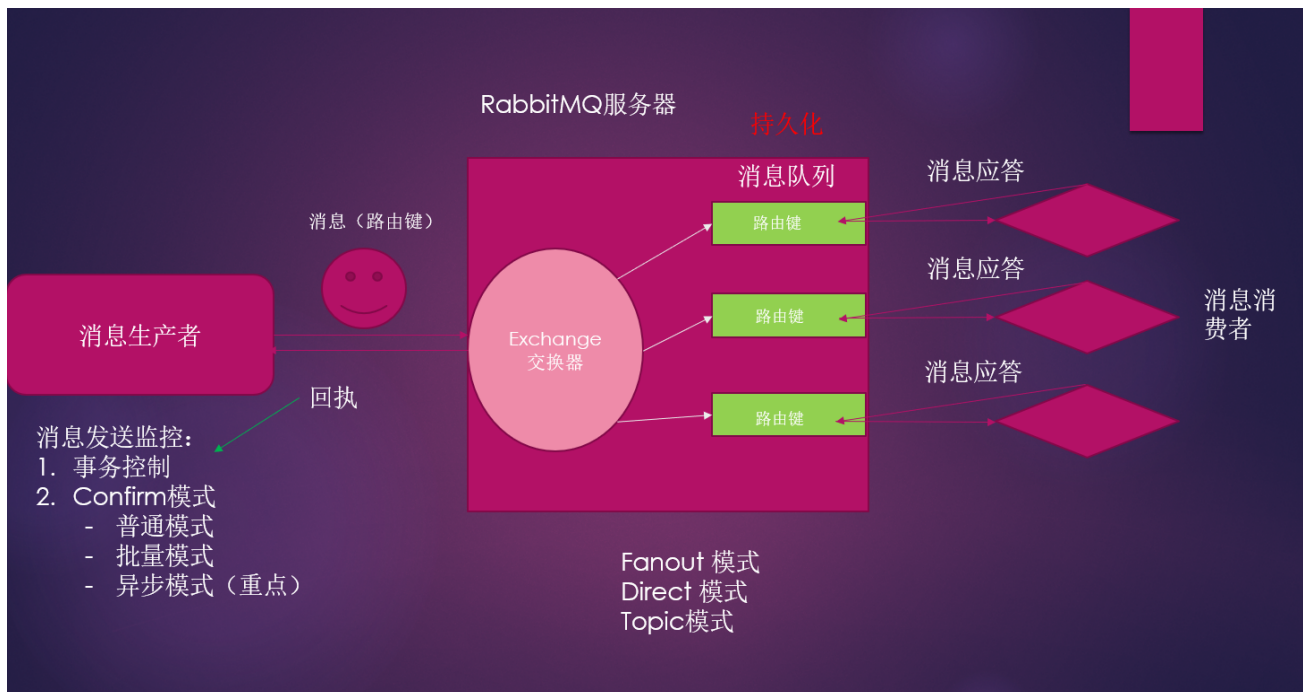


# RabbitMQ 学习指南



## 一. 代码部分

### 1. 新建一个maven 工程以及连接工具类

```
1 public class ConnectionUtil {
2
3     public static com.rabbitmq.client.Connection getConnection() throws IOException,
4         TimeoutException {
5
6         // 定义一个连接工厂
7         ConnectionFactory factory = new ConnectionFactory();
8
9         // 设置rabbitmq 服务器地址（我们使用docker 里的rabbitmq地址）
10        factory.setHost("120.78.138.11");
11
12        // 设置AMQP 的协议端口
13        factory.setPort(5672);
14
15        // 设置vhost
16        factory.setVirtualHost("/vhost_cris");
17
18        // 设置用户名和密码
19        factory.setUsername("cris");
20        factory.setPassword("123");
21    }
22 }
```

```
20
21     return factory.newConnection();
22 }
23 }
```

## 2. 简单队列模型

### 2.1 消息生产者模块

```
1  // 消息生产者
2  public class Send {
3
4      private static final String QUEUE_NAME = "test_queue";
5
6      public static void main(String[] args) throws IOException, TimeoutException {
7          // 获取一个连接
8          com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();
9
10         // 从连接中获取一个通道
11         Channel channel = connection.createChannel();
12
13         // 创建队列声明
14         channel.queueDeclare(QUEUE_NAME, false, false, false, null);
15
16         String msg = "hello,cris";
17
18         channel.basicPublish("", QUEUE_NAME, null, msg.getBytes());
19
20         System.out.println("send msg : " + msg);
21         channel.close();
22         connection.close();
23
24     }
25 }
```

