一、简单模式

一个生产者对应一个消费者！！

由生产者直接将消费通过队列发生给消费者

生产者

```
package com.znc.rabbitmqapi.quickstart;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;

import java.io.IOException;
import java.util.concurrent.TimeoutException;

/**
 *
 *生产者
 * @Author zhunc
 * @Date 2022/5/13 9:56
 */
public class Producer {
    public static void main(String[] args) throws Exception {
        //1.创建connectionFactory 并进行配置
        ConnectionFactory connectionFactory = new ConnectionFactory();
        //设置服务地址
        connectionFactory.setHost("192.168.1.101");
        //设置端口号
        connectionFactory.setPort(5672);
        //用户名
        connectionFactory.setUsername("root");
        //密码
        connectionFactory.setPassword("root");
        //设置虚拟主机地址，一个用户可以有多个虚拟主机地址，要确保该用户虚拟主机是对的，可以在控制台界面查看用户的虚拟主机地址
        connectionFactory.setVirtualHost("my_host");
        //2.通过连接工厂创建连接
        Connection connection = connectionFactory.newConnection();
        //3.通过连接创建channel，使用通道才能完成消息相关的操作
        Channel channel = connection.createChannel();
        String queueName = "test001";
        //4.发送数据
        for (int i = 0; i < 5; i++) {
```

```
            String msg = "hello rabbitmq!";
            // 向指定的队列中发送消息
            //参数：String exchange, String routingKey, BasicProperties props,
byte[] body
            /**
             * 参数明细：
             * 1、exchange，交换机，如果不指定将使用mq的默认交换机（设置为""）
             * 2、routingKey，路由key，交换机根据路由key来将消息转发到指定的队列，如果使用
默认交换机，routingKey设置为队列的名称
             * 3、props，消息的属性
             * 4、body，消息内容
             */
            channel.basicPublish("", queueName, null, msg.getBytes());
        }

        //5.关闭连接
        channel.close();
        connection.close();

    }
}
```

消费者

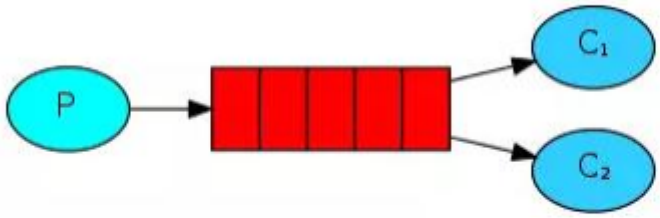
```
package com.znc.rabbitmqapi.quickstart;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;
import com.rabbitmq.client.QueueingConsumer;

/**
 * 消费者
 * @Author zhunc
 * @Date 2022/5/13 9:56
 */
public class Consumer {
    public static void main(String[] args) throws  Exception {
        //1.创建connectionFactory 并进行配置
        ConnectionFactory connectionFactory = new ConnectionFactory();
        connectionFactory.setHost("192.168.1.101");
        connectionFactory.setPort(5672);
        connectionFactory.setUsername("root");
        connectionFactory.setPassword("root");
        connectionFactory.setVirtualHost("my_host");
        //2.通过连接工厂创建连接
        Connection connection = connectionFactory.newConnection();
        //3.通过连接创建channel
        Channel channel = connection.createChannel();
        String queueName = "test001";

        // 4.声明（创建）队列
        //参数：String queue, boolean durable, boolean exclusive, boolean
autoDelete, Map<String, Object> arguments
```

```
        /**
         * 参数明细
         * 1、queue 队列名称
         * 2、durable 是否持久化，如果持久化，mq重启后队列还在
         * 3、exclusive 是否独占连接，队列只允许在该连接中访问，如果connection连接关闭队列则自动删除,如果将此参数设置true可用于临时队列的创建
         * 4、autoDelete 自动删除，队列不再使用时是否自动删除此队列，如果将此参数和exclusive参数设置为true就可以实现临时队列（队列不用了就自动删除）
         * 5、arguments 参数，可以设置一个队列的扩展参数，比如：可设置存活时间
         */
        channel.queueDeclare(queueName, true, false, false, null);
        //5.创建消费者
        QueueingConsumer queueingConsumer = new QueueingConsumer(channel);
        //6.设置channel
        // 监听队列，第二个参数：是否自动进行消息确认。
        //参数：String queue, boolean autoAck, Consumer callback
        /**
         * 参数明细：
         * 1、queue 队列名称
         * 2、autoAck 自动回复，当消费者接收到消息后要告诉mq消息已接收，如果将此参数设置为true表示会自动回复mq，如果设置为false要通过编程实现回复
         * 3、callback，消费方法，当消费者接收到消息要执行的方法
         */
        channel.basicConsume(queueName,true, queueingConsumer);
        while (true) {
            //7.获取消息
            QueueingConsumer.Delivery delivery = queueingConsumer.nextDelivery();
            String msg = new String(delivery.getBody());
            System.out.println("消费端：" + msg);


        }
    }
}
```

