### Java连接RabbitMQ

public static Connection getConnection(){  
      //创建Connection工厂  
      ConnectionFactory factory = new ConnectionFactory();  
      factory.setVirtualHost("/test-1");  
      factory.setPassword("test");  
      factory.setUsername("test");  
      factory.setHost("192.168.61.134");  
      factory.setPort(5672);  
      Connection connection = null;  
      //拿到连接  
      try {  
          connection = factory.newConnection();  
      } catch (IOException e) {  
          e.printStackTrace();  
      } catch (TimeoutException e) {  
          e.printStackTrace();  
      }  
​  
      return connection;  
  }  
​

### Hello-world

1、创建生产者，创建一个channel，发送消息到exchange，指定路由规则

//生产者  
public static void publish() throws Exception {  
  //1、获取connection  
  Connection connection = RabbitConfig.getConnection();  
  //2、创建channel  
  Channel channel = connection.createChannel();  
  //3、发送消息到exchange  
    String msg = "hello world!!!";  
  /\*\*  
    \* 参数1：指定exchange，使用“”。默认的exchange  
    \* 参数2：指定路由的规则，使用具体的队列名称。exchange为""时，消息直接发送到队列中  
    \* 参数3：制动传递的消息携带的properties  
    \* 参数4：指定传递的消息，byte[]类型  
    \*/  
  channel.basicPublish("", "helloworld", null,msg.getBytes());  
  //PS:exchange是不会将消息持久化的，Queue可以持久化，得配置  
​  
  System.out.println("生产者发布消息成功！");  
  //4、关闭管道和连接  
  channel.close();  
  connection.close();  
​  
}

2、创建一个消费者，创建一个channel，创建一个队列，并且消费队列

//消费者    
public static void consumer() throws Exception{  
  //1、获取连对象、  
  Connection connection = RabbitConfig.getConnection();  
​  
  //2、创建channel  
  Channel channel = connection.createChannel();  
​  
  //3、创建队列-helloworld  
  /\*\*  
        \* 参数1：queue 指定队列名称  
        \* 参数2：durable 是否开启持久化（true）  
        \* 参数3：exclusive 是否排外（conn.close()-》当前对列自动删除，当前队列只能被一个 消费者消费）  
        \* 参数4：autoDelete 如果这个队列没有其他消费者在消费，队列自动删除  
        \* 参数5：arguments 指定队列携带的信息  
        \*  
        \*/  
  channel.queueDeclare("helloworld",true,false,false,null);  
​  
​  
  //4.开启监听Queue  
  DefaultConsumer consumer = new DefaultConsumer(channel){  
      @Override  
      public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) throws IOException {  
          System.out.println("接收到消息:"+new String(body,"UTF-8"));  
      }  
  };  
  /\*\*  
        \* 参数1：queue 指定消费哪个队列  
        \* 参数1：deliverCallback 指定是否ACK（true:收到消息会立即告诉RabbiMQ，false：手动告诉）  
        \* 参数1：cancelCallback 指定消费回调  
        \*  
        \*/  
  channel.basicConsume("helloworld",true,consumer);  
  System.out.println("消费者开始监听队列");  
​  
  //5、键盘录入，让程序不结束！  
  System.in.read();  
​  
  //6、释放资源  
  channel.close();  
  connection.close();  
​  
}

### Springboot整合RabbitMQ

1.创建Springboot工程

2.导入依赖

<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-amqp</artifactId>  
</dependency>  
​

3.写入配置文件application.yml

spring:  
rabbitmq:  
  host: 192.168.164.133  
  virtual-host: /test-1  
  username: test  
  password: test  
  port: 5672  
​

4.创建config配置，声明exchange和queue，并且绑定在一起

@Configuration  
public class MQConfig {  
​  
​  
  //1、创建exchange topic  
  @Bean  
  public TopicExchange getTopicExchage() {  
      return new TopicExchange("boot-topic-exchange", true, false);  
  }  
​  
  //2、创建queue  
  @Bean  
  public Queue getQueue(){  
      return new Queue("topic-queue",true,false,false,null);  
  }  
​  
​  
  //3、绑定  
  @Bean  
  public Binding geteBinding(TopicExchange topicExchange,Queue queue){  
      return BindingBuilder.bind(queue).to(topicExchange).with("ERROR");  
  }  
​  
}  
​

