# 1. 安装

## 1.1 Erlang与RabbitMQ对应版本

<https://www.rabbitmq.com/which-erlang.html>

## 1.2 Erlang安装

### 1.2.1 源码安装

环境准备：

yum install gcc glibc-devel make ncurses-devel openssl-devel autoconf

yum install unixODBC unixODBC-devel

Erlang源码下载：<http://erlang.org/download/otp_src_22.3.tar.gz>

tar -zxvf otp\_src\_22.3.tar.gz

cd otp\_src\_22.3

./configure --prefix=/usr/local/erlang --enable-hipe --enable-threads --enable-smp-support --enable-kernel-poll --without-javac

make && make install

测试是否安装成功：/usr/local/erlang/bin/erl

配置到bin目录：把/usr/local/erlang/bin配置到PATH去

### 1.2.2 rpm安装（推荐）

安装依赖socat：yum -y install socat

erlang的rpm包下载，并上传到linux系统下以作准备：<https://packages.erlang-solutions.com/erlang/rpm/centos/7/x86_64/esl-erlang_22.3-1~centos~7_amd64.rpm>

下载rpm仓库：wget <http://packages.erlang-solutions.com/erlang-solutions-1.0-1.noarch.rpm>

更新rpm仓库（不用此步骤）：rpm -Uvh erlang-solutions-1.0-1.noarch.rpm

安装依赖epel-release：yum -y install epel-release

安装erlang：yum -y install esl-erlang\_22.3-1\_centos\_7\_amd64.rpm

rpm -qa|grep erlang //查看erlang是否安装成功

## 1.3 RabbitMQ安装

### 1.3.1 解压包安装

tar -xvf rabbitmq-server-generic-unix-3.8.3.tar.xz -C /usr/local/

cp /usr/local/rabbitmq\_server-3.8.3/ebin/rabbit.app /usr/local/rabbitmq\_server-3.8.3/ebin/rabbit.app.bak

vim /usr/local/rabbitmq\_server-3.8.3/ebin/rabbit.app //把loopback\_users里的<<"guest">>删除，以便在外网使用guest:guest登录

:wq

/usr/local/rabbitmq\_server-3.8.3/sbin/rabbitmq-plugins list //查看插件列表

/usr/local/rabbitmq\_server-3.8.3/sbin/rabbitmq-plugins enable rabbitmq\_management //打开web插件

/usr/local/rabbitmq\_server-3.8.3/sbin/rabbitmq-server start -detached //加-detached参数后台启动

ps -ef|grep rabbitmq //检查是否有该进程

防火墙放行15672端口，外网访问：<http://ip:15672>，用户名密码均为guest

HTTP的端口：15672

AMQP的端口：5672

clustering的端口：25672

### 1.3.2 rpm安装（推荐）

注意：对应erlang的rpm安装

rabbitmq的rpm包下载并上传到linux：<https://dl.bintray.com/rabbitmq/all/rabbitmq-server/3.8.3/rabbitmq-server-3.8.3-1.el7.noarch.rpm>

安装：rpm -ivh rabbitmq-server-3.8.3-1.el7.noarch.rpm

查看是否安装成功：rpm -qa|grep rabbitmq

备份配置文件：cp /usr/lib/rabbitmq/lib/rabbitmq\_server-3.8.3/ebin/rabbit.app /usr/lib/rabbitmq/lib/rabbitmq\_server-3.8.3/ebin/rabbit.app.bak

vim /usr/lib/rabbitmq/lib/rabbitmq\_server-3.8.3/ebin/rabbit.app //把loopback\_users里的<<"guest">>删除，以便在外网使用guest:guest登录

/usr/lib/rabbitmq/lib/rabbitmq\_server-3.8.3/sbin/rabbitmq-plugins enable rabbitmq\_management //打开web插件

防火墙开放：15672和5672端口

启动：systemctl start rabbitmq-server

状态：systemctl status rabbitmq-server

停止：systemctl stop rabbitmq-server

自启动：

查看是否开启自启动：systemctl list-unit-files|grep rabbitmq

打开自启动：systemctl enable rabbitmq-server.service

关闭自启动：systemctl disable rabbitmq-server.service

# 2. 结合Java使用

## 2.1 配置

创建用户、虚拟主机（一般以/开头）以及设置虚拟主机的权限为该用户

## 2.2 使用

注：如果Java服务与MQ服务器不同，务必打开MQ所在服务器的5672端口

### 2.2.1 项目依赖

|  |
| --- |
| <dependencies>  <dependency>  <groupId>com.rabbitmq</groupId>  <artifactId>amqp-client</artifactId>  <version>5.8.0</version>  </dependency> </dependencies> |

### 2.2.2 连接工具类

|  |
| --- |
| package com.he.rabbitmq.util;  import com.rabbitmq.client.Connection; import com.rabbitmq.client.ConnectionFactory;  import java.io.IOException; import java.util.concurrent.TimeoutException;  /\*\*  \* RabbitMQ连接工具类  \*/ public class ConnectionUtils {   /\*\*  \* 获取连接  \* @return  \* @throws IOException  \* @throws TimeoutException  \*/  public static Connection getConnection() throws IOException, TimeoutException {   //创建连接工厂  ConnectionFactory factory = new ConnectionFactory();   //设置服务器地址  factory.setHost("192.168.199.197");   //设置虚拟主机，类似msql的db  factory.setVirtualHost("/he");   //设置AMPQ协议的端口号  factory.setPort(5672);   //设置用户名  factory.setUsername("he");   //设置密码  factory.setPassword("123");   return factory.newConnection();  }  } |