### 2.2 消息消费者模块

```
1  // 消费者
2  public class Receive {
3
4      private static final String QUEUE_NAME = "test_queue";
5
6      public static void main(String[] args) throws IOException, TimeoutException,
7      ShutdownSignalException,
8      ConsumerCancelledException, InterruptedException {
9
10         // 获取一个连接
11         com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();
12
13         // 从连接中获取一个通道
```

```

13     Channel channel = connection.createChannel();
14
15     // 创建一个消费者
16     DefaultConsumer consumer = new DefaultConsumer(channel) {
17         @Override
18         public void handleDelivery(String consumerTag, Envelope envelope,
BasicProperties properties, byte[] body)
19             throws IOException {
20             System.out.println(new String(body) + "-----");
21         }
22     };
23
24     // 监听队列
25     channel.basicConsume(QueueName, true, consumer);
26
27
28 }
29
30 private static void oldApi() throws IOException, TimeoutException, InterruptedException
31 {
32     Connection connection = ConnectionUtil.getConnection();
33     Channel channel = connection.createChannel();
34
35     // 定义队列的消费者(老的API)
36     QueueingConsumer consumer = new QueueingConsumer(channel);
37
38     // 监听队列
39     channel.basicConsume(QueueName, true, consumer);
40     while (true) {
41         Delivery delivery = consumer.nextDelivery();
42         System.out.println(new String(delivery.getBody()));
43     }
44 }

```

### 3. work queue 工作队列（一个消息一个消费者消费）

#### 3.1 生产者

```

1 // 消息生产者
2 public class Send {
3
4     private static final String QueueName = "test_queue";
5
6     public static void main(String[] args) throws IOException, TimeoutException {
7         // 获取一个连接
8         com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();
9
10        // 从连接中获取一个通道
11        Channel channel = connection.createChannel();
12
13        // 创建队列声明
14        channel.queueDeclare(QueueName, false, false, false, null);

```

```

15
16     for (int i = 0; i < 50; i++) {
17
18         String msg = "hello,cris" +i;
19         System.out.println("send msg : " + msg);
20         channel.basicPublish("", QUEUE_NAME, null, msg.getBytes());
21     }
22     channel.close();
23     connection.close();
24
25 }
26 }

```

### 3.2 消费者1

```

1  // 消费者
2  public class Receive1 {
3
4      private static final String QUEUE_NAME = "test_queue";
5
6      public static void main(String[] args) throws IOException, TimeoutException,
ShutdownSignalException,
7          ConsumerCancelledException, InterruptedException {
8
9          // 获取一个连接
10         com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();
11
12         // 从连接中获取一个通道
13         Channel channel = connection.createChannel();
14
15         // 创建队列声明
16         channel.queueDeclare(QUEUE_NAME, false, false, false, null);
17
18         // 创建一个消费者
19         DefaultConsumer consumer = new DefaultConsumer(channel) {
20             // 消息到达触发该方法
21             @Override
22             public void handleDelivery(String consumerTag, Envelope envelope,
BasicProperties properties, byte[] body)
23                 throws IOException {
24                 System.out.println("Receive1 ----"+new String(body));
25                 try {
26                     Thread.sleep(1000);
27                 } catch (InterruptedException e) {
28
29                     e.printStackTrace();
30                 }finally {
31                     System.out.println("Receive1 has done");
32                 }
33             }
34         };
35

```

```

36         // 监听队列
37         channel.basicConsume(QueueName, true, consumer);
38     }
39 }

```

## 3.2 消费者2

```

1  // 消费者
2  public class Receive2 {
3
4      private static final String QueueName = "test_queue";
5
6      public static void main(String[] args) throws IOException, TimeoutException,
ShutdownSignalException,
7          ConsumerCancelledException, InterruptedException {
8
9          // 获取一个连接
10         com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();
11
12         // 从连接中获取一个通道
13         Channel channel = connection.createChannel();
14
15         // 创建队列声明
16         channel.queueDeclare(QueueName, false, false, false, null);
17
18         // 创建一个消费者
19         DefaultConsumer consumer = new DefaultConsumer(channel) {
20             // 消息到达触发该方法
21             @Override
22             public void handleDelivery(String consumerTag, Envelope envelope,
BasicProperties properties, byte[] body)
23                 throws IOException {
24                 System.out.println("Receive2 ----"+new String(body));
25                 try {
26                     Thread.sleep(2000);
27                 } catch (InterruptedException e) {
28
29                     e.printStackTrace();
30                 }finally {
31                     System.out.println("Receive2 has done");
32                 }
33             }
34         };
35
36         // 监听队列
37         channel.basicConsume(QueueName, true, consumer);
38     }
39 }