二、工作模式

定义：

消息产生者将消息放入队列，消费者可以有多个,消费者1,消费者2同时监听同一个队列,消息被消费。C1 C2共同争抢当前的消息队列内容,谁先拿到谁负责消费消息(隐患：高并发情况下,默认会产生某一个消息被多个消费者共同使用,可以设置一个开关(syncronize) 保证一只能被一个消费者使用)。发布消息与简单模式相同接受消息，增加autoACK



生产者

```
package com.znc.rabbitmqapi.quickstart;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;

import java.io.IOException;
import java.util.concurrent.TimeoutException;

/**
 *
 *生产者
 * @Author zhunc
 * @Date 2022/5/13 9:56
 */
public class Producer {
    public static void main(String[] args) throws Exception {
        //1.创建connectionFactory 并进行配置
        ConnectionFactory connectionFactory = new ConnectionFactory();
        //设置服务地址
        connectionFactory.setHost("192.168.1.101");
        //设置端口号
        connectionFactory.setPort(5672);
        //用户名
        connectionFactory.setUsername("root");
        //密码
        connectionFactory.setPassword("root");
        //设置虚拟主机地址，一个用户可以有多个虚拟主机地址，要确保该用户虚拟主机是对的，可以在控制台界面查看用户的虚拟主机地址
        connectionFactory.setVirtualHost("my_host");
        //2.通过连接工厂创建连接
        Connection connection = connectionFactory.newConnection();
        //3.通过连接创建channel，使用通道才能完成消息相关的操作
        Channel channel = connection.createChannel();
```

```
        String queueName = "test001";
        //4.发送数据
        for (int i = 0; i < 5; i++) {
            String msg = "hello rabbitmq!";
            // 向指定的队列中发送消息
            //参数：String exchange, String routingKey, BasicProperties props,
byte[] body
            /**
             * 参数明细：
             * 1、exchange，交换机，如果不指定将使用mq的默认交换机（设置为""）
             * 2、routingKey，路由key，交换机根据路由key来将消息转发到指定的队列，如果使用
默认交换机，routingKey设置为队列的名称
             * 3、props，消息的属性
             * 4、body，消息内容
             */
            channel.basicPublish("", queueName, null, msg.getBytes());
        }

        //5.关闭连接
        channel.close();
        connection.close();

    }
}
```