### 2.2.3 队列模型

#### （1）简单模型

##### a. 模型图

https://www.rabbitmq.com/img/tutorials/python-one.png

##### b. 生产者

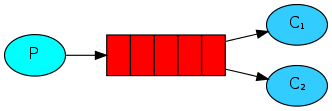
|  |
| --- |
| package com.he.rabbitmq.simple;  import com.he.rabbitmq.util.ConnectionUtils; import com.rabbitmq.client.Channel; import com.rabbitmq.client.Connection;  import java.io.IOException; import java.util.concurrent.TimeoutException;  /\*\*  \* 简单队列生产者  \*/ public class Producer {   /\*\*  \* 队列名称  \*/  private static final String QUEUE\_NAME = "test\_simple\_queue";   public static void main(String[] args) throws IOException, TimeoutException {   //获取连接  Connection connection = ConnectionUtils.getConnection();   //使用连接创建一个通道  Channel channel = connection.createChannel();   //使用通道进行队列声明  channel.queueDeclare(QUEUE\_NAME, false, false, false, null);   //使用通道进行消息发送  String msg = "hello simple queue";  channel.basicPublish("", QUEUE\_NAME, null, msg.getBytes());  System.out.println("消息【" + msg + "】发送成功");   //关闭通道  channel.close();   //关闭连接  connection.close();   } } |

##### c. 消费者

|  |
| --- |
| package com.he.rabbitmq.simple;  import com.he.rabbitmq.util.ConnectionUtils; import com.rabbitmq.client.\*;  import java.io.IOException; import java.util.concurrent.TimeoutException;  /\*\*  \* 简单队列消费者，消费者不需要关闭资源  \*/ public class Consumer {   /\*\*  \* 队列名称  \*/  private static final String QUEUE\_NAME = "test\_simple\_queue";   public static void main(String[] args) throws IOException, TimeoutException {   //获取连接  Connection connection = ConnectionUtils.getConnection();   //使用连接创建一个通道  Channel channel = connection.createChannel();   //使用通道声明队列  channel.queueDeclare(QUEUE\_NAME, false, false, false, null);   //创建消费者，并重写handleDelivery方法  DefaultConsumer consumer = new DefaultConsumer(channel) {  @Override  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {  System.out.println("消息【" + new String(body) + "】已被消费");  }  };   //监听队列  channel.basicConsume(QUEUE\_NAME, true, consumer);   System.out.println("===监听队列已就位===");  } } |

#### （2）工作模型

##### i. 模型图



##### ii. 轮询分发

###### a. 生产者

|  |
| --- |
| package com.he.rabbitmq.work.roundrobin;  import com.he.rabbitmq.util.ConnectionUtils; import com.rabbitmq.client.Channel; import com.rabbitmq.client.Connection;  import java.io.IOException; import java.util.concurrent.TimeoutException;  /\*\*  \* 工作队列生产者-轮询分发  \*/ public class Producer {   /\*\*  \* 队列名称  \*/  private static final String QUEUE\_NAME = "test\_work\_queue";   public static void main(String[] args) throws IOException, TimeoutException {   //获取连接  Connection connection = ConnectionUtils.getConnection();   //使用连接创建一个通道  Channel channel = connection.createChannel();   //使用通道进行队列声明  channel.queueDeclare(QUEUE\_NAME, false, false, false, null);   //使用通道进行消息发送  for (int i = 1; i <= 50; i ++) {  String msg = "hello work queue--" + i;  channel.basicPublish("", QUEUE\_NAME, null, msg.getBytes());   try {  Thread.sleep(i\*20);  } catch (InterruptedException e) {  e.printStackTrace();  } finally {  System.out.println("消息【" + msg + "】发送成功");  }  }   //关闭通道  channel.close();   //关闭连接  connection.close();  } } |

###### b. 消费者01

|  |
| --- |
| package com.he.rabbitmq.work.roundrobin;  import com.he.rabbitmq.util.ConnectionUtils; import com.rabbitmq.client.\*;  import java.io.IOException; import java.util.concurrent.TimeoutException;  /\*\*  \* 工作队列消费者01-轮询分发  \*/ public class Consumer01 {   /\*\*  \* 队列名称  \*/  private static final String QUEUE\_NAME = "test\_work\_queue";   public static void main(String[] args) throws IOException, TimeoutException {   //获取连接  Connection connection = ConnectionUtils.getConnection();   //使用连接创建一个通道  Channel channel = connection.createChannel();   //使用通道声明队列  channel.queueDeclare(QUEUE\_NAME, false, false, false, null);   //创建消费者，并重写handleDelivery方法  DefaultConsumer consumer = new DefaultConsumer(channel) {  @Override  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {   //消费者01模拟需要2s来处理消息  try {  Thread.sleep(2000);  } catch (InterruptedException e) {  e.printStackTrace();  }   System.out.println("消息【" + new String(body) + "】已被消费者01消费");  }  };   //监听队列  channel.basicConsume(QUEUE\_NAME, true, consumer);   System.out.println("===消费者01监听队列已就位===");  } } |

###### c. 消费者02

|  |
| --- |
| package com.he.rabbitmq.work.roundrobin;  import com.he.rabbitmq.util.ConnectionUtils; import com.rabbitmq.client.\*;  import java.io.IOException; import java.util.concurrent.TimeoutException;  /\*\*  \* 工作队列消费者02-轮询分发  \*/ public class Consumer02 {   /\*\*  \* 队列名称  \*/  private static final String QUEUE\_NAME = "test\_work\_queue";   public static void main(String[] args) throws IOException, TimeoutException {   //获取连接  Connection connection = ConnectionUtils.getConnection();   //使用连接创建一个通道  Channel channel = connection.createChannel();   //使用通道声明队列  channel.queueDeclare(QUEUE\_NAME, false, false, false, null);   //创建消费者，并重写handleDelivery方法  DefaultConsumer consumer = new DefaultConsumer(channel) {  @Override  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {   //消费者02模拟需要1s来处理消息  try {  Thread.sleep(1000);  } catch (InterruptedException e) {  e.printStackTrace();  }   System.out.println("消息【" + new String(body) + "】已被消费者02消费");  }  };   //监听队列  channel.basicConsume(QUEUE\_NAME, true, consumer);   System.out.println("===消费者02监听队列已就位===");  } } |