```

## 4. 公平分发（能者多劳）模式：fair dispatch，执行快的消费者可以消费更多的消息

### 4.1 生产者

```
1 // 消息生产者
2 public class Send {
3
4     private static final String QUEUE_NAME = "test_queue";
5
6     public static void main(String[] args) throws IOException, TimeoutException {
7         // 获取一个连接
8         com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();
9
10        // 从连接中获取一个通道
11        Channel channel = connection.createChannel();
12
13        // 创建队列声明（代码创建队列）
14        channel.queueDeclare(QUEUE_NAME, false, false, false, null);
15
16        /*
17         * 每个消费者发送确认消息到消息队列之前，消息队列不会发送下一个消息到消费者，即消费者和消息
18         * 队列形成了一应一答
19         * 限制消息队列发送给同一个消费者的消息不会超过一条（一次来回交流中），消费者一次只处理一次
20         * 消息
21         */
22        int prefetchCount = 1;
23        channel.basicQos(1);
24
25        for (int i = 0; i < 50; i++) {
26
27            String msg = "hello,cris" + i;
28            System.out.println("send msg : " + msg);
29            channel.basicPublish("", QUEUE_NAME, null, msg.getBytes());
30        }
31        channel.close();
32        connection.close();
33    }
34 }
```

### 4.2 消费者1

```
1 // 消费者
2 public class Receive1 {
3
4     private static final String QUEUE_NAME = "test_queue";
5
6     public static void main(String[] args) throws IOException, TimeoutException,
7         ShutdownSignalException,
8         ConsumerCancelledException, InterruptedException {
```

```

9      // 获取一个连接
10     com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();
11
12     // 从连接中获取一个通道
13     Channel channel = connection.createChannel();
14
15     // 创建队列声明
16     channel.queueDeclare(QUEUE_NAME, false, false, false, null);
17
18     channel.basicQos(1); // 保证一次只接受一个消息
19
20     // 创建一个消费者
21     DefaultConsumer consumer = new DefaultConsumer(channel) {
22         // 消息到达触发该方法
23         @Override
24         public void handleDelivery(String consumerTag, Envelope envelope,
BasicProperties properties, byte[] body)
25             throws IOException {
26             System.out.println("Receive1 ----" + new String(body));
27             try {
28                 Thread.sleep(500);
29             } catch (InterruptedException e) {
30
31                 e.printStackTrace();
32             } finally {
33                 System.out.println("Receive1 has done");
34                 // 手动回应消息队列已经处理好了
35                 channel.basicAck(envelope.getDeliveryTag(), false);
36             }
37         }
38     };
39
40     // 监听队列
41     boolean autoAck = false; // 关闭自动应答
42     channel.basicConsume(QUEUE_NAME, autoAck, consumer);
43
44 }
45 }

```

## 4.2 消费者2

```

1  // 消费者
2  public class Receive2 {
3
4      private static final String QUEUE_NAME = "test_queue";
5
6      public static void main(String[] args) throws IOException, TimeoutException,
ShutdownSignalException,
7          ConsumerCancelledException, InterruptedException {
8
9          // 获取一个连接
10         com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();

```

```

11
12 // 从连接中获取一个通道
13 Channel channel = connection.createChannel();
14
15 // 创建队列声明
16 channel.queueDeclare(QueueName, false, false, false, null);
17
18 channel.basicQos(1); // 保证一次只接受一个消息
19
20 // 创建一个消费者
21 DefaultConsumer consumer = new DefaultConsumer(channel) {
22     // 消息到达触发该方法
23     @Override
24     public void handleDelivery(String consumerTag, Envelope envelope,
BasicProperties properties, byte[] body)
25         throws IOException {
26         System.out.println("Receive2 ----" + new String(body));
27         try {
28             Thread.sleep(1000);
29         } catch (InterruptedException e) {
30
31             e.printStackTrace();
32         } finally {
33             System.out.println("Receive2 has done");
34             // 手动回应消息队列已经处理好了
35             channel.basicAck(envelope.getDeliveryTag(), false);
36         }
37     }
38 };
39
40 // 监听队列
41 boolean autoAck = false; // 关闭自动应答
42 channel.basicConsume(QueueName, autoAck, consumer);
43 }
44 }

```

## 5. 订阅/发布模式

### 5.1 fanout

消息发送到交换器，由交互器将消息发送到绑定的多个消息队列中，每个消息队列发送给绑定的消费者进行消费，注意：先启动消息发送端创建交换器，再启动不同的消费端)

- 生产者

```

1 //消息生产者
2 public class Send {
3
4     private static final String EXCHANGE_NAME = "test_queue_exchange";
5
6     public static void main(String[] args) throws IOException, TimeoutException {

