工具类：

```java
import com.rabbitmq.client.ConnectionFactory;

/**
 * RabbitMQ连接工厂工具类
 * @author qiyang
 */
public class FactoryConnectionUtil {
    private static final String IP_ADDRESS = "192.168.230.128";
    private static final int port = 5672;
     private static final String username = "root";
     private static final String password = "root";
     private static ConnectionFactory factory = null;

     static {
         factory = new ConnectionFactory();
         factory.setHost(IP_ADDRESS);
         factory.setPort(port);
         factory.setUsername(username);
         factory.setPassword(password);
     }

     public static ConnectionFactory getFactory(){
         return factory;
     }
}
```

场景1：单发送单接收

生产者：

```java
/**
 * 发送消息
 * @param exchangeName 交换器名称
 * @param queueName 队列名称
 * @param routingKey 路由键
 */
public static void send01(String exchangeName, String queueName, String routingKey){
    try {
        Connection connection = FactoryConnectionUtil.getFactory().newConnection();//创建连接
         Channel channel = connection.createChannel();//创建信道
         //创建一个type="direct"、持久化、非自动删除的交换器
         channel.exchangeDeclare(exchangeName, BuiltinExchangeType.DIRECT, true, false, null);
         //创建一个持久化、非排他的、非自动删除的队列
         channel.queueDeclare(queueName, true, false, false, null);
         //将交换器与队列通过路由键绑定
```

```
        channel.queueBind(queueName, exchangeName, routingKey);
        //发送一条持久化的消息: hello world
        String message = "hello world";
        channel.basicPublish(exchangeName, routingKey, MessageProperties.PERSISTENT_TEXT_PLAIN, mes
        channel.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

消费者：

```
/**
 * 接收消息
 * @param queueName 队列名称
 */
public static void receive_01(String queueName){
    try {
        Connection connection = FactoryConnectionUtil.getFactory().newConnection();/* 创建连接 */
        Channel channel = connection.createChannel();//创建信道
        Consumer consumer = new DefaultConsumer(channel){

            @Override
            public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties
                System.out.println("收到消息 : " + new String(body));
                channel.basicAck(envelope.getDeliveryTag(), false);
            }
        };

        channel.basicConsume(queueName, consumer);

        //休眠60秒后，信道关闭，关闭之后无法接收消息
        TimeUnit.SECONDS.sleep(60);
        channel.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

测试代码：

发送端：

```
@Test
public void testSend01() {
    Producer.send01("ex1", "q1", "k1");
}
```

接收端：

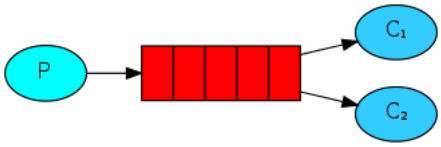
```
@Test
public void testReceive_01(){
    MyConsumer.receive_01("q1");
}
```

结果：

1 生产者发送的消息，发送几条，都会经过RabbitMQ

2 如果发送的消息，消费者还没有消费掉，那么在RabbitMQ中会记录有几条消息，如果已经被消费者消费掉了，那么消息就不会在Ral
3 如果消费者连接之后，会监听RabbitMQ中的消息，如果有就会获取到

场景2：单发送多接收



生产者：

```
/**
 * 发送消息
 * @param exchangeName 交换器名称
 * @param queueName 队列名称
 * @param routingKey 路由键
 */
public static void send01(String exchangeName, String queueName, String routingKey){
    try {
        Connection connection = FactoryConnectionUtil.getFactory().newConnection();//创建连接
        Channel channel = connection.createChannel();//创建信道
        //创建一个type="direct"、持久化、非自动删除的交换器
        channel.exchangeDeclare(exchangeName, BuiltinExchangeType.DIRECT, true, false, null);
        //创建一个持久化、非排他的、非自动删除的队列
        channel.queueDeclare(queueName, true, false, false, null);
        //将交换器与队列通过路由键绑定
        channel.queueBind(queueName, exchangeName, routingKey);
        //发送一条持久化的消息: hello world
        String message = "hello world";
        channel.basicPublish(exchangeName, routingKey, MessageProperties.PERSISTENT_TEXT_PLAIN, mes
        channel.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

消费者：

```
/**
 * 接收消息
 * @param queueName 队列名称
 * @param name 名称
 */
public static void receive_02(String queueName, String name ){
    try {
        Connection connection = FactoryConnectionUtil.getFactory().newConnection();/* 创建连接 */
        Channel channel = connection.createChannel();//创建信道
        Consumer consumer = new DefaultConsumer(channel){

            @Override
            public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties
                System.out.println( name + "收到消息 : " + new String(body));
                channel.basicAck(envelope.getDeliveryTag(), false);
```