消费者1

多个消费者代码都是一样的，只是类名不同

```
package com.znc.rabbitmqapi.work;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;
import com.rabbitmq.client.QueueingConsumer;

/**
 * 消费者
 * @Author zhunc
 * @Date 2022/5/13 9:56
 */
public class Consumer {
    public static void main(String[] args) throws  Exception {
        //1.创建connectionFactory 并进行配置
        ConnectionFactory connectionFactory = new ConnectionFactory();
        connectionFactory.setHost("192.168.1.101");
        connectionFactory.setPort(5672);
        connectionFactory.setUsername("root");
        connectionFactory.setPassword("root");
        connectionFactory.setVirtualHost("my_host");
        //2.通过连接工厂创建连接
        Connection connection = connectionFactory.newConnection();
        //3.通过连接创建channel
        Channel channel = connection.createChannel();
```

```
        String queueName = "test001";
        // 4.声明（创建）队列
        //参数：String queue, boolean durable, boolean exclusive, boolean
autoDelete, Map<String, Object> arguments
        /**
         * 参数明细
         * 1、queue 队列名称
         * 2、durable 是否持久化，如果持久化，mq重启后队列还在
         * 3、exclusive 是否独占连接，队列只允许在该连接中访问，如果connection连接关闭队列
则自动删除,如果将此参数设置true可用于临时队列的创建
         * 4、autoDelete 自动删除，队列不再使用时是否自动删除此队列，如果将此参数和
exclusive参数设置为true就可以实现临时队列（队列不用了就自动删除）
         * 5、arguments 参数，可以设置一个队列的扩展参数，比如：可设置存活时间
         */
        channel.queueDeclare(queueName, true, false, false, null);
        //5.创建消费者
        QueueingConsumer queueingConsumer = new QueueingConsumer(channel);
        //6.设置channel
        // 监听队列，第二个参数：是否自动进行消息确认。
        //参数：String queue, boolean autoAck, Consumer callback
        /**
         * 参数明细：
         * 1、queue 队列名称
         * 2、autoAck 自动回复，当消费者接收到消息后要告诉mq消息已接收，如果将此参数设置为
true表示会自动回复mq，如果设置为false要通过编程实现回复
         * 3、callback，消费方法，当消费者接收到消息要执行的方法
         */
        channel.basicConsume(queueName,true, queueingConsumer);
        while (true) {
            //7.获取消息
            QueueingConsumer.Delivery delivery =
queueingConsumer.nextDelivery();
            String msg = new String(delivery.getBody());
            System.out.println("消费端：" + msg);


        }
    }
}
```

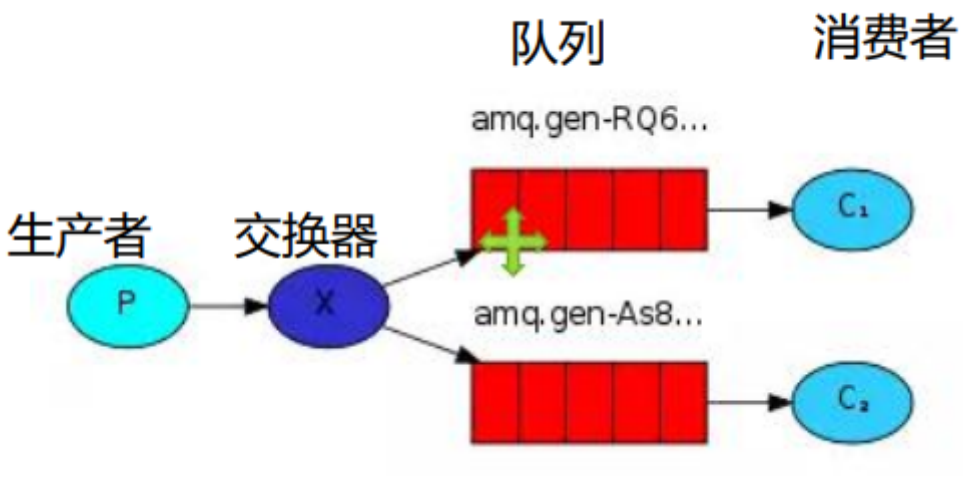
消费者2

```
package com.znc.rabbitmqapi.work;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;
import com.rabbitmq.client.QueueingConsumer;

/**
 * 消费者
 * @Author zhunc
 * @Date 2022/5/13 9:56
 */
```

```
public class Consumer2 {
    public static void main(String[] args) throws  Exception {
        //1.创建connectionFactory 并进行配置
        ConnectionFactory connectionFactory = new ConnectionFactory();
        connectionFactory.setHost("192.168.1.101");
        connectionFactory.setPort(5672);
        connectionFactory.setUsername("root");
        connectionFactory.setPassword("root");
        connectionFactory.setVirtualHost("my_host");
        //2.通过连接工厂创建连接
        Connection connection = connectionFactory.newConnection();
        //3.通过连接创建channel
        Channel channel = connection.createChannel();
        String queueName = "test001";
        // 4.声明（创建）队列
        //参数：String queue, boolean durable, boolean exclusive, boolean
autoDelete, Map<String, Object> arguments
        /**
         * 参数明细
         * 1、queue 队列名称
         * 2、durable 是否持久化，如果持久化，mq重启后队列还在
         * 3、exclusive 是否独占连接，队列只允许在该连接中访问，如果connection连接关闭队列
则自动删除,如果将此参数设置true可用于临时队列的创建
         * 4、autoDelete 自动删除，队列不再使用时是否自动删除此队列，如果将此参数和
exclusive参数设置为true就可以实现临时队列（队列不用了就自动删除）
         * 5、arguments 参数，可以设置一个队列的扩展参数，比如：可设置存活时间
         */
        channel.queueDeclare(queueName, true, false, false, null);
        //5.创建消费者
        QueueingConsumer queueingConsumer = new QueueingConsumer(channel);
        //6.设置channel
        // 监听队列，第二个参数：是否自动进行消息确认。
        //参数：String queue, boolean autoAck, Consumer callback
        /**
         * 参数明细：
         * 1、queue 队列名称
         * 2、autoAck 自动回复，当消费者接收到消息后要告诉mq消息已接收，如果将此参数设置为
true表示会自动回复mq，如果设置为false要通过编程实现回复
         * 3、callback，消费方法，当消费者接收到消息要执行的方法
         */
        channel.basicConsume(queueName,true, queueingConsumer);
        while (true) {
            //7.获取消息
            QueueingConsumer.Delivery delivery =
queueingConsumer.nextDelivery();
            String msg = new String(delivery.getBody());
            System.out.println("消费端: " + msg);


        }
    }
}
```