```



```

7      // 获取一个连接
8      com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();
9
10     // 从连接中获取一个通道
11     Channel channel = connection.createChannel();
12
13     // 交换器声明
14     channel.exchangeDeclare(EXCHANGE_NAME, "fanout");
15
16     /*
17      * 每个消费者发送确认消息到消息队列之前，消息队列不会发送下一个消息到消费者，即消费者和消息
    队列形成了一应一答
18      * 限制消息队列发送给同一个消费者的消息不会超过一条（一次来回交流中），消费者一次只处理一次
    消息
19      */
20     int prefetchCount = 1;
21     channel.basicQos(1);
22
23     String msg = "hello,cris,i am exchange...";
24     channel.basicPublish(EXCHANGE_NAME, "", null, msg.getBytes());
25     System.out.println("发送信息成功" + msg);
26
27     channel.close();
28     connection.close();
29
30 }
31 }

```

- 两个消费者

```

1  // 消费者
2  public class Receive1 {
3
4      private static final String QUEUE_NAME = "test_queue_fanout";
5      private static final String EXCHANGE_NAME = "test_queue_exchange";
6
7      public static void main(String[] args) throws IOException, TimeoutException,
    ShutdownSignalException,
8          ConsumerCancelledException, InterruptedException {
9
10     // 获取一个连接
11     com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();
12
13     // 从连接中获取一个通道
14     Channel channel = connection.createChannel();
15
16     // 创建队列声明
17     channel.queueDeclare(QUEUE_NAME, false, false, false, null);
18
19     // 绑定消息队列到交换器
20     channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "");
21

```

```

22     channel.basicQos(1); // 保证一次只接受一个消息
23
24     // 创建一个消费者
25     DefaultConsumer consumer = new DefaultConsumer(channel) {
26         // 消息到达触发该方法
27         @Override
28         public void handleDelivery(String consumerTag, Envelope envelope,
BasicProperties properties, byte[] body)
29             throws IOException {
30             System.out.println("Receive1 ----" + new String(body));
31             try {
32                 Thread.sleep(500);
33             } catch (InterruptedException e) {
34
35                 e.printStackTrace();
36             } finally {
37                 System.out.println("Receive1 has done");
38                 // 手动回应消息队列已经处理好了
39                 channel.basicAck(envelope.getDeliveryTag(), false);
40             }
41         }
42     };
43
44     // 监听队列
45     boolean autoAck = false; // 关闭自动应答
46     channel.basicConsume(QUEUE_NAME, autoAck, consumer);
47
48 }
49 }
50
51 // 消费者
52 public class Receive2 {
53
54     // 每个消费者绑定不同的队列
55     private static final String QUEUE_NAME = "test_queue2_fanout";
56     // 每个队列绑定相同的交换器
57     private static final String EXCHANGE_NAME = "test_queue_exchange";
58
59     public static void main(String[] args) throws IOException, TimeoutException,
ShutdownSignalException,
60         ConsumerCancelledException, InterruptedException {
61
62         // 获取一个连接
63         com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();
64
65         // 从连接中获取一个通道
66         Channel channel = connection.createChannel();
67
68         // 创建队列声明
69         channel.queueDeclare(QUEUE_NAME, false, false, false, null);
70
71         // 绑定消息队列到交换器
72         channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "");

```

```

73
74     channel.basicQos(1); // 保证一次只接受一个消息
75
76     // 创建一个消费者
77     DefaultConsumer consumer = new DefaultConsumer(channel) {
78         // 消息到达触发该方法
79         @Override
80         public void handleDelivery(String consumerTag, Envelope envelope,
81 BasicProperties properties, byte[] body)
82             throws IOException {
83             System.out.println("Receive2 ----" + new String(body));
84             try {
85                 Thread.sleep(500);
86             } catch (InterruptedException e) {
87
88                 e.printStackTrace();
89             } finally {
90                 System.out.println("Receiv2 has done");
91                 // 手动回应消息队列已经处理好了
92                 channel.basicAck(envelope.getDeliveryTag(), false);
93             }
94         }
95     };
96
97     // 监听队列
98     boolean autoAck = false; // 关闭自动应答
99     channel.basicConsume(QUEUE_NAME, autoAck, consumer);
100
101 }

```

## 5.2 direct

根据消息的路由键，交换器将消息发送到消息队列中的路由键完全匹配的消息队列中, routingKey（消息和消息队列需要一致）

- 生产者

```

1 public class Send {
2
3     private static final String EXCHANGE_NAME = "test_exchange_direct";
4
5     public static void main(String[] args) throws IOException, TimeoutException {
6
7         Connection connection = ConnectionUtil.getConnection();
8         Channel channel = connection.createChannel();
9         // 声明创建一个交换器
10        channel.exchangeDeclare(EXCHANGE_NAME, "direct");
11
12        /*
13         * 每个消费者发送确认消息到消息队列之前，消息队列不会发送下一个消息到消费者，即消费者和消息
14         队列形成了一应一答

