JMS – ActiveMQ

凯盛软件

JMS Java消息服务

凯盛软件

Java消息服务（Java Message Service，JMS）应用程序接口是一个Java平台中关于面向消息中间件（MOM）的API，用于在两个应用程序之间，或分布式系统中发送消息，进行异步通信。Java消息服务是一个与具体平台无关的API，绝大多数MOM提供商都对JMS提供支持。

Java消息服务的规范包括两种消息模式

- 点对点
- 发布者 / 订阅者

消息中间件


Apache
ACTIVEMQ


APACHE
kafka
A distributed streaming platform


RabbitMQ


RocketMQ
阿里巴巴中间件Aliware

JMS API

凯盛软件

- ConnectionFactory
- Connection
- Session
- Destination

  目标是一个包装了消息目标标识符的被管对象，消息目标是指消息发布和接收的地点，或者是队列，或者是主题
- MessageConsumer

  由会话创建的对象，用于接收发送到目标的消息
- MessageProducer

  由会话创建的对象，用于发送消息到目标
- Message

  是在消费者和生产者之间传送的对象，也就是说从一个应用程序创送到另一个应用程序

消息的类型

凯盛软件

- StreamMessage Java原始数据流
- MapMessage 键值对
- TextMessage 字符串对象
- ObjectMessage 序列化对象
- ByteMessage 字节数据流

ActiveMQ安装

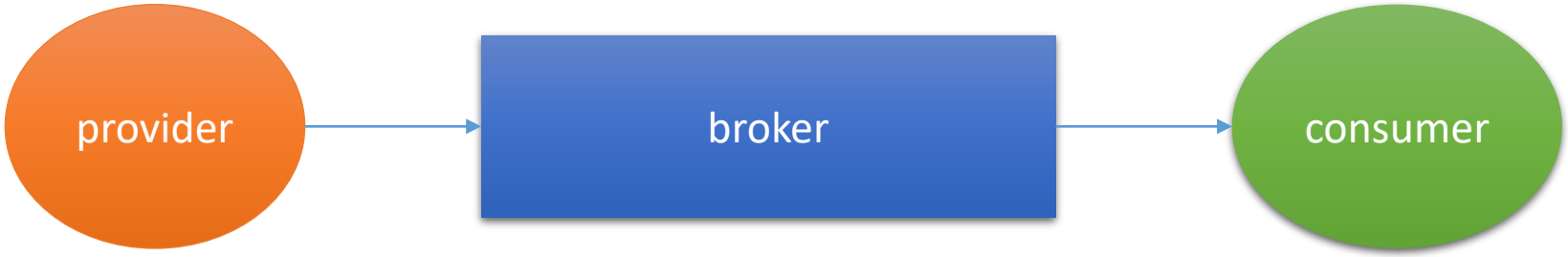
凯盛软件

- 下载 http://activemq.apache.org/download.html
- 解压文件夹，双击 home/bin/winXX/wrapper.exe 进行启动
- 浏览器中访问 http://localhost:8161
- 管理员账号和密码为 admin / admin

点对点消息

在点对点或队列模型下，一个生产者向一个特定的队列发布消息，一个消费者从该队列中读取消息。这里，生产者知道消费者的队列，并直接将消息发送到消费者的队列。这种模式被概括为：

- 只有一个消费者将获得消息
- 生产者不需要在接收者消费该消息期间处于运行状态，接收者也同样不需要在消息发送时处于运行状态。
- 每一个成功处理的消息都由接收者签收



消息生产者

凯盛软件

```
//1. 创建连接工厂
ConnectionFactory connectionFactory = new ActiveMQConnectionFactory("tcp://localhost:61616");
//2. 创建连接 并 开启
Connection connection = connectionFactory.createConnection();
connection.start();
//3. 创建Session
Session session = connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
//4. 创建消息目的地
Destination destination = session.createQueue("weixin-Queue");
//5. 创建生产者
MessageProducer producer = session.createProducer(destination);
//6. 发送消息
TextMessage textMessage = session.createTextMessage("Hello,MQ3");
producer.send(textMessage);
//7. 释放资源
producer.close();
session.close();
connection.close();
```