三、发布订阅 public/subscribe(广播)

定义：每个消费者监听自己的队列；生产者将消息发给broker，由交换机将消息转发到绑定此交换机的每个队列，每个绑定**
交换机的队列都将接收到消息。发送时，指定发送到exchange而不是Queue，exchange为fanout类型**



相关场景:邮件群发,群聊天,广播(广告)

获取channel工具类

```
public class RabbitMqUtils {
     private static  ConnectionFactory connectionFactory;
     private static  Connection connection;
     private static  Channel channel;
    static {
        connectionFactory = new ConnectionFactory();
        //1.创建connectionFactory 并进行配置
        connectionFactory.setHost("192.168.1.101");
        connectionFactory.setPort(5672);
        connectionFactory.setUsername("root");
        connectionFactory.setPassword("root");
        connectionFactory.setVirtualHost("my_host");
        //2.通过连接工厂创建连接
        try {
             connection = connectionFactory.newConnection();
             channel = connection.createChannel();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static Channel getChannel() throws  Exception{

        //3.通过连接创建channel
        return channel;
    }

    public static void close() throws Exception {
        channel.close();
```

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

生产者

```
package com.znc.rabbitmqapi.exchange.fanout;

import com.rabbitmq.client.Channel;
import com.znc.rabbitmqapi.utils.RabbitMqUtils;

/**
 * @Author zhunc
 * @Date 2022/5/13 15:04
 */
public class ProducerFanoutExchange {
    public static void main(String[] args) throws Exception {
        Channel channel = RabbitMqUtils.getChannel();
        //声明交换机名称
        String exchangeName = "test_fanout_exchange";
        //声明路由key
        String routingKey = "";

        for (int i = 0; i < 5; i++) {
            String message =i+" hello world rabbitmq fanout exchange message";
            channel.basicPublish(exchangeName, routingKey, null,
message.getBytes());
        }

        RabbitMqUtils.close();
    }
}
```

