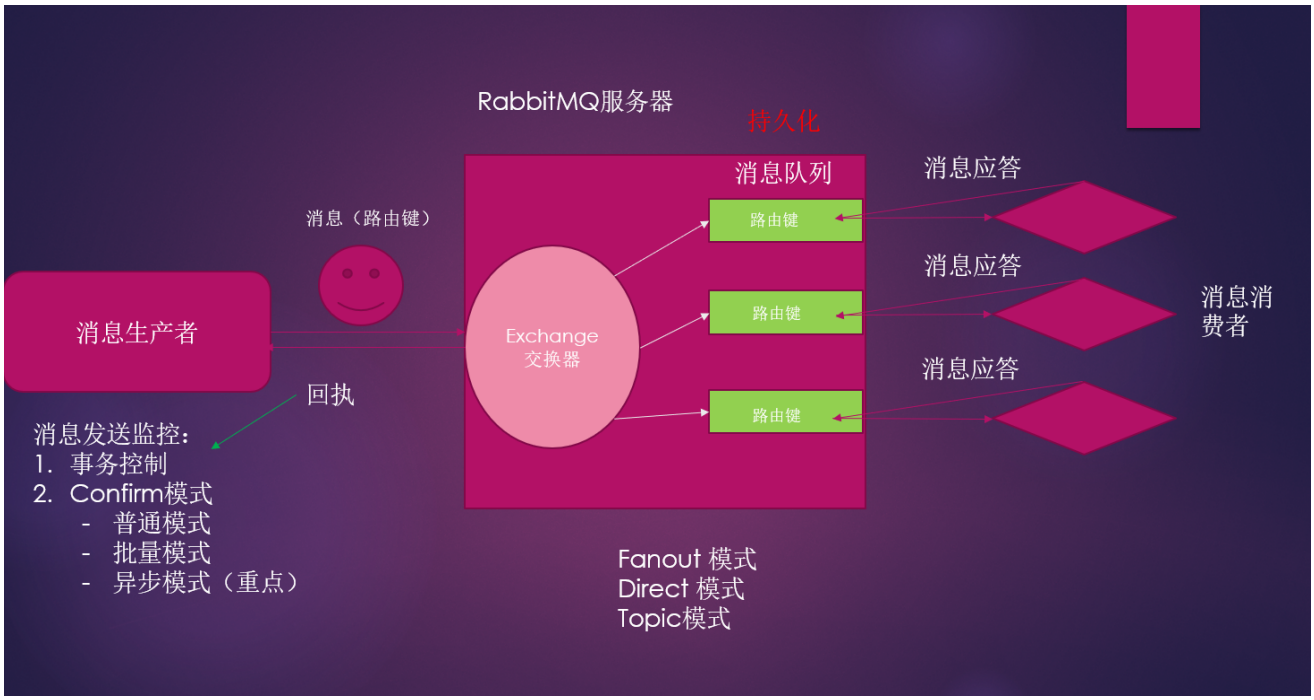
RabbitMQ 学习指南



一. 代码部分

1. 新建一个maven 工程以及连接工具类

```
public class ConnectionUtil {

    public static com.rabbitmq.client.Connection getConnection() throws IOException,
TimeoutException {

        // 定义一个连接工厂
        ConnectionFactory factory = new ConnectionFactory();

        // 设置rabbitMQ 服务器地址（我们使用docker 里的rabbitmq地址）
        factory.setHost("120.78.138.11");

        // 设置AMQP 的协议端口
        factory.setPort(5672);

        // 设置vhost
        factory.setVirtualHost("/vhost_cris");

        // 设置用户名和密码
        factory.setUsername("cris");
        factory.setPassword("123");
```

```
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,
    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
```

```
        Channel channel = connection.createChannel();

        // 创建一个消费者
        DefaultConsumer consumer = new DefaultConsumer(channel) {
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
BasicProperties properties, byte[] body)
                    throws IOException {
                System.out.println(new String(body) + "-----------");
            }
        };

        // 监听队列
        channel.basicConsume(QUEUE_NAME, true, consumer);


    }

    private static void oldApi() throws IOException, TimeoutException, InterruptedException
{
        Connection connection = ConnectionUtil.getConnection();
        Channel channel = connection.createChannel();

        // 定义队列的消费者(老的API)
        QueueingConsumer consumer = new QueueingConsumer(channel);

        // 监听队列
        channel.basicConsume(QUEUE_NAME, true, consumer);
        while (true) {
            Delivery delivery = consumer.nextDelivery();
            System.out.println(new String(delivery.getBody()));
        }
    }
}
```