##### iii. 公平分发

###### 生产者

|  |
| --- |
| package com.he.rabbitmq.work.fairdispatch;  import com.he.rabbitmq.util.ConnectionUtils; import com.rabbitmq.client.Channel; import com.rabbitmq.client.Connection;  import java.io.IOException; import java.util.concurrent.TimeoutException;  /\*\*  \* 工作队列生产者-公平分发  \*/ public class Producer {   /\*\*  \* 队列名称  \*/  private static final String QUEUE\_NAME = "test\_work\_queue";   public static void main(String[] args) throws IOException, TimeoutException {   //获取连接  Connection connection = ConnectionUtils.getConnection();   //使用连接创建一个通道  Channel channel = connection.createChannel();   //使用通道进行队列声明  channel.queueDeclare(QUEUE\_NAME, false, false, false, null);   //每次只发一条消息给消费者，需要消费者手动确认  channel.basicQos(1);   //使用通道进行消息发送  for (int i = 1; i <= 50; i ++) {  String msg = "hello work queue--" + i;  channel.basicPublish("", QUEUE\_NAME, null, msg.getBytes());   try {  Thread.sleep(i\*5);  } catch (InterruptedException e) {  e.printStackTrace();  } finally {  System.out.println("消息【" + msg + "】发送成功");  }  }   //关闭通道  channel.close();   //关闭连接  connection.close();  } } |

###### 消费者01

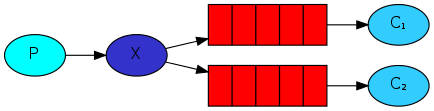
|  |
| --- |
| package com.he.rabbitmq.work.fairdispatch;  import com.he.rabbitmq.util.ConnectionUtils; import com.rabbitmq.client.\*;  import java.io.IOException; import java.util.concurrent.TimeoutException;  /\*\*  \* 工作队列消费者01-公平分发  \*/ public class Consumer01 {   /\*\*  \* 队列名称  \*/  private static final String QUEUE\_NAME = "test\_work\_queue";   public static void main(String[] args) throws IOException, TimeoutException {   //获取连接  Connection connection = ConnectionUtils.getConnection();   //使用连接创建一个通道  final Channel channel = connection.createChannel();   //使用通道声明队列  channel.queueDeclare(QUEUE\_NAME, false, false, false, null);   //每次只发一条消息给消费者，需要消费者手动确认  channel.basicQos(1);   //创建消费者，并重写handleDelivery方法  DefaultConsumer consumer = new DefaultConsumer(channel) {  @Override  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {   //消费者01模拟需要2s来处理消息  try {  Thread.sleep(2000);  } catch (InterruptedException e) {  e.printStackTrace();  }   channel.basicAck(envelope.getDeliveryTag(), false);   System.out.println("消息【" + new String(body) + "】已被消费者01消费");  }  };   //监听队列  channel.basicConsume(QUEUE\_NAME, false, consumer);   System.out.println("===消费者01监听队列已就位===");  } } |

###### 消费者02

|  |
| --- |
| package com.he.rabbitmq.work.fairdispatch;  import com.he.rabbitmq.util.ConnectionUtils; import com.rabbitmq.client.\*;  import java.io.IOException; import java.util.concurrent.TimeoutException;  /\*\*  \* 工作队列消费者02-公平分发  \*/ public class Consumer02 {   /\*\*  \* 队列名称  \*/  private static final String QUEUE\_NAME = "test\_work\_queue";   public static void main(String[] args) throws IOException, TimeoutException {   //获取连接  Connection connection = ConnectionUtils.getConnection();   //使用连接创建一个通道  final Channel channel = connection.createChannel();   //使用通道声明队列  channel.queueDeclare(QUEUE\_NAME, false, false, false, null);   //每次只发一条消息给消费者，需要消费者手动确认  channel.basicQos(1);   //创建消费者，并重写handleDelivery方法  DefaultConsumer consumer = new DefaultConsumer(channel) {  @Override  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {   //消费者02模拟需要1s来处理消息  try {  Thread.sleep(1000);  } catch (InterruptedException e) {  e.printStackTrace();  }   channel.basicAck(envelope.getDeliveryTag(), false);   System.out.println("消息【" + new String(body) + "】已被消费者02消费");  }  };   //监听队列  channel.basicConsume(QUEUE\_NAME, false, consumer);   System.out.println("===消费者02监听队列已就位===");  } } |

#### （3）发布/订阅模型

##### a. 模型图



##### b. 生产者

|  |
| --- |
| package com.he.rabbitmq.ps;  import com.he.rabbitmq.util.ConnectionUtils; import com.rabbitmq.client.Channel; import com.rabbitmq.client.Connection;  import java.io.IOException; import java.util.concurrent.TimeoutException;  /\*\*  \* 发布订阅模式的生产者  \* 一个生产者，一个交换机，多个队列，多个消费者，队列绑定到交换机上，往交换机发送消息  \*/ public class Producer {   /\*\*  \* 交换机名称  \*/  private static final String EXCHANGE\_NAME = "test\_exchange\_fanout";   public static void main(String[] args) throws IOException, TimeoutException {   //获取连接  Connection connection = ConnectionUtils.getConnection();   //使用连接创建一个通道  Channel channel = connection.createChannel();   //使用通道进行交换机声明  channel.exchangeDeclare(EXCHANGE\_NAME, "fanout", false, false, null);   //使用通道进行消息发送，由于队列还没绑定交换机，所以此时消息会丢失  String msg = "hello exchange";  channel.basicPublish(EXCHANGE\_NAME, "", null, msg.getBytes());  System.out.println("消息【" + msg + "】发送成功");   //关闭通道  channel.close();   //关闭连接  connection.close();   } } |