消息消费者

凯盛软件

```
//1. 创建连接工厂
ConnectionFactory connectionFactory = new
ActiveMQConnectionFactory(ActiveMQConnectionFactory.DEFAULT_BROKER_URL);
//2. 创建并启动连接
Connection connection = connectionFactory.createConnection();
connection.start();
//3. 创建Session
Session session = connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
//4. 创建目的地对象
Destination destination = session.createQueue( “weixin-Queue” );
//5. 创建消费者
MessageConsumer consumer = session.createConsumer(destination);
```

```
//6. 获取消息
consumer.setMessageListener(new MessageListener() {
    @Override
    public void onMessage(Message message) {
        TextMessage textMessage = (TextMessage) message;
        try {
            System.out.println(textMessage.getText());
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }
});
System.in.read();
//7. 释放资源
consumer.close();
session.close();
connection.close();
```

事务

凯盛软件

```
//使用事务，需要手动提交才可以发送消息
Session session = con.createSession(true,Session.AUTO_ACKNOWLEDGE);

//提交事务
session.commit();

//回滚事务
session.rollback();
```

签收模式

凯盛软件

- Session.AUTO_ACKNOWLEDGE 自动签收
- Session.CLIENT_ACKNOWLEDGE 消费端手工签收，可以方便失败时记录日志或其他处理，消费端接收消息后如果不签收，那么该消息依然会被认为未消费

生产端：

```
Session session = con.createSession(true,Session.CLIENT_ACKNOWLEDGE);
```

消费端：

```
Session session = con.createSession(false,Session.CLIENT_ACKNOWLEDGE);

//手动签收
textMessage.acknowledge();
```

持久模式

凯盛软件

```
MessageProducer messageProducer = session.createProducer(destination);
messageProducer.setDeliveryMode(DeliveryMode.PERSISTENT);
```

- DeliveryMode.PERSISTENT　持久传输，MQ服务重启后，未消费的消息还会存在
- DeliveryMode.NON_PERSISTENT　非持久传输，MQ服务重启后，未消费的消息将会消失

重试机制 Redelivery

凯盛软件

- 消息消费者接收到消息后进行处理，但是在处理过程中出现了异常，希望将该消息重新放入的消息队列中进行重试处理
- 触发Redelivery机制的方式
    - 在一个有事务的Session中，调用了rollback方法
    - 在Session中使用CLIENT_ACKNOWLEDGE签收模式，并且调用了session.recover()方法
    - 在Session中使用AUTO_ACKNOWLEDGE签收模式，在异步(messageListener)消费消息情况下，如果onMessage方法异常且没有被catch，此消息会被redelivery
- 触发Redelivery机制的消息会被重新放入到消息队列中，如果超过重试次数，持久性消息会被放入DLQ队列中，非持久性消息会被放弃

- 调用rollback

```
@Override
public void onMessage(Message message) {
    TextMessage textMessage = (TextMessage) message;
    try {
        String text = textMessage.getText();
        if(text.equals("Queue-6")) {
            throw new RuntimeException("异常");
        }
        System.out.println(text);
        session.commit();
    } catch (Exception e) {
        e.printStackTrace();
        try {
            session.rollback();
        } catch (JMSException e1) {
        }
    }
}
```

- 自动签收模式下，不catch异常

```
@Override
public void onMessage(Message message) {
    TextMessage textMessage = (TextMessage) message;
    try {
        String text = textMessage.getText();
        if(text.equals("Queue-6")) {
            throw new RuntimeException("异常");
        }
        System.out.println(text);
    } catch (Exception e) {
        e.printStackTrace();
        throw new RuntimeException(e);
    }
}
```