3. work queue 工作队列（一个消息一个消费者消费）

3.1 生产者

```
// 消息生产者
public class Send {

    private static final String QUEUE_NAME = "test_queue";

    public static void main(String[] args) throws IOException, TimeoutException {
        // 获取一个连接
        com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();

        // 从连接中获取一个通道
        Channel channel = connection.createChannel();

        // 创建队列声明
        channel.queueDeclare(QUEUE_NAME, 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
```

```
        // 监听队列
        channel.basicConsume(QUEUE_NAME, true, consumer);
    }
}
```

3.2 消费者2

```
// 消费者
public class Receive2 {

    private static final String QUEUE_NAME = "test_queue";

    public static void main(String[] args) throws IOException, TimeoutException, ShutdownSignalException,
            ConsumerCancelledException, InterruptedException {

        // 获取一个连接
        com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();

        // 从连接中获取一个通道
        Channel channel = connection.createChannel();

        // 创建队列声明
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);

        // 创建一个消费者
        DefaultConsumer consumer = new DefaultConsumer(channel) {
            // 消息到达触发该方法
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope, BasicProperties properties, byte[] body)
                    throws IOException {
                System.out.println("Receive2 ----"+new String(body));
                try {
                    Thread.sleep(2000);
                } catch (InterruptedException e) {

                    e.printStackTrace();
                }finally {
                    System.out.println("Receive2 has done");
                }
            }
        };

        // 监听队列
        channel.basicConsume(QUEUE_NAME, true, consumer);
    }
}
```

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

4.1 生产者

```
// 消息生产者
public class Send {

    private static final String QUEUE_NAME = "test_queue";

    public static void main(String[] args) throws IOException, TimeoutException {
        // 获取一个连接
        com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();

        // 从连接中获取一个通道
        Channel channel = connection.createChannel();

        // 创建队列声明（代码创建队列）
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);

        /*
         * 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
         * 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
         */
        int prefetchCount = 1;
        channel.basicQos(1);

        for (int i = 0; i < 50; i++) {

            String msg = "hello,cris" +i;
            System.out.println("send msg : " + msg);
            channel.basicPublish("", QUEUE_NAME, null, msg.getBytes());
        }
        channel.close();
        connection.close();
    }
}
```

4.2 消费者1

```
// 消费者
public class Receive1 {

    private static final String QUEUE_NAME = "test_queue";

    public static void main(String[] args) throws IOException, TimeoutException, ShutdownSignalException,
            ConsumerCancelledException, InterruptedException {

```

```
        // 获取一个连接
        com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();

        // 从连接中获取一个通道
        Channel channel = connection.createChannel();

        // 创建队列声明
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);

        channel.basicQos(1); // 保证一次只接受一个消息

        // 创建一个消费者
        DefaultConsumer consumer = new DefaultConsumer(channel) {
            // 消息到达触发该方法
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
BasicProperties properties, byte[] body)
                    throws IOException {
                System.out.println("Receive1 ----" + new String(body));
                try {
                    Thread.sleep(500);
                } catch (InterruptedException e) {

                    e.printStackTrace();
                } finally {
                    System.out.println("Receive1 has done");
                    // 手动回应消息队列已经处理好了
                    channel.basicAck(envelope.getDeliveryTag(), false);
                }
            }
        };

        // 监听队列
        boolean autoAck = false; // 关闭自动应答
        channel.basicConsume(QUEUE_NAME, autoAck, consumer);

    }
}
```