##### c. 消费者01

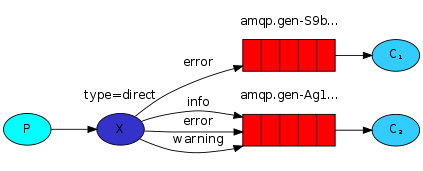
|  |
| --- |
| package com.he.rabbitmq.ps;  import com.he.rabbitmq.util.ConnectionUtils; import com.rabbitmq.client.\*;  import java.io.IOException; import java.util.concurrent.TimeoutException;  /\*\*  \* 发布订阅模式的消费者01  \*/ public class Consumer01 {   /\*\*  \* 队列名称  \*/  private static final String QUEUE\_NAME = "test\_exchange\_mail";   /\*\*  \* 交换机名称  \*/  private static final String EXCHANGE\_NAME = "test\_exchange\_fanout";   public static void main(String[] args) throws IOException, TimeoutException {   //获取连接  Connection connection = ConnectionUtils.getConnection();   //使用连接创建一个通道  final Channel channel = connection.createChannel();   //使用通道进行交换机声明  channel.exchangeDeclare(EXCHANGE\_NAME, "fanout", false, false, null);   //使用通道声明队列  channel.queueDeclare(QUEUE\_NAME, false, false, false, null);   //绑定队列到交换机上  channel.queueBind(QUEUE\_NAME, EXCHANGE\_NAME, "");   //每次只发送一条消息到消费者  channel.basicQos(1);   DefaultConsumer consumer = new DefaultConsumer(channel) {  @Override  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {   System.out.println("消息【" + new String(body) + "】已被mail消费");   //手动消息确认  channel.basicAck(envelope.getDeliveryTag(), false);  }  };   //消息监听，消息应答方式为false  channel.basicConsume(QUEUE\_NAME, false, consumer);   System.out.println("===消费者mail监听队列已就位===");  } } |

##### d. 消费者02

|  |
| --- |
| package com.he.rabbitmq.ps;  import com.he.rabbitmq.util.ConnectionUtils; import com.rabbitmq.client.\*;  import java.io.IOException; import java.util.concurrent.TimeoutException;  /\*\*  \* 发布订阅模式的消费者02  \*/ public class Consumer02 {   /\*\*  \* 队列名称  \*/  private static final String QUEUE\_NAME = "test\_exchange\_sms";   /\*\*  \* 交换机名称  \*/  private static final String EXCHANGE\_NAME = "test\_exchange\_fanout";   public static void main(String[] args) throws IOException, TimeoutException {   //获取连接  Connection connection = ConnectionUtils.getConnection();   //使用连接创建一个通道  final Channel channel = connection.createChannel();   //使用通道进行交换机声明  channel.exchangeDeclare(EXCHANGE\_NAME, "fanout", false, false, null);   //使用通道声明队列  channel.queueDeclare(QUEUE\_NAME, false, false, false, null);   //绑定队列到交换机上  channel.queueBind(QUEUE\_NAME, EXCHANGE\_NAME, "");   //每次只发送一条消息到消费者  channel.basicQos(1);   DefaultConsumer consumer = new DefaultConsumer(channel) {  @Override  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {   System.out.println("消息【" + new String(body) + "】已被sms消费");   //手动消息确认  channel.basicAck(envelope.getDeliveryTag(), false);  }  };   //消息监听，消息应答方式为false  channel.basicConsume(QUEUE\_NAME, false, consumer);   System.out.println("===消费者sms监听队列已就位===");  } } |

#### （4）路由模型

##### a. 模型图



##### b. 生产者

|  |
| --- |
| package com.he.rabbitmq.routing;  import com.he.rabbitmq.util.ConnectionUtils; import com.rabbitmq.client.Channel; import com.rabbitmq.client.Connection;  import java.io.IOException; import java.util.concurrent.TimeoutException;  /\*\*  \* 路由模式-生产者，exchange type 必须为direct  \*/ public class Producer {   /\*\*  \* 交换机名称  \*/  public static final String EXCHANGE\_NAME = "test\_routing\_exchange";   public static void main(String[] args) throws IOException, TimeoutException {   //获取连接  Connection connection = ConnectionUtils.getConnection();   //创建通道  Channel channel = connection.createChannel();   //声明一个类型为direct的交换机  channel.exchangeDeclare(EXCHANGE\_NAME, "direct");   //每次只给消费者分发一条消息  channel.basicQos(1);   //生产消息  for (int i = 1; i <= 10; i ++) {  String routingKey = i%2 == 0 ? "info" : "error";  String msg = "hello routing!" + routingKey + i;  channel.basicPublish(EXCHANGE\_NAME, routingKey, null, msg.getBytes());  System.out.println("消息【" + msg + "】发送成功");  }   //关闭通道  channel.close();   //关闭连接  connection.close();  } } |

##### c. 消费者01

|  |
| --- |
| package com.he.rabbitmq.routing;  import com.he.rabbitmq.util.ConnectionUtils; import com.rabbitmq.client.\*;  import java.io.IOException; import java.util.concurrent.CopyOnWriteArrayList; import java.util.concurrent.TimeoutException;  /\*\*  \* 路由模式-消费者01，exchange type 必须为direct  \*/ public class Consumer01 {   /\*\*  \* 交换机名称  \*/  public static final String EXCHANGE\_NAME = "test\_routing\_exchange";   /\*\*  \* 队列名称  \*/  public static final String QUEUE\_NAME = "test\_routing\_queue01";   public static void main(String[] args) throws IOException, TimeoutException {   //获取连接  Connection connection = ConnectionUtils.getConnection();   //创建通道  final Channel channel = connection.createChannel();   //声明一个类型为direct的交换机  channel.exchangeDeclare(EXCHANGE\_NAME, "direct");   //声明队列  channel.queueDeclare(QUEUE\_NAME, true, false, false, null);   //队列绑定，以routingKey进行消息区分  CopyOnWriteArrayList<String> routingKeys = new CopyOnWriteArrayList<String>();  routingKeys.add("error");  for (String routingKey : routingKeys) {  channel.queueBind(QUEUE\_NAME, EXCHANGE\_NAME, routingKey);  }   //每次只给消费者分发一条消息  channel.basicQos(1);   //创建消费者，并重写消费者的handleDelivery方法  DefaultConsumer consumer = new DefaultConsumer(channel) {  @Override  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {   System.out.println("Routing\_Consumer01: " + new String(body));   //消息应答  channel.basicAck(envelope.getDeliveryTag(), false);  }  };   //消息监听  channel.basicConsume(QUEUE\_NAME, false, consumer);   System.out.println("===Routing\_Consumer01监听队列已就位===");  } } |