消费者

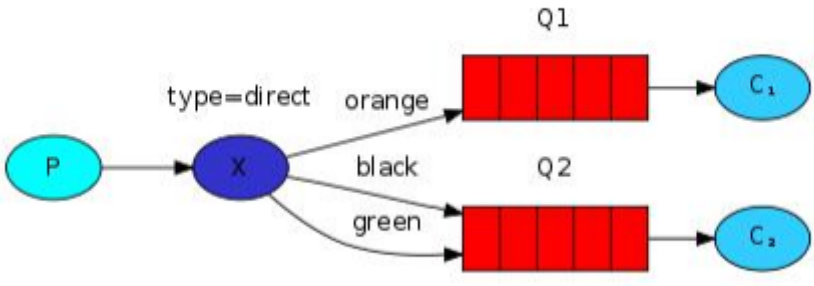
```
/**
 * @Author zhunc
 * @Date 2022/5/13 15:05
 */
public class ConsumerFanoutExchange {
    public static void main(String[] args) throws  Exception{
        Channel channel = RabbitMqUtils.getChannel();
        //声明交换机名称
        String exchangeName = "test_fanout_exchange";
        //声明路由key
        String routingKey = "";
        //交换机类型direct
        String exchangeType = "fanout";
        //队列名称
        String queueName = "test_fanout_queue";
        /**
         * 声明一个交换机
         * 参数明细
         * 1.exchange 交换机名称
         * 2.type 交换机类型 direct、topic、fanout、headers
         * 3.durable 是否需要持久化，true为持久化
```

```
         * 4.autoDelete 当最后一个绑定到Exchange上的队列删除后，自动删除改Exchange
         * 5.internal 当前Exchange是否用于内部使用、默认为false
         * 6.arguments 扩展参数、用于扩展AMQP协议自定化使用
         */
        channel.exchangeDeclare(exchangeName, exchangeType, true, false, false,
null);
        //声明一个队列
        channel.queueDeclare(queueName, false, false, false, null);
        //创建一个绑定关系
        channel.queueBind(queueName, exchangeName, routingKey);

        QueueingConsumer queueingConsumer = new QueueingConsumer(channel);

        //6.设置channel
        // 监听队列，第二个参数：是否自动进行消息确认。
        //参数：String queue, boolean autoAck, Consumer callback
        /**
         * 参数明细：
         * 1、queue 队列名称
         * 2、autoAck 自动回复，当消费者接收到消息后要告诉mq消息已接收，如果将此参数设置为
true表示会自动回复mq，如果设置为false要通过编程实现回复
         * 3、callback，消费方法，当消费者接收到消息要执行的方法
         */
        channel.basicConsume(queueName,true, queueingConsumer);
        while (true) {
            //7.获取消息
            QueueingConsumer.Delivery delivery =
queueingConsumer.nextDelivery();
            String msg = new String(delivery.getBody());
            System.out.println("消费端：" + msg);


        }
    }
}
```

四、routing路由模式

定义

1.消息生产者将消息发送给交换机按照路由判断,路由是字符串(info) 当前产生的消息携带路由字符(对象的方法),交换机
根据路由的key,只能匹配上路由key对应的消息队列,对应的消费者才能消费消息;
2.根据业务功能定义路由字符串
3.从系统的代码逻辑中获取对应的功能字符串,将消息任务扔到对应的队列中



业务场景：

error 通知;EXCEPTION;错误通知的功能;传统意义的错误通知;客户通知;利用key路由,可以将程序中的错误封装成消息传入到消息队列中,开发者可以自定义消费者,实时接收错误;

生产者

```
package com.znc.rabbitmqapi.exchange.direct;

import com.rabbitmq.client.Channel;
import com.znc.rabbitmqapi.utils.RabbitMqUtils;

/**
 * @Author zhunc
 * @Date 2022/5/13 15:04
 */
public class ProducerDirectExchange {
    public static void main(String[] args) throws Exception {
        Channel channel = RabbitMqUtils.getChannel();
        //声明交换机名称
        String exchangeName = "test_direct_exchange";
        //声明路由key
        String routingKey = "test_direct";
        for (int i = 0; i < 3; i++) {
            String message = i + ": hello world rabbitmq direct exchange
message";
            channel.basicPublish(exchangeName, routingKey, null,
message.getBytes());
        }

        RabbitMqUtils.close();
    }
}
```