4.2 消费者2

```
// 消费者
public class Receive2 {

    private static final String QUEUE_NAME = "test_queue";

    public static void main(String[] args) throws IOException, TimeoutException,
ShutdownSignalException,
            ConsumerCancelledException, InterruptedException {

        // 获取一个连接
        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("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(QUEUE_NAME, 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 {
```

```
        // 获取一个连接
        com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();

        // 从连接中获取一个通道
        Channel channel = connection.createChannel();

        // 交换器声明
        channel.exchangeDeclare(EXCHANGE_NAME, "fanout");

        /*
         * 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
         * 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
         */
        int prefetchCount = 1;
        channel.basicQos(1);

        String msg = "hello,cris,i am exchange...";
        channel.basicPublish(EXCHANGE_NAME, "", null, msg.getBytes());
        System.out.println("发送信息成功" + msg);

        channel.close();
        connection.close();

    }
}
```

- 两个消费者

```
// 消费者
public class Receive1 {

    private static final String QUEUE_NAME = "test_queue_fanout";
    private static final String EXCHANGE_NAME = "test_queue_exchange";

    public static void main(String[] args) throws IOException, TimeoutException, ShutdownSignalException,
            ConsumerCancelledException, InterruptedException {

        // 获取一个连接
        com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();

        // 从连接中获取一个通道
        Channel channel = connection.createChannel();

        // 创建队列声明
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);

        // 绑定消息队列到交换器
        channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "");

```

```
        channel.basicQos(1); // 保证一次只接受一个消息

        // 创建一个消费者
        DefaultConsumer consumer = new DefaultConsumer(channel) {
            // 消息到达触发该方法
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
BasicProperties properties, byte[] body)
                    throws IOException {
                System.out.println("Receive1 ----" + new String(body));
                try {
                    Thread.sleep(500);
                } catch (InterruptedException e) {

                    e.printStackTrace();
                } finally {
                    System.out.println("Receive1 has done");
                    // 手动回应消息队列已经处理好了
                    channel.basicAck(envelope.getDeliveryTag(), false);
                }
            }
        };

        // 监听队列
        boolean autoAck = false; // 关闭自动应答
        channel.basicConsume(QUEUE_NAME, autoAck, consumer);

    }
}

// 消费者
public class Receive2 {

    // 每个消费者绑定不同的队列
    private static final String QUEUE_NAME = "test_queue2_fanout";
    // 每个队列绑定相同的交换器
    private static final String EXCHANGE_NAME = "test_queue_exchange";

    public static void main(String[] args) throws IOException, TimeoutException,
ShutdownSignalException,
            ConsumerCancelledException, InterruptedException {

        // 获取一个连接
        com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();

        // 从连接中获取一个通道
        Channel channel = connection.createChannel();

        // 创建队列声明
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);

        // 绑定消息队列到交换器
        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,
    BasicProperties properties, byte[] body)
81                      throws IOException {
82                  System.out.println("Receive2 ----" + new String(body));
83                  try {
84                      Thread.sleep(500);
85                  } catch (InterruptedException e) {
86
87                      e.printStackTrace();
88                  } finally {
89                      System.out.println("Receiv2 has done");
90                      // 手动回应消息队列已经处理好了
91                      channel.basicAck(envelope.getDeliveryTag(), false);
92                  }
93              }
94          };
95
96          // 监听队列
97          boolean autoAck = false; // 关闭自动应答
98          channel.basicConsume(QUEUE_NAME, autoAck, consumer);
99
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
    队列形成了一应一答
```

```
         * 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
         */
        int prefetchCount = 1;
        channel.basicQos(prefetchCount);

        String mString = "hello,cirs, i am direct mode";
        String routingKey = "info"; // 设置消息的路由键
        // 将消息发送到交换器
        channel.basicPublish(EXCHANGE_NAME, routingKey, null, mString.getBytes());
        System.out.println("消息发送成功！"+mString);

        channel.close();
        connection.close();
    }
}
```