```

```

14      * 限制消息队列发送给同一个消费者的消息不会超过一条（一次来回交流中），消费者一次只处理一次
    消息
15      */
16      int prefetchCount = 1;
17      channel.basicQos(prefetchCount);
18
19      String mString = "hello,cirs, i am direct mode";
20      String routingKey = "info"; // 设置消息的路由键
21      // 将消息发送到交换器
22      channel.basicPublish(EXCHANGE_NAME, routingKey, null, mString.getBytes());
23      System.out.println("消息发送成功! "+mString);
24
25      channel.close();
26      connection.close();
27  }
28 }

```

- 两个消费者

```

1  // 消费者
2  public class Receive1 {
3
4      private static final String QUEUE_NAME = "test_queue_direct1";
5      private static final String EXCHANGE_NAME = "test_exchange_direct";
6
7      public static void main(String[] args) throws IOException, TimeoutException,
ShutdownSignalException,
8          ConsumerCancelledException, InterruptedException {
9
10         // 获取一个连接
11         com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();
12
13         // 从连接中获取一个通道
14         Channel channel = connection.createChannel();
15
16         // 创建队列声明
17         channel.queueDeclare(QUEUE_NAME, false, false, false, null);
18
19         // 绑定消息队列到交换器(需要指定routingKey)
20         String routingKey = "error";
21         channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, routingKey);
22
23         channel.basicQos(1); // 保证一次只接受一个消息
24
25         // 创建一个消费者
26         DefaultConsumer consumer = new DefaultConsumer(channel) {
27             // 消息到达触发该方法
28             @Override
29             public void handleDelivery(String consumerTag, Envelope envelope,
BasicProperties properties, byte[] body)
30                 throws IOException {
31                 System.out.println("Receive1 ----" + new String(body));

```

```

32         try {
33             Thread.sleep(500);
34         } catch (InterruptedException e) {
35
36             e.printStackTrace();
37         } finally {
38             System.out.println("Receive1 has done");
39             // 手动回应消息队列已经处理好了
40             channel.basicAck(envelope.getDeliveryTag(), false);
41         }
42     }
43 };
44
45 // 监听队列
46 boolean autoAck = false; // 关闭自动应答
47 channel.basicConsume(QUEUE_NAME, autoAck, consumer);
48 }
49 }
50
51 // 消费者
52 public class Receive2 {
53
54     private static final String QUEUE_NAME = "test_queue_direct2";
55     private static final String EXCHANGE_NAME = "test_exchange_direct";
56
57     public static void main(String[] args) throws IOException, TimeoutException,
ShutdownSignalException,
58         ConsumerCancelledException, InterruptedException {
59
60         // 获取一个连接
61         com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();
62
63         // 从连接中获取一个通道
64         Channel channel = connection.createChannel();
65
66         // 创建队列声明
67         channel.queueDeclare(QUEUE_NAME, false, false, false, null);
68
69         // 绑定消息队列到交换器(需要指定routingKey)
70         String routingKey = "error";
71         String routingKey1 = "info";
72         String routingKey2 = "warn";
73         channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, routingKey);
74         channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, routingKey1);
75         channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, routingKey2);
76
77         channel.basicQos(1); // 保证一次只接受一个消息
78
79         // 创建一个消费者
80         DefaultConsumer consumer = new DefaultConsumer(channel) {
81             // 消息到达触发该方法
82             @Override

```

```

83         public void handleDelivery(String consumerTag, Envelope envelope,
BasicProperties properties, byte[] body)
84             throws IOException {
85             System.out.println("Receive2 ----" + new String(body));
86             try {
87                 Thread.sleep(500);
88             } catch (InterruptedException e) {
89
90                 e.printStackTrace();
91             } finally {
92                 System.out.println("Receive2 has done");
93                 // 手动回应消息队列已经处理好了
94                 channel.basicAck(envelope.getDeliveryTag(), false);
95             }
96         }
97     };
98
99     // 监听队列
100     boolean autoAck = false; // 关闭自动应答
101     channel.basicConsume(QUEUE_NAME, autoAck, consumer);
102 }
103 }