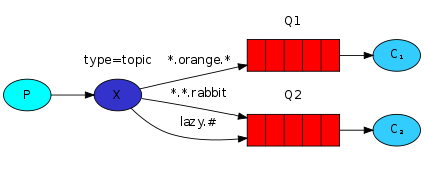
##### d. 消费者02

|  |
| --- |
| package com.he.rabbitmq.routing;  import com.he.rabbitmq.util.ConnectionUtils; import com.rabbitmq.client.\*;  import java.io.IOException; import java.util.concurrent.CopyOnWriteArrayList; import java.util.concurrent.TimeoutException;  /\*\*  \* 路由模式-消费者02，exchange type 必须为direct  \*/ public class Consumer02 {   /\*\*  \* 交换机名称  \*/  public static final String EXCHANGE\_NAME = "test\_routing\_exchange";   /\*\*  \* 队列名称  \*/  public static final String QUEUE\_NAME = "test\_routing\_queue02";   public static void main(String[] args) throws IOException, TimeoutException {   //获取连接  Connection connection = ConnectionUtils.getConnection();   //创建通道  final Channel channel = connection.createChannel();   //声明一个类型为direct的交换机  channel.exchangeDeclare(EXCHANGE\_NAME, "direct");   //声明队列  channel.queueDeclare(QUEUE\_NAME, true, false, false, null);   //队列绑定，以routingKey进行消息区分  CopyOnWriteArrayList<String> routingKeys = new CopyOnWriteArrayList<String>();  routingKeys.add("error");  routingKeys.add("info");  for (String routingKey : routingKeys) {  channel.queueBind(QUEUE\_NAME, EXCHANGE\_NAME, routingKey);  }   //每次只给消费者分发一条消息  channel.basicQos(1);   //创建消费者，并重写消费者的handleDelivery方法  DefaultConsumer consumer = new DefaultConsumer(channel) {  @Override  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {   System.out.println("Routing\_Consumer02: " + new String(body));   //消息应答  channel.basicAck(envelope.getDeliveryTag(), false);  }  };   //消息监听  channel.basicConsume(QUEUE\_NAME, false, consumer);   System.out.println("===Routing\_Consumer02监听队列已就位===");  } } |

#### （5）主题模型

##### a. 模型图

通配符：#匹配多个英文（用.隔开），\*匹配一个英文



##### b. 生产者

|  |
| --- |
| package com.he.rabbitmq.topic;  import com.he.rabbitmq.util.ConnectionUtils; import com.rabbitmq.client.Channel; import com.rabbitmq.client.Connection;  import java.io.IOException; import java.util.concurrent.TimeoutException;  /\*\*  \* 主题模式-生产者，exchange type 必须为topic  \*/ public class Producer {   /\*\*  \* 交换机名称  \*/  public static final String EXCHANGE\_NAME = "test\_topic\_exchange";   public static void main(String[] args) throws IOException, TimeoutException {   //获取连接  Connection connection = ConnectionUtils.getConnection();   //创建通道  Channel channel = connection.createChannel();   //声明一个类型为direct的交换机  channel.exchangeDeclare(EXCHANGE\_NAME, "topic");   //每次只给消费者分发一条消息  channel.basicQos(1);   //生产消息  for (int i = 1; i <= 10; i ++) {  String routingKey = i%2 == 0 ? "goods.delete" : "goods.add";  String msg = "hello topic!" + routingKey + i;  channel.basicPublish(EXCHANGE\_NAME, routingKey, null, msg.getBytes());  System.out.println("消息【" + msg + "】发送成功");  }   //关闭通道  channel.close();   //关闭连接  connection.close();  } } |

##### c. 消费者01

|  |
| --- |
| package com.he.rabbitmq.topic;  import com.he.rabbitmq.util.ConnectionUtils; import com.rabbitmq.client.\*;  import java.io.IOException; import java.util.concurrent.CopyOnWriteArrayList; import java.util.concurrent.TimeoutException;  /\*\*  \* 主题模式-消费者01，exchange type 必须为topic  \*/ public class Consumer01 {   /\*\*  \* 交换机名称  \*/  public static final String EXCHANGE\_NAME = "test\_topic\_exchange";   /\*\*  \* 队列名称  \*/  public static final String QUEUE\_NAME = "test\_topic\_queue01";   public static void main(String[] args) throws IOException, TimeoutException {   //获取连接  Connection connection = ConnectionUtils.getConnection();   //创建通道  final Channel channel = connection.createChannel();   //声明一个类型为direct的交换机  channel.exchangeDeclare(EXCHANGE\_NAME, "topic");   //声明队列  channel.queueDeclare(QUEUE\_NAME, true, false, false, null);   //队列绑定，以routingKey进行消息区分  CopyOnWriteArrayList<String> routingKeys = new CopyOnWriteArrayList<String>();  routingKeys.add("goods.add");  for (String routingKey : routingKeys) {  channel.queueBind(QUEUE\_NAME, EXCHANGE\_NAME, routingKey);  }   //每次只给消费者分发一条消息  channel.basicQos(1);   //创建消费者，并重写消费者的handleDelivery方法  DefaultConsumer consumer = new DefaultConsumer(channel) {  @Override  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {   System.out.println("Topic\_Consumer01: " + new String(body));   //消息应答  channel.basicAck(envelope.getDeliveryTag(), false);  }  };   //消息监听  channel.basicConsume(QUEUE\_NAME, false, consumer);   System.out.println("===Topic\_Consumer01监听队列已就位===");  } } |