5.发布消息到RabbitMQ

@Test  
  void contextLoads() throws IOException {  
      rabbitTemplate.convertAndSend("boot-topic-exchange","ERROR","error");  
  }  
​

6.创建消费者监听器接收消息

@Component  
public class Consumer {  
  @RabbitListener(queues = "topic-queue")  
  public void getMassage(Object massage){  
      System.out.println("接收到消息："+massage);  
  }  
​  
}  
​

### Springboot整合RabbitMQ手动ACK

1.修改配置文件 关闭自动ack 开启手动ack

spring:  
rabbitmq:  
  listener:  
    simple:  
      acknowledge-mode: manual  
​

2.修改消费者类为手动ack

@RabbitListener(queues = "topic-queue")  
public void getMassage(String msg, Channel channel, Message message) throws IOException {  
  System.out.println("接收到消息："+msg);  
  //手动ack  
  channel.basicAck(message.getMessageProperties().getDeliveryTag(),false);  
}  
​

#### 通过Springboot实现Confirm以及Return机制

1、编写配置文件，开启Confirm和Return

spring:  
rabbitmq:  
  publisher-confirm-type: simple  
  publisher-returns: true  
​

2、 指定RabbitTemplate对象，开启Confirm个Return，实现回调方法

@Component  
public class PublisherConfirmAndReturnConfig implements RabbitTemplate.ConfirmCallback,RabbitTemplate.ReturnCallback {  
@Autowired  
private RabbitTemplate rabbitTemplate;  
​  
//在类加载的时候回执行这个方法  
@PostConstruct  
public void initMethod(){  
  rabbitTemplate.setConfirmCallback(this::confirm);  
  rabbitTemplate.setReturnCallback(this::returnedMessage);  
}  
​  
@Override  
public void confirm(CorrelationData correlationData, boolean ack, String cause) {  
  if (ack){  
      System.out.println("消息发送到Exchange成功!!!");  
  }else {  
      System.out.println("消息发送到Exchange失败");  
  }  
​  
}  
​  
@Override  
public void returnedMessage(Message message, int replyCode, String replyText, String exchange, String routingKey) {  
  System.out.println("消息发送到Queue失败！！！");  
​  
}  
​  
}  
​

#### Springboot解决消息重复消费

1、添加redis依赖

<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-data-redis</artifactId>  
</dependency>]  
​

2、修改配置文件

redis:  
  host: 192.168.164.136  
  port: 6379  
​

3、修改生产者

@Test  
void contextLoads() throws IOException {  
  //携带信息发送  
  CorrelationData messageId = new CorrelationData(UUID.randomUUID().toString());  
  rabbitTemplate.convertAndSend("boot-topic-exchange","ERROR","error",messageId);  
  System.in.read();  
}  
​

4、修改消费者

@RabbitListener(queues = "topic-queue")  
public void getMassage(String msg, Channel channel, Message message) throws IOException {  
  //1、获取messageID  
  String messageID = message.getMessageProperties().getHeader("spring\_returned\_message\_correlation");  
  //2、用redis的setnx()方法放入值 放入成功返回true 放入失败返回false  
  if (redisTemplate.opsForValue().setIfAbsent(messageID, "0", 10, TimeUnit.SECONDS)) {  
      //3、消费消息  
      System.out.println("接收到消息：" + msg);  
      //4、设置value值为1  
      redisTemplate.opsForValue().set(messageID, "1",10,TimeUnit.SECONDS);  
      //5、手动ack  
      channel.basicAck(message.getMessageProperties().getDeliveryTag(), false);  
  } else {  
      //6、如果放入值失败 获取messageID对应的value  
      String s = redisTemplate.opsForValue().get(messageID);  
      //7、value=0 什么都不做  
      if ("0".equalsIgnoreCase(s)) {  
          return;  
          //8、value=1 手动ack  
      } else {  
          channel.basicAck(message.getMessageProperties().getDeliveryTag(), false);  
      }  
​  
  }  
​  
}  
​