```

## 5.3 topic

- 生产者

```

1  //消息生产者
2  public class Send {
3
4      private static final String EXCHANGE_NAME = "test_exchange_topic";
5
6      public static void main(String[] args) throws IOException, TimeoutException {
7          // 获取一个连接
8          com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();
9
10         // 从连接中获取一个通道
11         Channel channel = connection.createChannel();
12
13         // 交换器声明
14         channel.exchangeDeclare(EXCHANGE_NAME, "topic");
15
16         /*
17          * 每个消费者发送确认消息到消息队列之前，消息队列不会发送下一个消息到消费者，即消费者和消息
队列形成了一应一答
18          * 限制消息队列发送给同一个消费者的消息不会超过一条（一次来回交流中），消费者一次只处理一次
消息
19          */
20         int prefetchCount = 1;
21         channel.basicQos(1);
22
23         String msg = "hello,cris,i am topic...";

```

```

24     channel.basicPublish(EXCHANGE_NAME, "goods.delete", null, msg.getBytes());
25
26     System.out.println("发送信息成功" + msg);
27     channel.close();
28     connection.close();
29
30 }
31 }

```

- 两个消费者

```

1  // 消费者
2  public class Receive1 {
3
4      private static final String QUEUE_NAME = "test_queue_topic";
5      private static final String EXCHANGE_NAME = "test_exchange_topic";
6
7      public static void main(String[] args) throws IOException, TimeoutException,
ShutdownSignalException,
8          ConsumerCancelledException, InterruptedException {
9
10         // 获取一个连接
11         com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();
12
13         // 从连接中获取一个通道
14         Channel channel = connection.createChannel();
15
16         // 创建队列声明
17         channel.queueDeclare(QUEUE_NAME, false, false, false, null);
18
19         // 绑定消息队列到交换机
20         channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "goods.update");
21
22         channel.basicQos(1); // 保证一次只接受一个消息
23
24         // 创建一个消费者
25         DefaultConsumer consumer = new DefaultConsumer(channel) {
26             // 消息到达触发该方法
27             @Override
28             public void handleDelivery(String consumerTag, Envelope envelope,
BasicProperties properties, byte[] body)
29                 throws IOException {
30                 System.out.println("Receive1 ----" + new String(body));
31                 try {
32                     Thread.sleep(500);
33                 } catch (InterruptedException e) {
34
35                     e.printStackTrace();
36                 } finally {
37                     System.out.println("Receive1 has done");
38                     // 手动回应消息队列已经处理好了
39                     channel.basicAck(envelope.getDeliveryTag(), false);

```

```

40         }
41     }
42 };
43
44 // 监听队列
45 boolean autoAck = false; // 关闭自动应答
46 channel.basicConsume(QueueName, autoAck, consumer);
47 }
48 }
49
50 // 消费者
51 public class Receive2 {
52
53     // 每个消费者绑定不同的队列
54     private static final String QueueName = "test_queue_topic2";
55     // 每个队列绑定相同的交换器
56     private static final String ExchangeName = "test_exchange_topic";
57
58     public static void main(String[] args) throws IOException, TimeoutException,
ShutdownSignalException,
59         ConsumerCancelledException, InterruptedException {
60
61         // 获取一个连接
62         com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();
63
64         // 从连接中获取一个通道
65         Channel channel = connection.createChannel();
66
67         // 创建队列声明
68         channel.queueDeclare(QueueName, false, false, false, null);
69
70         // 绑定消息队列到交换器
71         channel.queueBind(QueueName, ExchangeName, "goods.#");
72
73         channel.basicQos(1); // 保证一次只接受一个消息
74
75         // 创建一个消费者
76         DefaultConsumer consumer = new DefaultConsumer(channel) {
77             // 消息到达触发该方法
78             @Override
79             public void handleDelivery(String consumerTag, Envelope envelope,
BasicProperties properties, byte[] body)
80                 throws IOException {
81                 System.out.println("Receive2 ----" + new String(body));
82                 try {
83                     Thread.sleep(500);
84                 } catch (InterruptedException e) {
85
86                     e.printStackTrace();
87                 } finally {
88                     System.out.println("Receiv2 has done");
89                     // 手动回应消息队列已经处理好了
90                     channel.basicAck(envelope.getDeliveryTag(), false);

```



```

91         }
92     }
93 };
94
95 // 监听队列
96 boolean autoAck = false; // 关闭自动应答
97 channel.basicConsume(QUEUE_NAME, autoAck, consumer);
98 }
99 }

```

## 6. 注意

工作队列（一般使用能者多劳模式，此时需要消费者手动发送应答给消息队列）针对的是一个消息队列有多个消费者来消费，消息队列一般需要持久化消息 订阅/发布模式（fanout, direct, topic）是针对消息（携带路由键）发送到交换器，由交换器来根据路由键将消息转发到对应的消息队列中（消息队列中也要设置路由键）