- 两个消费者

```
// 消费者
public class Receive1 {

    private static final String QUEUE_NAME = "test_queue_direct1";
    private static final String EXCHANGE_NAME = "test_exchange_direct";

    public static void main(String[] args) throws IOException, TimeoutException, ShutdownSignalException,
            ConsumerCancelledException, InterruptedException {

        // 获取一个连接
        com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();

        // 从连接中获取一个通道
        Channel channel = connection.createChannel();

        // 创建队列声明
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);

        // 绑定消息队列到交换器(需要指定routingKey)
        String routingKey = "error";
        channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, routingKey);

        channel.basicQos(1); // 保证一次只接受一个消息

        // 创建一个消费者
        DefaultConsumer consumer = new DefaultConsumer(channel) {
            // 消息到达触发该方法
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope, BasicProperties properties, byte[] body)
                    throws IOException {
                System.out.println("Receive1 ----" + new String(body));
```

```
                try {
                    Thread.sleep(500);
                } catch (InterruptedException e) {

                    e.printStackTrace();
                } finally {
                    System.out.println("Receive1 has done");
                    // 手动回应消息队列已经处理好了
                    channel.basicAck(envelope.getDeliveryTag(), false);
                }
            }
        };

        // 监听队列
        boolean autoAck = false; // 关闭自动应答
        channel.basicConsume(QUEUE_NAME, autoAck, consumer);
    }
}

// 消费者
public class Receive2 {

    private static final String QUEUE_NAME = "test_queue_direct2";
    private static final String EXCHANGE_NAME = "test_exchange_direct";

    public static void main(String[] args) throws IOException, TimeoutException,
ShutdownSignalException,
            ConsumerCancelledException, InterruptedException {

        // 获取一个连接
        com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();

        // 从连接中获取一个通道
        Channel channel = connection.createChannel();

        // 创建队列声明
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);

        // 绑定消息队列到交换器(需要指定routingKey)
        String routingKey = "error";
        String routingKey1 = "info";
        String routingKey2 = "warn";
        channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, routingKey);
        channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, routingKey1);
        channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, routingKey2);

        channel.basicQos(1); // 保证一次只接受一个消息

        // 创建一个消费者
        DefaultConsumer consumer = new DefaultConsumer(channel) {
            // 消息到达触发该方法
            @Override
```

```
            public void handleDelivery(String consumerTag, Envelope envelope,
BasicProperties properties, byte[] body)
                    throws IOException {
                System.out.println("Receive2 ----" + new String(body));
                try {
                    Thread.sleep(500);
                } catch (InterruptedException e) {

                    e.printStackTrace();
                } finally {
                    System.out.println("Receive2 has done");
                    // 手动回应消息队列已经处理好了
                    channel.basicAck(envelope.getDeliveryTag(), false);
                }
            }
        };

        // 监听队列
        boolean autoAck = false; // 关闭自动应答
        channel.basicConsume(QUEUE_NAME, autoAck, consumer);
    }
}
```

5.3 topic

- 生产者

```
//消息生产者
public class Send {

    private static final String EXCHANGE_NAME = "test_exchange_topic";

    public static void main(String[] args) throws IOException, TimeoutException {
        // 获取一个连接
        com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();

        // 从连接中获取一个通道
        Channel channel = connection.createChannel();

        // 交换器声明
        channel.exchangeDeclare(EXCHANGE_NAME, "topic");

        /*
         * 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
         * 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
         */
        int prefetchCount = 1;
        channel.basicQos(1);

        String msg = "hello,cris,i am topic...";
```

```
        channel.basicPublish(EXCHANGE_NAME, "goods.delete", null, msg.getBytes());

        System.out.println("发送信息成功" + msg);
        channel.close();
        connection.close();

    }
}
```