- 手动签收模式，调用session.recover()

```
@Override
public void onMessage(Message message) {
    TextMessage textMessage = (TextMessage) message;
    try {
        String text = textMessage.getText();
        if(text.equals("Queue-6")) {
            throw new RuntimeException("xxx");
        }
        textMessage.acknowledge();
    } catch (Exception e) {
        e.printStackTrace();
        session.recover();
    }
}
```

自定义重试参数

凯盛软件

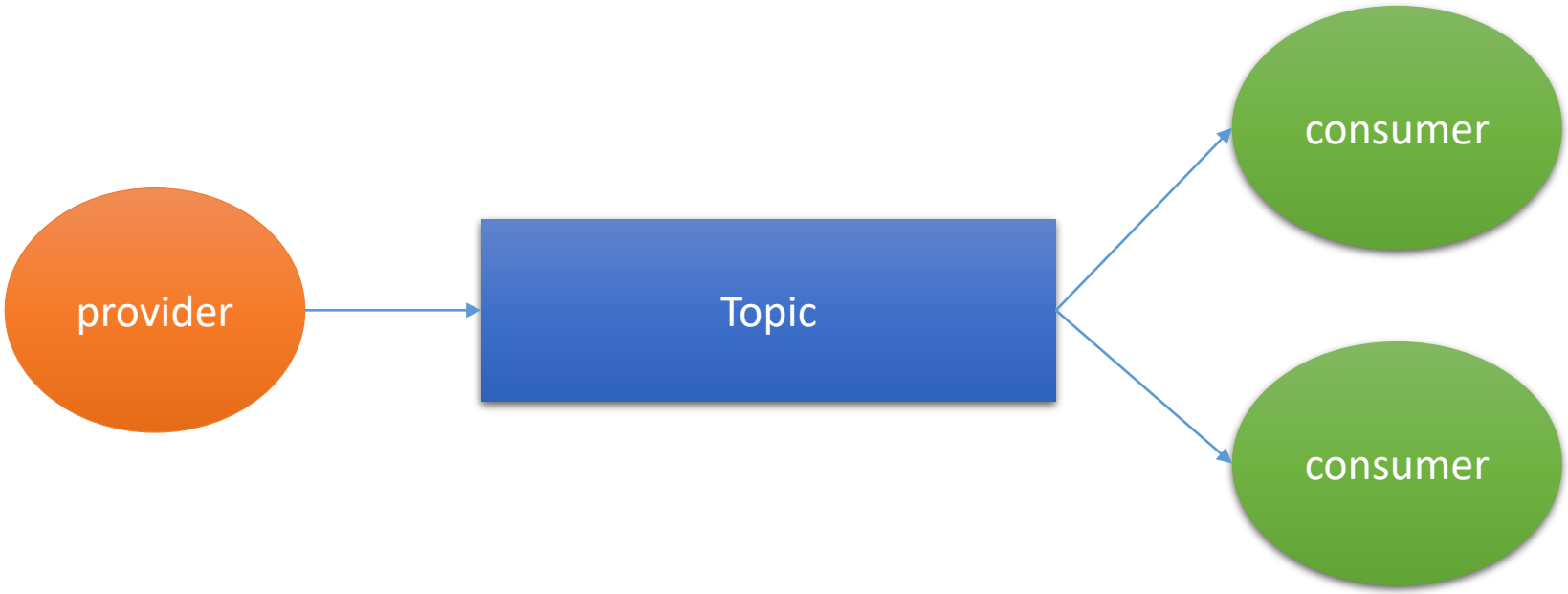
```
ActiveMQConnectionFactory connectionFactory = new ActiveMQConnectionFactory("tcp://localhost:61616");

RedeliveryPolicy redeliveryPolicy = new RedeliveryPolicy();
//设置重试次数
redeliveryPolicy.setMaximumRedeliveries(3);
//设置初次重试延迟时间，单位毫秒
redeliveryPolicy.setInitialRedeliveryDelay(5000);
//设置每次重试延迟时间，单位毫秒
redeliveryPolicy.setRedeliveryDelay(5000);

connectionFactory.setRedeliveryPolicy(redeliveryPolicy);
```