##### d. 消费者02

|  |
| --- |
| package com.he.rabbitmq.topic;  import com.he.rabbitmq.util.ConnectionUtils; import com.rabbitmq.client.\*;  import java.io.IOException; import java.util.concurrent.CopyOnWriteArrayList; import java.util.concurrent.TimeoutException;  /\*\*  \* 主题模式-消费者02，exchange type 必须为topic  \*/ public class Consumer02 {   /\*\*  \* 交换机名称  \*/  public static final String EXCHANGE\_NAME = "test\_topic\_exchange";   /\*\*  \* 队列名称  \*/  public static final String QUEUE\_NAME = "test\_topic\_queue02";   public static void main(String[] args) throws IOException, TimeoutException {   //获取连接  Connection connection = ConnectionUtils.getConnection();   //创建通道  final Channel channel = connection.createChannel();   //声明一个类型为direct的交换机  channel.exchangeDeclare(EXCHANGE\_NAME, "topic");   //声明队列  channel.queueDeclare(QUEUE\_NAME, true, false, false, null);   //队列绑定，以routingKey进行消息区分，通配符#会匹配前缀为goods.的所有消息  CopyOnWriteArrayList<String> routingKeys = new CopyOnWriteArrayList<String>();  routingKeys.add("goods.#");  for (String routingKey : routingKeys) {  channel.queueBind(QUEUE\_NAME, EXCHANGE\_NAME, routingKey);  }   //每次只给消费者分发一条消息  channel.basicQos(1);   //创建消费者，并重写消费者的handleDelivery方法  DefaultConsumer consumer = new DefaultConsumer(channel) {  @Override  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {   System.out.println("Topic\_Consumer02: " + new String(body));   //消息应答  channel.basicAck(envelope.getDeliveryTag(), false);  }  };   //消息监听  channel.basicConsume(QUEUE\_NAME, false, consumer);   System.out.println("===Topic\_Consumer02监听队列已就位===");  } } |

### 2.2.4 消息应答与持久化

#### （1）消息应答

boolean autoAck = true;  
channel.basicConsume(QUEUE\_NAME, autoAck, consumer);

当autoAck为true时，采取自动应答，也就是当rabbitmq把消息发送给消费者后，就会从内存中删除该消息，此时，假若消费者宕机，则该消息就找不回了。当autoAck为false时，在消费者处理完消息后，需手动发送消息确认，尽管消费者宕机，也会把该消息交给下一个消费者处理

#### （2）持久化

在队列声明时：

boolean durable = false;  
channel.queueDeclare(QUEUE\_NAME, durable, false, false, null);

当durable为false时，假如rabbitmq宕机，则内存里的所有消息都将丢失。当durable为true时，会把消息持久化到硬盘，尽管rabbitmq服务器宕机，数据也能找回。

注意：持久化只有在声明时指定，不能后续修改。

### 2.2.5 消费端限流与消息重回队列

#### （1）消费端限流

channel.basicQos(1); //前提：必须为手动ack机制，消费者应答前，不处理其他消息，每次只处理一条消息，减轻消费者服务器压力

#### （2）消息重回队列

boolean requeue = false;  
channel.basicNack(envelope.getDeliveryTag(), false, requeue);

一般设置为false，如果设置为true，则在队列末尾重新回到队列

### 2.2.6 Exchange类型

如果不指定exchange名称，例如：

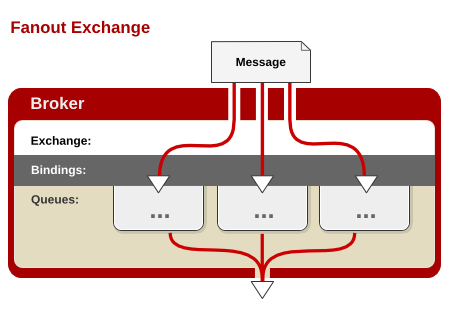
String exchange = “”;

channel.basicPublish(exchange, QUEUE\_NAME, null, msg.getBytes());

则为匿名分发。

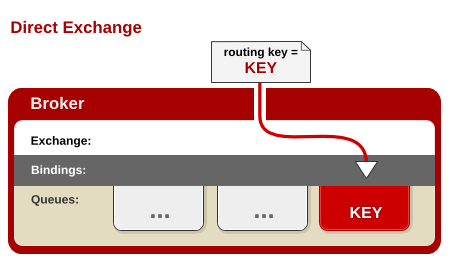
#### fanout

不处理路由键：只需要把队列绑定到交换机上，每条发送到该交换机的消息，都会转发到与该交换机绑定的队列上。交换机转发消息速度最快。



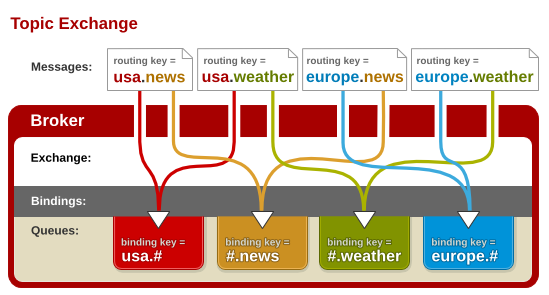
#### direct

处理路由键：消息发送以及把队列绑定到交换机时，都需要指定routingKey，每条发送到该交换机指定routingKey的消息，都会转发到绑定队列时指定该routingKey的队列上。要求该消息与一个特定的路由键完全匹配。



#### topic

通配符：#匹配多个英文（用.隔开），\*匹配一个英文



### 2.2.7 消息确认机制

#### （1）事务机制

在消费者发送消息到mq的代码前后，使用txSelect/txCommit/txRollback，每一次commit都是一次请求，因此，降低了RabbitMQ的吞吐量。

以为简单模型为例：

##### a. 生产者

|  |
| --- |
| package com.he.rabbitmq.tx;  import com.he.rabbitmq.util.ConnectionUtils; import com.rabbitmq.client.Channel; import com.rabbitmq.client.Connection;  import java.io.IOException; import java.util.concurrent.TimeoutException;  /\*\*  \* 消息确认机制之事务机制-生产者  \*/ public class TxProducer {   public static final String QUEUE\_NAME = "test\_tx\_queue";   public static void main(String[] args) throws IOException, TimeoutException {   Connection connection = ConnectionUtils.getConnection();   Channel channel = connection.createChannel();   channel.queueDeclare(QUEUE\_NAME, false, false, false, null);   String msg = "hello tx";   //开启事务  channel.txSelect();  try {  channel.basicPublish("", QUEUE\_NAME, null, msg.getBytes());   int i = 1/0;   //提交事务  channel.txCommit();   System.out.println("消息【" + msg + "】发送成功");  } catch (Exception e) {   //回滚事务  channel.txRollback();  System.out.println("消息【" + msg + "】发送失败，回滚！！！");  }   //关闭通道  channel.close();   //关闭连接  connection.close();  } } |