消费者

```
package com.znc.rabbitmqapi.exchange.direct;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.QueueingConsumer;
import com.znc.rabbitmqapi.utils.RabbitMqUtils;
```

```java
/**
 * @Author zhunc
 * @Date 2022/5/13 15:05
 */
public class ConsumerDirectExchange {
    public static void main(String[] args) throws  Exception{
        Channel channel = RabbitMqUtils.getChannel();
        //声明交换机名称
        String exchangeName = "test_direct_exchange";
        //声明路由key
        String routingKey = "test_direct";
        //交换机类型direct
        String exchangeType = "direct";
        //队列名称
        String queueName = "test_direct_queue";
        /**
         * 声明一个交换机
         * 参数明细
         * 1.exchange 交换机名称
         * 2.type 交换机类型 direct、topic、fanout、headers
         * 3.durable 是否需要持久化，true为持久化
         * 4.autoDelete 当最后一个绑定到Exchange上的队列删除后，自动删除改Exchange
         * 5.internal 当前Exchange是否用于内部使用、默认为false
         * 6.arguments 扩展参数、用于扩展AMQP协议自定化使用
         */
        channel.exchangeDeclare(exchangeName, exchangeType, true, false, false,
null);
        //声明一个队列
        channel.queueDeclare(queueName, false, false, false, null);
        //创建一个绑定关系
        channel.queueBind(queueName, exchangeName, routingKey);

        QueueingConsumer queueingConsumer = new QueueingConsumer(channel);

        //6.设置channel
        // 监听队列，第二个参数：是否自动进行消息确认。
        //参数：String queue, boolean autoAck, Consumer callback
        /**
         * 参数明细：
         * 1、queue 队列名称
         * 2、autoAck 自动回复，当消费者接收到消息后要告诉mq消息已接收，如果将此参数设置为
true表示会自动回复mq，如果设置为false要通过编程实现回复
         * 3、callback，消费方法，当消费者接收到消息要执行的方法
         */
        channel.basicConsume(queueName,true, queueingConsumer);
        while (true) {
            //7.获取消息
            QueueingConsumer.Delivery delivery =
queueingConsumer.nextDelivery();
            String msg = new String(delivery.getBody());
            System.out.println("消费端：" + msg);


        }
    }
}
```

五、topic主题模式（路由模式的一种）

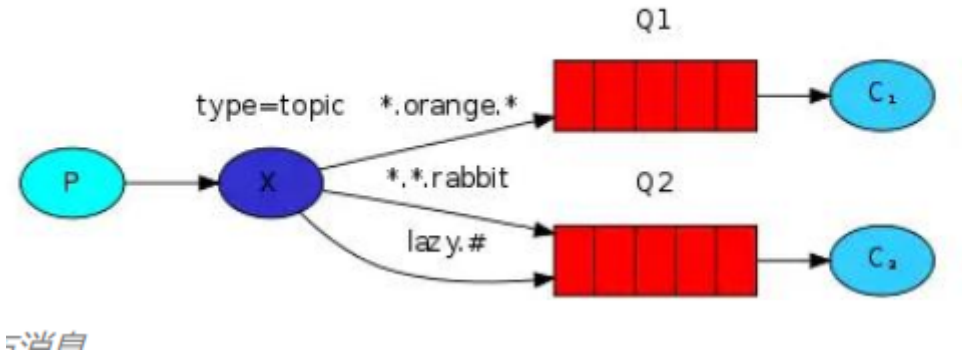
* # 代表通配符，* 代表一个单词,#代表零个或多个单词
路由功能添加模糊匹配
消息产生者产生消息,把消息交给交换机
交换机根据key的规则模糊匹配到对应的队列,由队列的监听消费者接收消息消费