发布/订阅 消息

凯盛软件

发布者 / 订阅者模型支持向一个特定的消息主题发布消息。0或多个订阅者可能对接收来自特定消息主题的消息感兴趣。在这种模型下，发布者和订阅者彼此不知道对方。这种模式被概括为：

- 多个消费者可以获得消息
- 在发布者和订阅者之间存在时间依赖性。发布者需要建立一个订阅（subscription），以便客户能够购订阅。订阅者必须保持持续的活动状态以接收消息。



添加Maven依赖

凯盛软件

```
<dependency>
    <groupId>org.apache.activemq</groupId>
    <artifactId>activemq-all</artifactId>
    <version>5.15.0</version>
</dependency>
```

消息发布者

凯盛软件

```
//1. 创建连接工厂
ConnectionFactory connectionFactory = new ActiveMQConnectionFactory("tcp://localhost:61616");
//2. 创建连接 并 开启
Connection connection = connectionFactory.createConnection();
connection.start();
//3. 创建Session
Session session = connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
//4. 创建Topic对象
Topic topic = session.createTopic("weixin-Topic");
//5. 创建生产者
MessageProducer producer = session.createProducer(topic);
//6. 发送消息
TextMessage textMessage = session.createTextMessage("Hello,Topic MQ");
producer.send(textMessage);
//7. 释放资源
producer.close();
session.close();
connection.close();
```

消息订阅者

凯盛软件

```
//1. 创建连接工厂
ConnectionFactory connectionFactory = new
ActiveMQConnectionFactory(ActiveMQConnectionFactory.DEFAULT_BROKER_URL);
//2. 创建并启动连接
Connection connection = connectionFactory.createConnection();
connection.start();
//3. 创建Session
Session session = connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
//4. 创建目的地对象
Topic topic = session.createTopic("weixin-Topic");
//5. 创建消费这
MessageConsumer consumer = session.createConsumer(topic);
```

```
//6. 获取消息
consumer.setMessageListener(new MessageListener() {
    @Override
    public void onMessage(Message message) {
        TextMessage textMessage = (TextMessage) message;
        try {
            System.out.println(textMessage.getText());
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }
});
System.in.read();
//7. 释放资源
consumer.close();
session.close();
connection.close();
```

JMS + Spring

凯盛软件

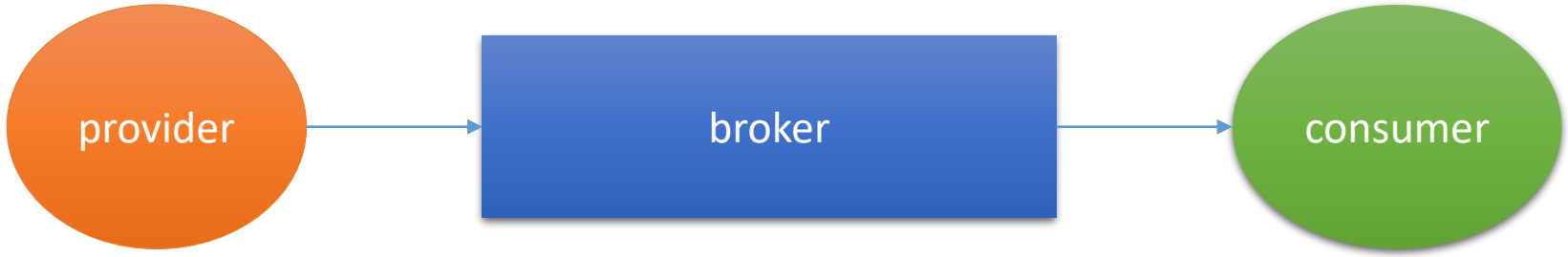
- 添加Maven依赖

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jms</artifactId>
    <version>4.3.9.RELEASE</version>
</dependency>
```

- 创建链接工厂

```
<!--配置ActiveMQ ConnectionFactory-->
<bean id="activeMQConnectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">
    <property name="brokerURL" value="tcp://localhost:61616"/>
</bean>