##### b. 消费者（不用变）

|  |
| --- |
| package com.he.rabbitmq.tx;  import com.he.rabbitmq.util.ConnectionUtils; import com.rabbitmq.client.\*;  import java.io.IOException; import java.util.concurrent.TimeoutException;  /\*\*  \* 消息确认机制之事务机制-消费者  \*/ public class TxConsumer {   public static final String QUEUE\_NAME = "test\_tx\_queue";   public static void main(String[] args) throws IOException, TimeoutException {   Connection connection = ConnectionUtils.getConnection();   Channel channel = connection.createChannel();   channel.queueDeclare(QUEUE\_NAME, false, false, false, null);   channel.basicConsume(QUEUE\_NAME, true, new DefaultConsumer(channel) {  @Override  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {  System.out.println("[tx\_queue] 消费：" + new String(body));  }  });  } } |

#### （2）[Publisher Confirms](https://www.rabbitmq.com/tutorials/tutorial-seven-java.html)

原理：

生产者将信道设置成confirm模式，一旦信道进入confirm模式，所有在该信道上面发布的消息都会被指派一个唯一的ID(从1开始)，一旦消息被投递到所有匹配的队列之后，broker就会发送一个确认给生产者（包含消息的唯一ID）,这就使得生产者知道消息已经正确到达目的队列了，如果消息和队列是可持久化的，那么确认消息会将消息写入磁盘之后发出，broker回传给生产者的确认消息中deliver-tag域包含了确认消息的序列号，此外broker也可以设置basic.ack的multiple域，表示到这个序列号之前的所有消息都已经得到了处理。

confirm模式最大的好处在于他是异步的，一旦发布一条消息，生产者应用程序就可以在等信道返回确认的同时继续发送下一条消息，当消息最终得到确认之后，生产者应用便可以通过回调方法来处理该确认消息，如果RabbitMQ因为自身内部错误导致消息丢失，就会发送一条nack消息，生产者应用程序同样可以在回调方法中处理该nack消息。

在channel 被设置成 confirm 模式之后，所有被 publish 的后续消息都将被 confirm（即 ack） 或者被nack一次。但是没有对消息被 confirm 的快慢做任何保证，并且同一条消息不会既被 confirm又被nack 。

注：如果原先使用了事务机制进行消息确认，不能再修改为confirm机制

###### i. 同步

a. 生产者01（单条）

缺点：每次一条，效率低

|  |
| --- |
| package com.he.rabbitmq.confirm;  import com.he.rabbitmq.util.ConnectionUtils; import com.rabbitmq.client.Channel; import com.rabbitmq.client.Connection;  import java.io.IOException; import java.util.concurrent.TimeoutException;  /\*\*  \* confirm机制（单条发送后确认）-生产者01  \*/ public class CfProducer01 {   public static final String QUEUE\_NAME = "test\_cf\_queue";   public static void main(String[] args) throws IOException, TimeoutException, InterruptedException {   Connection connection = ConnectionUtils.getConnection();   Channel channel = connection.createChannel();   channel.queueDeclare(QUEUE\_NAME, false, false, false, null);   String msg = "hello cf single";   //开启confirm模式  channel.confirmSelect();   //发送一条消息  channel.basicPublish("", QUEUE\_NAME, null, msg.getBytes());   //等待确认  if (channel.waitForConfirms()) {  System.out.println("消息【" + msg + "】发送成功");  } else {  System.out.println("消息【" + msg + "】发送失败");  }   //关闭通道  channel.close();   //关闭连接  connection.close();  } } |

b. 生产者02（批量）

缺点：成批的成功也可能成批的失败

|  |
| --- |
| package com.he.rabbitmq.confirm;  import com.he.rabbitmq.util.ConnectionUtils; import com.rabbitmq.client.Channel; import com.rabbitmq.client.Connection;  import java.io.IOException; import java.util.concurrent.TimeoutException;  /\*\*  \* confirm机制（单条发送后确认）-生产者02  \*/ public class CfProducer02 {   public static final String QUEUE\_NAME = "test\_cf\_queue";   public static void main(String[] args) throws IOException, TimeoutException, InterruptedException {   Connection connection = ConnectionUtils.getConnection();   Channel channel = connection.createChannel();   channel.queueDeclare(QUEUE\_NAME, false, false, false, null);   String msg = "hello cf single";   //开启confirm模式  channel.confirmSelect();   //批量发送消息  for (int i = 0; i < 10; i ++) {  channel.basicPublish("", QUEUE\_NAME, null, (msg+i).getBytes());  }   //等待确认  if (channel.waitForConfirms()) {  System.out.println("消息批量发送成功");  } else {  System.out.println("消息批量发送失败");  }   //关闭通道  channel.close();   //关闭连接  connection.close();  } } |

c. 消费者（不用变）

|  |
| --- |
| package com.he.rabbitmq.confirm;  import com.he.rabbitmq.util.ConnectionUtils; import com.rabbitmq.client.\*;  import java.io.IOException; import java.util.concurrent.TimeoutException;  /\*\*  \* 消息确认机制之Confirm机制-消费者  \*/ public class CfConsumer {   public static final String QUEUE\_NAME = "test\_cf\_queue";   public static void main(String[] args) throws IOException, TimeoutException {   Connection connection = ConnectionUtils.getConnection();   Channel channel = connection.createChannel();   channel.queueDeclare(QUEUE\_NAME, false, false, false, null);   channel.basicConsume(QUEUE\_NAME, true, new DefaultConsumer(channel) {  @Override  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {  System.out.println("[cf\_queue] 消费：" + new String(body));  }  });  } } |

