下一节 第五章RabbitMQ和SpringBoot适配