<!--Spring适配的连接工厂-->
<bean id="connectionFactory" class="org.springframework.jms.connection.SingleConnectionFactory">
    <property name="targetConnectionFactory" ref="activeMQConnectionFactory"/>
</bean>
```

点对点



- 配置JmsTemplate

```
<!--配置JMSTemplate-->
<bean id=“jmsTemplate” class=“org.springframework.jms.core.JmsTemplate” >
    <property name=“connectionFactory” ref=“connectionFactory” />
    <!--默认的目的地名称 可以省略，在发送时指定-->
    <property name="defaultDestinationName" value="weixin-Queue"/>
</bean>
```

发送请求

凯盛软件

```
@Autowired
private JmsTemplate jmsTemplate;

@Test
public void sendMessage() {
    jmsTemplate.send("weixin-Queue", new MessageCreator() {
        @Override
        public Message createMessage(Session session) throws JMSException {
            return session.createTextMessage("Spring MQ");
        }
    });
}
```

接收请求

凯盛软件

- 创建MessageListener

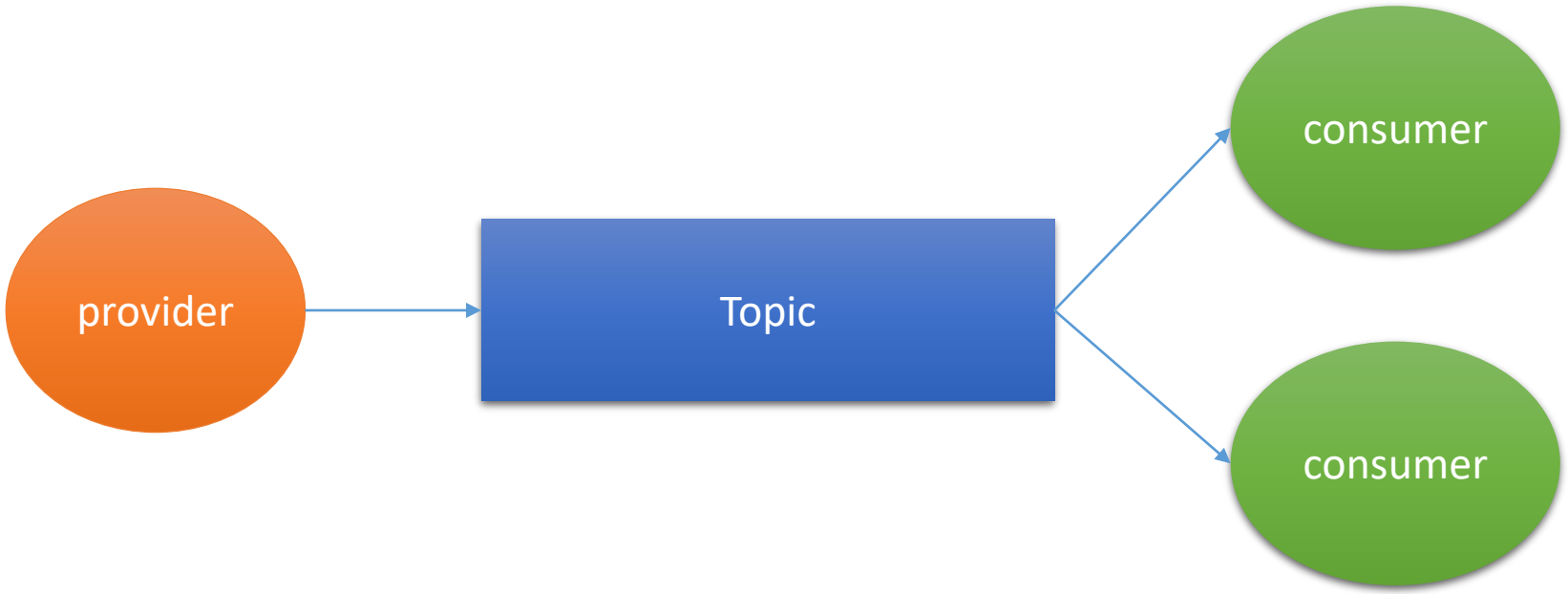
```
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;

public class WeixinListener implements MessageListener {
    @Override
    public void onMessage(Message message) {
        TextMessage textMessage = (TextMessage) message;
        try {
            System.out.println(textMessage.getText());
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }
}
```

- 在Spring中配置监听器。配置监听器后，当MQ队列中有消息就会自动触发监听器的运行