- 两个消费者

```
// 消费者
public class Receive1 {

    private static final String QUEUE_NAME = "test_queue_topic";
    private static final String EXCHANGE_NAME = "test_exchange_topic";

    public static void main(String[] args) throws IOException, TimeoutException,
ShutdownSignalException,
            ConsumerCancelledException, InterruptedException {

        // 获取一个连接
        com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();

        // 从连接中获取一个通道
        Channel channel = connection.createChannel();

        // 创建队列声明
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);

        // 绑定消息队列到交换器
        channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "goods.update");

        channel.basicQos(1); // 保证一次只接受一个消息

        // 创建一个消费者
        DefaultConsumer consumer = new DefaultConsumer(channel) {
            // 消息到达触发该方法
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
BasicProperties properties, byte[] body)
                    throws IOException {
                System.out.println("Receive1 ----" + new String(body));
                try {
                    Thread.sleep(500);
                } catch (InterruptedException e) {

                    e.printStackTrace();
                } finally {
                    System.out.println("Receive1 has done");
                    // 手动回应消息队列已经处理好了
                    channel.basicAck(envelope.getDeliveryTag(), false);
```

```
                }
            }
        };

        // 监听队列
        boolean autoAck = false; // 关闭自动应答
        channel.basicConsume(QUEUE_NAME, autoAck, consumer);
    }
}

// 消费者
public class Receive2 {

    // 每个消费者绑定不同的队列
    private static final String QUEUE_NAME = "test_queue_topic2";
    // 每个队列绑定相同的交换器
    private static final String EXCHANGE_NAME = "test_exchange_topic";

    public static void main(String[] args) throws IOException, TimeoutException,
ShutdownSignalException,
            ConsumerCancelledException, InterruptedException {

        // 获取一个连接
        com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();

        // 从连接中获取一个通道
        Channel channel = connection.createChannel();

        // 创建队列声明
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);

        // 绑定消息队列到交换器
        channel.queueBind(QUEUE_NAME, EXCHANGE_NAME, "goods.#");

        channel.basicQos(1); // 保证一次只接受一个消息

        // 创建一个消费者
        DefaultConsumer consumer = new DefaultConsumer(channel) {
            // 消息到达触发该方法
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
BasicProperties properties, byte[] body)
                    throws IOException {
                System.out.println("Receive2 ----" + new String(body));
                try {
                    Thread.sleep(500);
                } catch (InterruptedException e) {

                    e.printStackTrace();
                } finally {
                    System.out.println("Receiv2 has done");
                    // 手动回应消息队列已经处理好了
                    channel.basicAck(envelope.getDeliveryTag(), false);
```

```
                }
            }
        };

        // 监听队列
        boolean autoAck = false; // 关闭自动应答
        channel.basicConsume(QUEUE_NAME, autoAck, consumer);
    }
}
```

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

```
// 消息生产者
public class TxSend {

    private static final String QUEUE_NAME = "test_queue_tx";

    public static void main(String[] args) throws IOException, TimeoutException {
        // 获取一个连接
        com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();

        // 从连接中获取一个通道
        Channel channel = connection.createChannel();

        // 创建队列声明
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);

        String msg = "hello,cris,i am tx";

        // 开启事务模式
        try {
            channel.txSelect();
            channel.basicPublish("", QUEUE_NAME, null, msg.getBytes());
            int i = 1/0;
            channel.txCommit();     // 事务提交最好放在最后面执行
        } catch (Exception e) {
            channel.txRollback();
```

```
            e.printStackTrace();
            System.out.println("-----------");
        }finally{
            System.out.println("send msg : " + msg);
            channel.close();
            connection.close();
        }
    }
}
```