```
            }
        };

        channel.basicConsume(queueName, consumer);

        //休眠60秒后，信道关闭，关闭之后无法接收消息
        TimeUnit.SECONDS.sleep(60);
        channel.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

测试代码：

生产者

```
@Test
public void testSend02(){
    for (int i = 0; i < 5; i++) {
        Producer.send01("ex1", "q1", "k1");
    }
}
```

消费者

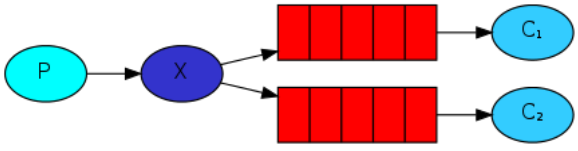
```
@Test
public void testReceive_02_A(){
    MyConsumer.receive_02("q1","name1");
}

@Test
public void testReceive_02_B(){
    MyConsumer.receive_02("q1","name2");
}
```

先运行消费者代码：分别先运行testReceive_02_A、testReceive_02_B
之后运行生产者代码

测试结果：

生产者发送了五条信息，结果发现有的数据被testReceive_02_A 接收到了，有的被testReceive_02_B接收到了

场景3：发布/订阅模式



生产者：

```
/**
 * 发布订阅消息
 * @param exchangeName 交换器名称
 * @param queueName1 队列1名称
 * @param queueName2 队列2名称
```

```
 * @param routingKey 路由键
 */
public static void send_02(String exchangeName, String queueName1, String queueName2, String routing
    try {
        Connection connection = FactoryConnectionUtil.getFactory().newConnection();//创建连接
        Channel channel = connection.createChannel();//创建信道
        //创建一个type="FANOUT"、持久化、非自动删除的交换器
        channel.exchangeDeclare(exchangeName, BuiltinExchangeType.FANOUT, true, false, null);
        //创建一个持久化、非排他的、非自动删除的队列
        channel.queueDeclare(queueName1, true, false, false, null);
        //将交换器与队列通过路由键绑定
        channel.queueBind(queueName1, exchangeName, routingKey);
        channel.queueDeclare(queueName2,true,false,false,null);
        channel.queueBind(queueName2,exchangeName,routingKey);
        //发送一条持久化的消息: hello world
        String message = "hello world";
        channel.basicPublish(exchangeName, routingKey, MessageProperties.PERSISTENT_TEXT_PLAIN, mes
        channel.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

消费者:

```
/**
 * 接收消息
 * @param queueName 队列名称
 * @param name 名称
 */
public static void receive_02(String queueName, String name ){
    try {
        Connection connection = FactoryConnectionUtil.getFactory().newConnection();/* 创建连接 */
        Channel channel = connection.createChannel();//创建信道
        Consumer consumer = new DefaultConsumer(channel){

            @Override
            public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties
                System.out.println( name + "收到消息 : " + new String(body));
                channel.basicAck(envelope.getDeliveryTag(), false);
            }
        };

        channel.basicConsume(queueName, consumer);

        //休眠60秒后, 信道关闭, 关闭之后无法接收消息
        TimeUnit.SECONDS.sleep(60);
        channel.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

测试代码:

生产者：

发送消息到两个队列上

```
@Test
public void testSend03(){
    for (int i = 0; i < 5; i++) {
        Producer.send_02("ex2","q_01","q_02","rKey");
    }
}
```

消费者：

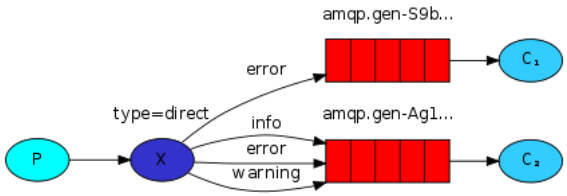
分别接收两个不同的队列

```
@Test
public void testReceive_03_A(){
    MyConsumer.receive_02("q_01","name1");
}

@Test
public void testReceive_03_B(){
    MyConsumer.receive_02("q_02","name2");
}
```