## 7. 消息确认机制

生产者将消息发送给rabbitmq 服务器需要知道消息是否成功到达（默认生产者是不知道的）可以通过 AMOQ 实现了事务机制/Confirm模式 两种方式解决

### 7.1 事务模式

生产者channel设置事务确认消息发送，不建议这种模式，影响消息的吞吐量，效率低

```

1 // 消息生产者
2 public class TxSend {
3
4     private static final String QUEUE_NAME = "test_queue_tx";
5
6     public static void main(String[] args) throws IOException, TimeoutException {
7         // 获取一个连接
8         com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();
9
10        // 从连接中获取一个通道
11        Channel channel = connection.createChannel();
12
13        // 创建队列声明
14        channel.queueDeclare(QUEUE_NAME, false, false, false, null);
15
16        String msg = "hello,cris,i am tx";
17
18        // 开启事务模式
19        try {
20            channel.txSelect();
21            channel.basicPublish("", QUEUE_NAME, null, msg.getBytes());
22            int i = 1/0;
23            channel.txCommit(); // 事务提交最好放在最后面执行
24        } catch (Exception e) {
25            channel.txRollback();

```

```

26         e.printStackTrace();
27         System.out.println("-----");
28     }finally{
29         System.out.println("send msg : " + msg);
30         channel.close();
31         connection.close();
32     }
33 }
34 }

```

## 7.2 confirm 模式

和事务模式互斥（需创建新消息队列）；rabbitmq不允许随便修改消息队列属性

- 1 - 普通模式：发送一条返回回执再发送一条（一定要先启动发送端创建队列，然后再启动服务端）
- 2 - 批量模式：多条消息发送返回回执
- 3 - 异步模式：由生产者维护自己维护消息是否发送成功（根据rabbitmq服务器的回执进行判断并后续处理）

```

1  // 消费者
2  public class Confirm_Receive {
3
4      private static final String QUEUE_NAME = "test_queue_confirm";
5
6      public static void main(String[] args) throws IOException, TimeoutException,
ShutdownSignalException,
7          ConsumerCancelledException, InterruptedException {
8
9          // 获取一个连接
10         com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();
11
12         // 从连接中获取一个通道
13         Channel channel = connection.createChannel();
14
15         // 创建一个消费者
16         DefaultConsumer consumer = new DefaultConsumer(channel) {
17             @Override
18             public void handleDelivery(String consumerTag, Envelope envelope,
BasicProperties properties, byte[] body)
19                 throws IOException {
20                 System.out.println(new String(body) + "-----");
21             }
22         };
23
24         // 监听队列
25         channel.basicConsume(QUEUE_NAME, true, consumer);
26     }
27 }
28
29 // confirm普通模式
30 public class Confirm_Send {

```

```

31
32 // 和事务型队列互斥
33 private static final String QUEUE_NAME = "test_queue_confirm";
34
35 public static void main(String[] args) throws IOException, TimeoutException,
InterruptedException {
36     // 获取一个连接
37     com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();
38
39     // 从连接中获取一个通道
40     Channel channel = connection.createChannel();
41
42     // 创建队列声明
43     channel.queueDeclare(QUEUE_NAME, false, false, false, null);
44
45     // 将生产者设置为confirm 模式
46     channel.confirmSelect();
47
48     String msg = "hello,cris,i am confirm";
49     channel.basicPublish("", QUEUE_NAME, null, msg.getBytes());
50     if(!channel.waitForConfirms()) {
51         System.out.println("send fail!");
52     }else {
53         System.out.println("send success!");
54     }
55 }
56 }
57
58 // confirm批量模式
59 public class Confirm_Send2 {
60
61     // 和事务型队列互斥
62     private static final String QUEUE_NAME = "test_queue_confirm";
63
64     public static void main(String[] args) throws IOException, TimeoutException,
InterruptedException {
65         // 获取一个连接
66         com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();
67
68         // 从连接中获取一个通道
69         Channel channel = connection.createChannel();
70
71         // 创建队列声明
72         channel.queueDeclare(QUEUE_NAME, false, false, false, null);
73
74         // 将生产者设置为confirm 模式
75         channel.confirmSelect();
76
77         String msg = "hello,cris,i am confirm";
78
79         // 批量发送
80         for (int i = 0; i < 10; i++) {
81