发送时，指定topic交换机，指定routingKey

接收时，指定topic交换机，指定bindingkey为模糊匹配字符串

上面的路由模式是根据路由key进行完整的匹配（完全相等才发送消息），这里的通配符模式通俗的来讲就是模糊匹配。

符号“#”表示匹配一个或多个词，符号“*”表示匹配一个词。
　　与路由模式相似，但是，主题模式是一种模糊的匹配方式。



生产者

```
package com.znc.rabbitmqapi.exchange.topic;

import com.rabbitmq.client.Channel;
import com.znc.rabbitmqapi.utils.RabbitMqUtils;

/**
 * @Author zhunc
 * @Date 2022/5/13 15:04
```

```
 */
public class ProducerTopicExchange {
    public static void main(String[] args) throws Exception {
        Channel channel = RabbitMqUtils.getChannel();
        //声明交换机名称
        String exchangeName = "test_topic_exchange";
        //声明路由key
        String routingKey1 = "user.apple";
        String routingKey2 = "user.orange.yellow";
        String routingKey3 = "user.znc.sdf";
        String routingKey4 = "znc.user";

        String message = ": hello world rabbitmq topic exchange message";
        channel.basicPublish(exchangeName, routingKey1, null,
(routingKey1+message).getBytes());
        channel.basicPublish(exchangeName, routingKey2, null,
(routingKey2+message).getBytes());
        channel.basicPublish(exchangeName, routingKey3, null,
(routingKey3+message).getBytes());
        channel.basicPublish(exchangeName, routingKey4, null,
(routingKey4+message).getBytes());

        RabbitMqUtils.close();
    }
}
```

消费者

```
package com.znc.rabbitmqapi.exchange.topic;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.QueueingConsumer;
import com.znc.rabbitmqapi.utils.RabbitMqUtils;

/**
 * @Author zhunc
 * @Date 2022/5/13 15:05
 */
public class ConsumerTopicExchange {
    public static void main(String[] args) throws  Exception{
        Channel channel = RabbitMqUtils.getChannel();
        //声明交换机名称
        String exchangeName = "test_topic_exchange";
        //声明路由key
        String routingKey = "user.*";
        //交换机类型direct
        String exchangeType = "topic";
        //队列名称
        String queueName = "test_topic_queue";
        /**
         * 声明一个交换机
         * 参数明细
         * 1.exchange 交换机名称
         * 2.type 交换机类型 direct、topic、fanout、headers
```

```
         * 3.durable 是否需要持久化，true为持久化
         * 4.autoDelete 当最后一个绑定到Exchange上的队列删除后，自动删除改Exchange
         * 5.internal 当前Exchange是否用于内部使用、默认为false
         * 6.arguments 扩展参数、用于扩展AMQP协议自定化使用
         */
        channel.exchangeDeclare(exchangeName, exchangeType, true, false, false,
null);
        //声明一个队列
        channel.queueDeclare(queueName, false, false, false, null);
        //创建一个绑定关系
        channel.queueBind(queueName, exchangeName, routingKey);

        QueueingConsumer queueingConsumer = new QueueingConsumer(channel);

        //6.设置channel
        // 监听队列，第二个参数：是否自动进行消息确认。
        //参数：String queue, boolean autoAck, Consumer callback
        /**
         * 参数明细：
         * 1、queue 队列名称
         * 2、autoAck 自动回复，当消费者接收到消息后要告诉mq消息已接收，如果将此参数设置为
true表示会自动回复mq，如果设置为false要通过编程实现回复
         * 3、callback，消费方法，当消费者接收到消息要执行的方法
         */
        channel.basicConsume(queueName,true, queueingConsumer);
        while (true) {
            //7.获取消息

            QueueingConsumer.Delivery delivery =
queueingConsumer.nextDelivery();
            String msg = new String(delivery.getBody());
            System.out.println("消费端：" + msg);


        }
    }
}
```

Exchange的类型

Exchange 主要用于将消息发送到指定的队列。
Exchange有多个种类，常用的有direct，fanout，topic。前三种类似集合对应关系那样，（direct）1:1,（fanout）1：N,（topic）N:1
direct： 1:1类似完全匹配
fanout：1：N 可以把一个消息并行发布到多个队列上去，简单的说就是，当多个队列绑定到fanout的交换器,那么交换器一次性拷贝多个消息分别发送到绑定的队列上，每个队列有这个消息的副本。
topic N:1 ，多个交换器可以路由消息到同一个队列。根据模糊匹配，比如一个队列的routing key 为*.test ，那么凡是到达交换器的消息中的routing key 后缀.test都被路由到这个队列上。