```
<!--监听器-->
<bean id="listener" class="com.kaishengit.mq.listener.WeixinListener"/>

<!--监听器容器-->
<bean class="org.springframework.jms.listener.DefaultMessageListenerContainer">
    <property name="connectionFactory" ref="connectionFactory"/>
    <property name="destinationName" value="weixin-Queue"/>
    <property name="messageListener" ref="listener"/>
</bean>
```

订阅发布模式

凯盛软件



- 配置JmsTemplate

```
<!--配置JMSTemplate-->
<bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate">
    <property name="connectionFactory" ref="connectionFactory"/>
</bean>
```

- 配置目的地对象

```
<!--Topic对象-->
<bean id="destination" class="org.apache.activemq.command.ActiveMQTopic">
    <!--主题名称-->
    <constructor-arg name="name" value="weixin-Topic"/>
</bean>
```

发送消息

凯盛软件

```
@Autowired
private JmsTemplate jmsTemplate;
@Autowired
private Destination destination;

@Test
public void sendMessage() throws IOException {
    jmsTemplate.send(destination, new MessageCreator() {
        @Override
        public Message createMessage(Session session) throws JMSException {
            return session.createTextMessage("Spring MQ");
        }
    });

    System.in.read();
}
```

接收消息

凯盛软件

```
<!--监听器-->
<bean id="listener" class="com.kaishengit.mq.listener.WeixinListener"/>

<!--监听器容器-->
<bean class="org.springframework.jms.listener.DefaultMessageListenerContainer">
    <property name="connectionFactory" ref="connectionFactory"/>
    <property name="destination" ref="destination"/>
    <property name="messageListener" ref="listener"/>
</bean>
```

添加多个监听器

凯盛软件

```
<!--Queue监听-->
<jms:listener-container acknowledge="client" connection-factory="connectionFactory"
                        concurrency="3-10">
    <jms:listener destination="Spring-Queue1" ref="quereConsuerm1"/>
    <jms:listener destination="Spring-Queue2" ref="quereConsuerm2"/>
</jms:listener-container>
<!--Topic监听-->
<jms:listener-container destination-type="topic" connection-factory="connectionFactory"
                        concurrency="3-10" acknowledge="client">
    <jms:listener destination="spring-Topic" ref="topicConsumer"/>
</jms:listener-container>
```

使用注解模式接收Queue消息

凯盛软件

```
<jms:annotation-driven container-factory="jmsListenerContainerFactory"/>

<bean id="jmsListenerContainerFactory" class="org.springframework.jms.config.DefaultJmsListenerContainerFactory">
    <property name="connectionFactory" ref="connectionFactory"/>
    <!--线程池的配置，控制在3-10个线程-->
    <property name="concurrency" value="3-10"/>
</bean>
```

```
@Component
public class SpringAnnotationListener {

    @JmsListener(destination = "spring-mq")
    public void doSomething(String message) {
        System.out.println("<<<<<<<<<<<< " + message);
    }
}
```

使用注解模式接收Topic消息

凯盛软件

```
<!--开启基于注解的JMS接收模式-->
<bean id="jmsListenerContainerFactory" class="org.springframework.jms.config.DefaultJmsListenerContainerFactory">
    <property name="connectionFactory" ref="connectionFactory"/>
    <property name="concurrency" value="5-15"/>
    <!--监听发布/订阅模式-->
    <property name="pubSubDomain" value="true"/>
</bean>