测试结果：

两个消费者，都可以接收到5条信息

场景4：按线路(routing)发送



生产者：

如上图所示发送消息到队列A和B上，结果如图所示

```
public static void send_03(String exchangeName){
    try{
        String queueA = "queueA";
        String queueB = "queueB";
        String routingKeyError = "error";
        String routingKeyInfo = "info";
        String routingKeyWarning="warning";
        Connection connection = FactoryConnectionUtil.getFactory().newConnection();//创建连接
        Channel channel = connection.createChannel();//创建信道
         //创建一个type="direct"、持久化、非自动删除的交换器
         channel.exchangeDeclare(exchangeName, BuiltinExchangeType.DIRECT, true, false, null);
         //创建一个持久化、非排他的、非自动删除的队列
         channel.queueDeclare(queueA, true, false, false, null);
         channel.queueBind(queueA, exchangeName, routingKeyError);

         channel.queueDeclare(queueB,true,false,false,null);
```

```
        channel.queueBind(queueB,exchangeName,routingKeyError);
        channel.queueBind(queueB,exchangeName,routingKeyInfo);
        channel.queueBind(queueB,exchangeName,routingKeyWarning);

        String messageInfo = "info xxx";
        String messageError = "error xxx";
        String messageWarning = "warning xxx";
        channel.basicPublish(exchangeName,routingKeyError, MessageProperties.PERSISTENT_TEXT_PLAIN,
        channel.basicPublish(exchangeName,routingKeyInfo,MessageProperties.PERSISTENT_TEXT_PLAIN,me
        channel.basicPublish(exchangeName,routingKeyWarning,MessageProperties.PERSISTENT_TEXT_PLAIN
        channel.close();
    }catch (Exception e){
        e.printStackTrace();
    }
}
```

消费者：

```
/**
 * 接收消息
 * @param queueName 队列名称
 */
public static void receive_01(String queueName){
    try {
        Connection connection = FactoryConnectionUtil.getFactory().newConnection();/* 创建连接 */
        Channel channel = connection.createChannel();//创建信道
        Consumer consumer = new DefaultConsumer(channel){

            @Override
            public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties
                System.out.println("收到消息 : " + new String(body));
                channel.basicAck(envelope.getDeliveryTag(), false);
            }
        };

        channel.basicConsume(queueName, consumer);

        //休眠60秒后，信道关闭，关闭之后无法接收消息
        TimeUnit.SECONDS.sleep(60);
        channel.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

测试代码：

生产者

```
@Test
public void testSend04() {
  Producer.send_03("myexchange");
}
```

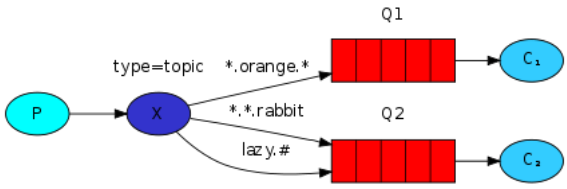
消费者

```
@Test
```

```
public void testReceive_04(){
    MyConsumer.receive_01("queueA");
}

@Test
public void testReceive_05(){
    MyConsumer.receive_01("queueB");
}
```

测试结果 如图所示

场景4：Topic方式发送



生产者：

```
public static void send05(String exChangeName){
    try{
        String queueA = "queueA";
        String queueB = "queueB";
        Connection connection = FactoryConnectionUtil.getFactory().newConnection();//创建连接
        Channel channel = connection.createChannel();//创建信道
        channel.exchangeDeclare(exChangeName,BuiltinExchangeType.TOPIC);

        channel.queueDeclare(queueA, true, false, false, null);
         channel.queueBind(queueA, exChangeName, "*.orange.*");

         channel.queueDeclare(queueB, true, false, false, null);
         channel.queueBind(queueB, exChangeName, "*.*.rabbit");
         channel.queueBind(queueB, exChangeName, "lazy.#");

         channel.basicPublish(exChangeName, "quick.orange.rabbit", null, "quick.orange.rabbit msg!!!
         channel.basicPublish(exChangeName, "lazy.write.elephant", null,"lazy.write.elephant msg !!!

         channel.close();
     }catch (Exception e){
         e.printStackTrace();
     }
}
```

消费者：

```
public static void receive_01(String queueName){
    try {
        Connection connection = FactoryConnectionUtil.getFactory().newConnection();/* 创建连接 */
        Channel channel = connection.createChannel();//创建信道
        Consumer consumer = new DefaultConsumer(channel){

            @Override
```

```
            public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties p
                System.out.println("收到消息 : " + new String(body));
                 channel.basicAck(envelope.getDeliveryTag(), false);
             }
        };

        channel.basicConsume(queueName, consumer);

        //休眠60秒后，信道关闭，关闭之后无法接收消息
        TimeUnit.SECONDS.sleep(60);
        channel.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

测试代码：

生产者

```
@Test
public void testSent05(){
    Producer.send05("topicExchange");
}
```

消费者

```
@Test
public void testReceive_04(){
    MyConsumer.receive_01("queueA");
}

@Test
public void testReceive_05(){
    MyConsumer.receive_01("queueB");
}
```

测试结果：

如图所示：queueA只有一个消息，而queueB有两个消息

”quick.orange.rabbit msg“ 这条消息两边都有收到