```

```

82         channel.basicPublish("", QUEUE_NAME, null, msg.getBytes());
83     }
84     // 再确认
85     if(!channel.waitForConfirms()) {
86         System.out.println("send fail!");
87     }else {
88         System.out.println("send success!");
89     }
90 }
91 }
92
93 // confirm异步模式
94 public class Confirm_Send3 {
95
96     // 和事务型队列互斥
97     private static final String QUEUE_NAME = "test_queue_confirm_async";
98
99     public static void main(String[] args) throws IOException, TimeoutException,
100 InterruptedException {
101         // 获取一个连接
102         com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();
103
104         // 从连接中获取一个通道
105         Channel channel = connection.createChannel();
106
107         // 创建队列声明
108         channel.queueDeclare(QUEUE_NAME, false, false, false, null);
109
110         // 将生产者设置为confirm 模式
111         channel.confirmSelect();
112
113         // 生产者需要维护一个发送的消息的序列号集合
114         final SortedSet<Long> confirmSet = Collections.synchronizedSortedSet(new
115 TreeSet<Long>());
116
117         channel.addConfirmListener(new ConfirmListener() {
118
119             // 消息发送失败怎么处理
120             @Override
121             public void handleNack(long deliveryTag, boolean multiple) throws IOException {
122                 System.out.println("Nack, SeqNo: " + deliveryTag + ", multiple:" +
123 multiple);
124                 if (multiple) {
125                     confirmSet.headSet(deliveryTag + 1).clear();
126                 } else {
127                     confirmSet.remove(deliveryTag);
128                 }
129             }
130
131             // 消息发送成功怎么处理
132             @Override
133             public void handleAck(long deliveryTag, boolean multiple) throws IOException {
134                 if (multiple) {

```

```

132         System.out.println("~~~~~"+ deliveryTag);
133         confirmSet.headSet(deliveryTag + 1).clear();
134     } else {
135         System.out.println("~~~~~"+ deliveryTag);
136         confirmSet.remove(deliveryTag);
137     }
138 }
139 });
140
141 String msg = "hello,cris,i am confirm";
142 for (int i = 0; i < 10; i++) {
143     long no = channel.getNextPublishSeqNo();
144     System.out.println("-----no"+no);
145
146     channel.basicPublish("", QUEUE_NAME, null, msg.getBytes());
147
148     confirmSet.add(no);
149 }
150 }
151 }

```

## 8. Spring 整合RabbitMQ

### 8.1 pom.xml

```

1     <dependency>
2         <groupId>org.springframework.amqp</groupId>
3         <artifactId>spring-rabbit</artifactId>
4         <version>1.7.5.RELEASE</version>
5     </dependency>

```

### 8.2 application.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xmlns:rabbit="http://www.springframework.org/schema/rabbit"
5     xsi:schemaLocation="http://www.springframework.org/schema/beans
6         http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
7         http://www.springframework.org/schema/rabbit
8         http://www.springframework.org/schema/rabbit/spring-rabbit-1.7.xsd">
9
10     <!-- 1. 定义RabbitMQ 的连接工厂 -->
11     <rabbit:connection-factory id="connectionFactory" host="120.78.138.11" port="5672"
12         username="cris" password="123" virtual-host="/vhost_cris"/>
13
14     <!-- 2. 定义Rabbit 模板, 指定连接工厂和交换器 -->
15     <rabbit:template id="rabbitTemplate" connection-factory="connectionFactory"
16         exchange="fanoutExchange"></rabbit:template>

```

```

14 <!-- 定义MQ 的管理 -->
15 <rabbit:admin connection-factory="connectionFactory"/>
16
17 <!-- 定义队列,自动声明 -->
18 <rabbit:queue name="myQueue" auto-declare="true" durable="true" ></rabbit:queue>
19
20 <!-- 定义交换器, 自动声明 -->
21 <rabbit:fanout-exchange name="fanoutExchange" auto-declare="true">
22     <rabbit:bindings>
23         <rabbit:binding queue="myQueue"></rabbit:binding>
24     </rabbit:bindings>
25 </rabbit:fanout-exchange>
26
27 <!-- 3. 消费者 -->
28 <bean id="consumer" class="com.cris.spring.Consumer"></bean>
29
30 <!-- 队列监听 -->
31 <rabbit:listener-container connection-factory="connectionFactory">
32     <rabbit:listener ref="consumer" queue-names="myQueue" method="listen"/>
33 </rabbit:listener-container>
34 </beans>

```

### 8.3 消费者

```

1 public class Consumer {
2
3     public void listen(String str) {
4         System.out.println("-----" + str);
5     }
6 }

```

### 8.4 生产者

```

1 public class Send {
2
3     public static void main(String[] args) throws InterruptedException {
4         AbstractApplicationContext context = new
ClassPathXmlApplicationContext("classpath:application.xml");
5
6         RabbitTemplate template = context.getBean(RabbitTemplate.class);
7
8         // 发送消息
9         template.convertAndSend("cris, i like u!");
10        context.destroy();
11    }
12 }

```

## 9. SpringBoot 整合RabbitMQ 参考我的SpringBoot 高级整合篇