###### ii. 异步

a. 生产者

|  |
| --- |
| package com.he.rabbitmq.confirm;  import com.he.rabbitmq.util.ConnectionUtils; import com.rabbitmq.client.Channel; import com.rabbitmq.client.ConfirmListener; import com.rabbitmq.client.Connection;  import java.io.IOException; import java.util.Collections; import java.util.SortedSet; import java.util.TreeSet; import java.util.concurrent.TimeoutException;  /\*\*  \* confirm机制（异步）-生产者03  \*/ public class CfProducer03 {   public static final String QUEUE\_NAME = "test\_cf\_queue";   public static void main(String[] args) throws IOException, TimeoutException, InterruptedException {   Connection connection = ConnectionUtils.getConnection();   Channel channel = connection.createChannel();   channel.queueDeclare(QUEUE\_NAME, false, false, false, null);   //开启confirm模式  channel.confirmSelect();   final SortedSet<Long> confirmSet = Collections.synchronizedSortedSet(new TreeSet<Long>());   //通道添加消息确认监听  channel.addConfirmListener(new ConfirmListener() {   //消息发送成功的回调（这里只做简单的移除SortedSet里面的发送成功的消息的id）  public void handleAck(long deliveryTag, boolean multiple) throws IOException {   if (multiple) {  //多条的情况  System.out.println("=======handleAck-multiple======");  confirmSet.headSet(deliveryTag +1).clear();  } else {  //单条的情况  System.out.println("=======handleAck======not-multiple");  confirmSet.remove(deliveryTag);  }  }   //消息发送失败的回调（这里只做简单的移除SortedSet里面的发送失败的消息的id，可以自行实现1s、3s、10s重试之类的）  public void handleNack(long deliveryTag, boolean multiple) throws IOException {  if (multiple) {  System.out.println("=======handleNack-multiple======");  confirmSet.headSet(deliveryTag + 1).clear();  } else {  System.out.println("=======handleNack======not-multiple");  confirmSet.remove(deliveryTag);  }  }  });   String msg = "hello cf single";   for (;;) {   //获取下一个消息的序列号  long nextPublishSeqNo = channel.getNextPublishSeqNo();   //消息发送  channel.basicPublish("", QUEUE\_NAME, null, msg.getBytes());   //把这个消息的序列号存到SortedSet中  confirmSet.add(nextPublishSeqNo);   }   } } |

### 2.2.8 死信队列（DLX）

Dead-Letter-Exchange在RabbitMQ中其实是一个交换机，用于处理死信

#### （1） 生产者

|  |
| --- |
| package com.he.rabbitmq.dlx;  import com.he.rabbitmq.util.ConnectionUtils; import com.rabbitmq.client.AMQP; import com.rabbitmq.client.Channel; import com.rabbitmq.client.ConfirmListener; import com.rabbitmq.client.Connection;  import java.io.IOException; import java.util.concurrent.TimeoutException;  /\*\*  \* 死信队列-生产者，例如：消息过期后变成死信  \*/ public class DLXProducer {   /\*\*  \* 普通交换机名称，不是死信队列  \*/  public static final String EXCHANGE\_NAME = "test\_dlx\_exchange";   public static void main(String[] args) throws IOException, TimeoutException {   Connection connection = ConnectionUtils.getConnection();   Channel channel = connection.createChannel();   channel.exchangeDeclare(EXCHANGE\_NAME, "topic", true, false, null);   channel.confirmSelect();   channel.addConfirmListener(new ConfirmListener() {  public void handleAck(long deliveryTag, boolean multiple) throws IOException {  System.out.println("========消息ack========");  }   public void handleNack(long deliveryTag, boolean multiple) throws IOException {  System.out.println("========消息Nack========");  }  });   String msg = "Hello dlx";   AMQP.BasicProperties properties = new AMQP.BasicProperties()  .builder()  .contentEncoding("utf-8")  .deliveryMode(2) //1.不是持久化消息 2.持久化消息  .expiration("10000") //设置10s过期，也可以声明队列时，设置队列内消息的过期时间  .build();  channel.basicPublish(EXCHANGE\_NAME, "dlx.add", properties, msg.getBytes());  System.out.println("[DLXProducer]: " + msg);   channel.close();   connection.close();  } } |

#### （2） 消费者

|  |
| --- |
| package com.he.rabbitmq.dlx;  import com.he.rabbitmq.util.ConnectionUtils; import com.rabbitmq.client.\*;  import java.io.IOException; import java.util.HashMap; import java.util.Map; import java.util.concurrent.TimeoutException;  /\*\*  \* 死信队列-消费者  \*/ public class DLXConsumer {   /\*\*  \* 普通交换机名称，不是死信队列  \*/  public static final String EXCHANGE\_NAME = "test\_dlx\_exchange";   /\*\*  \* 普通队列，不是死信队列的queue  \*/  public static final String QUEUE\_NAME = "test\_dlx\_queue";   /\*\*  \* 死信队列  \*/  public static final String EXCHANGE\_DLX\_NAME = "dlx.exchange";   /\*\*  \* 死信队列的queue  \*/  public static final String QUEUE\_DLX\_NAME = "dlx.queue";   public static void main(String[] args) throws IOException, TimeoutException {   Connection connection = ConnectionUtils.getConnection();   final Channel channel = connection.createChannel();   //普通队列、交换机的声明和绑定  channel.exchangeDeclare(EXCHANGE\_NAME, "topic", true, false, null);   /\*\*  \* x-dead-letter-exchange属性为死信队列（本质是一个交换机）的名称，并在普通队列声明时添加进去  \*/  Map<String, Object> properties = new HashMap<String, Object>();  properties.put("x-dead-letter-exchange", EXCHANGE\_DLX\_NAME);  channel.queueDeclare(QUEUE\_NAME, true, false, false, properties);  channel.queueBind(QUEUE\_NAME, EXCHANGE\_NAME, "dlx.#");   //死信队列、交换机的声明和绑定  channel.exchangeDeclare(EXCHANGE\_DLX\_NAME, "topic", true, false, null);  channel.queueDeclare(QUEUE\_DLX\_NAME, true, false, false, null);  //死信队列的路由键必须为#  channel.queueBind(QUEUE\_DLX\_NAME, EXCHANGE\_DLX\_NAME, "#");   DefaultConsumer consumer = new DefaultConsumer(channel) {  @Override  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {   System.out.println("[DLXConsumer]: " + new String(body));   channel.basicAck(envelope.getDeliveryTag(), false);  }  };   channel.basicConsume(QUEUE\_NAME, false, consumer);   } } |

# 3. 附页代码

RabbitMQ+SpringBoot整合代码