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

```
- 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）
- 批量模式：多条消息发送返回回执
- 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）
```

```
// 消费者
public class Confirm_Receive {

    private static final String QUEUE_NAME = "test_queue_confirm";

    public static void main(String[] args) throws IOException, TimeoutException,
ShutdownSignalException,
            ConsumerCancelledException, InterruptedException {

        // 获取一个连接
        com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();

        // 从连接中获取一个通道
        Channel channel = connection.createChannel();

        // 创建一个消费者
        DefaultConsumer consumer = new DefaultConsumer(channel) {
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
BasicProperties properties, byte[] body)
                    throws IOException {
                System.out.println(new String(body) + "-----------");
            }
        };

        // 监听队列
        channel.basicConsume(QUEUE_NAME, true, consumer);
    }
}

// confirm普通模式
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
```

```
            channel.basicPublish("", QUEUE_NAME, null, msg.getBytes());
        }
        // 再确认
        if(!channel.waitForConfirms()) {
            System.out.println("send fail!");
        }else {
            System.out.println("send success!");
        }
    }
}

// confirm异步模式
public class Confirm_Send3 {

    // 和事务型队列互斥
    private static final String QUEUE_NAME = "test_queue_confirm_async";

    public static void main(String[] args) throws IOException, TimeoutException,
InterruptedException {
        // 获取一个连接
        com.rabbitmq.client.Connection connection = ConnectionUtil.getConnection();

        // 从连接中获取一个通道
        Channel channel = connection.createChannel();

        // 创建队列声明
        channel.queueDeclare(QUEUE_NAME, false, false, false, null);

        // 将生产者设置为confirm 模式
        channel.confirmSelect();

        // 生产者需要维护一个发送的消息的序列号集合
        final SortedSet<Long> confirmSet = Collections.synchronizedSortedSet(new
TreeSet<Long>());

        channel.addConfirmListener(new ConfirmListener() {

            // 消息发送失败怎么处理
            @Override
            public void handleNack(long deliveryTag, boolean multiple) throws IOException {
                System.out.println("Nack, SeqNo: " + deliveryTag + ", multiple:" +
multiple);
                if (multiple) {
                    confirmSet.headSet(deliveryTag + 1).clear();
                } else {
                    confirmSet.remove(deliveryTag);
                }
            }

            // 消息发送成功怎么处理
            @Override
            public void handleAck(long deliveryTag, boolean multiple) throws IOException {
                if (multiple) {
```

```
                    System.out.println("~~~~~~~~"+ deliveryTag);
                    confirmSet.headSet(deliveryTag + 1).clear();
                } else {
                    System.out.println("~~~~~~~~"+ deliveryTag);
                    confirmSet.remove(deliveryTag);
                }
            }
        });

        String msg = "hello,cris,i am confirm";
        for (int i = 0; i < 10; i++) {
            long no = channel.getNextPublishSeqNo();
            System.out.println("--------------------no"+no);

            channel.basicPublish("", QUEUE_NAME, null, msg.getBytes());

            confirmSet.add(no);
        }
    }
}
```

8. Spring 整合RabbitMQ

8.1 pom.xml

```
    <dependency>
      <groupId>org.springframework.amqp</groupId>
      <artifactId>spring-rabbit</artifactId>
      <version>1.7.5.RELEASE</version>
    </dependency>
```

8.2 application.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:rabbit="http://www.springframework.org/schema/rabbit"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
    http://www.springframework.org/schema/rabbit
http://www.springframework.org/schema/rabbit/spring-rabbit-1.7.xsd">

  <!-- 1. 定义RabbitMQ 的连接工厂 -->
  <rabbit:connection-factory id="connectionFactory" host="120.78.138.11" port="5672"
username="cris" password="123" virtual-host="/vhost_cris"/>

    <!-- 2. 定义Rabbit 模板，指定连接工厂和交换器 -->
  <rabbit:template id="rabbitTemplate" connection-factory="connectionFactory"
exchange="fanoutExchange"></rabbit:template>

```

```
  <!-- 定义MQ 的管理 -->
  <rabbit:admin connection-factory="connectionFactory"/>

  <!-- 定义队列,自动声明 -->
  <rabbit:queue name="myQueue" auto-declare="true" durable="true" ></rabbit:queue>

  <!-- 定义交换器, 自动声明 -->
  <rabbit:fanout-exchange name="fanoutExchange" auto-declare="true">
    <rabbit:bindings>
      <rabbit:binding queue="myQueue"></rabbit:binding>
    </rabbit:bindings>
  </rabbit:fanout-exchange>

  <!-- 3. 消费者 -->
  <bean id="consumer" class="com.cris.spring.Consumer"></bean>

  <!-- 队列监听 -->
  <rabbit:listener-container connection-factory="connectionFactory">
    <rabbit:listener ref="consumer" queue-names="myQueue" method="listen"/>
  </rabbit:listener-container>
</beans>
```

8.3 消费者

```
public class Consumer {

    public void listen(String str) {
        System.out.println("-----------" + str);
    }
}
```

8.4 生产者

```
public class Send {

    public static void main(String[] args) throws InterruptedException {
        AbstractApplicationContext context = new
ClassPathXmlApplicationContext("classpath:application.xml");

        RabbitTemplate template = context.getBean(RabbitTemplate.class);

        // 发送消息
        template.convertAndSend("cris, i like u!");
        context.destroy();
    }
}
```

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