<jms:annotation-driven container-factory="jmsListenerContainerFactory"/>

@JmsListener(destination = "spring-Topic")
public void doSomething(String message) {
    System.out.println("****** " + message);
}
```

同时监听Queue和Topic

凯盛软件

```
<!--开启基于注解的JMS接收模式-->
<bean id="jmsListenerContainerFactory" class="org.springframework.jms.config.DefaultJmsListenerContainerFactory">
    <property name="connectionFactory" ref="connectionFactory"/>
    <property name="concurrency" value="5-15"/>
    <!--监听发布/订阅模式-->
    <property name="pubSubDomain" value="true"/>
</bean>
<bean id="jmsQueueListenerContainerFactory" class="org.springframework.jms.config.DefaultJmsListenerContainerFactory">
    <property name="connectionFactory" ref="connectionFactory"/>
    <property name="concurrency" value="5-15"/>
</bean>
<jms:annotation-driven container-factory="jmsListenerContainerFactory"/>
```

```java
@JmsListener(destination = "spring-Topic")
public void doSomething(String message) {
    System.out.println("****** " + message);
}

@JmsListener(destination = "Spring-Queue",containerFactory = "jmsQueueListenerContainerFactory")
public void getMessageFromQueue(String message) {
    System.out.println(">>>>>" + message);
}
```

重试机制

凯盛软件

- Spring中的JMS监听器在触发到重试机制后会自动运行。可以自定义重试的参数

```
<!--重试机制-->
<bean id="redeliveryPolicy" class="org.apache.activemq.RedeliveryPolicy">
    <property name="maximumRedeliveries" value="3"/>
    <property name="initialRedeliveryDelay" value="3000"/>
    <property name="redeliveryDelay" value="3000"/>
</bean>

<!--ActiveMQ ConnectionFactory-->
<bean id="activeMQConnectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">
    <property name="brokerURL" value="tcp://localhost:61616"/>
    <property name="redeliveryPolicy" ref="redeliveryPolicy"/>
 </bean>
```

```
<!--JMSTemplate-->
<bean id="jmsTemplate"  class="org.springframework.jms.core.JmsTemplate" >
    <!--配置链接工厂-->
    <property name="connectionFactory"  ref="connectionFactory" />
    <!--2表示客户端签收模式-->
    <property name="sessionAcknowledgeMode" value="2"/>
</bean>
```

使用客户端签收模式，需要使用到JMS中的session对象进行recover操作来触发重试机制，但是JMS中的MessageListener接口的onMessage方法中只有Message参数，没有Session参数。此时可以让监听器实现org.springframework.jms.listener.SessionAwareMessageListener接口，该接口的OnMessage方法中有Message和Session两个参数

```
@Component
public class QuereConsuerm implements SessionAwareMessageListener {

    @Override
    public void onMessage(Message message, Session session) throws JMSException {
```

- 监听器完整示例

```
@Component
public class QuereConsuerm implements SessionAwareMessageListener {

    @Override
    public void onMessage(Message message, Session session) throws JMSException {
        try {
            TextMessage textMessage = (TextMessage) message;
            System.out.println("&&&& -> " + textMessage.getText());
            if(1==1) {
                throw new RuntimeException();
            }
            message.acknowledge();
        } catch (Exception ex) {
            ex.printStackTrace();
            session.recover();
        }
    }
}
```

- 在使用@JmsListener注解的监听器方法上，也可以让方法的参数为JMS的Message对象和Session对象，Spring会自动注入这两个对象

```
@JmsListener(destination = "Spring-Queue",containerFactory = "jmsQueueListenerContainerFactory")
public void getMessageFromQueue(Message message, Session session) {
    TextMessage textMessage = (TextMessage) message;
    try {
        System.out.println(">>>>>" + textMessage.getText());
        if (1==1) {
            throw new RuntimeException();
        }
        message.acknowledge();
    } catch (JMSException e) {
        e.printStackTrace();
        try {
            session.recover();
        } catch (JMSException e1) {
            e1.printStackTrace();
        }
    }
}
```