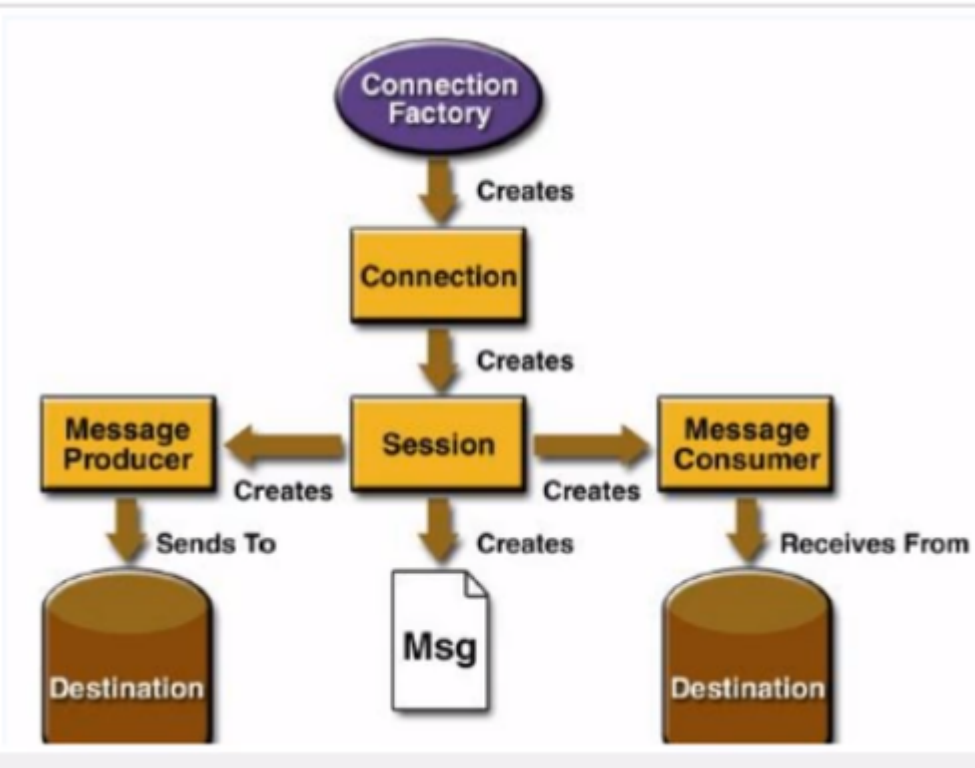
1.在何种情况下使用了消息中间件

1. 系统之间接口耦合比较严重:每新增一个下游功能,都要对上游的相关接口进行改造.
2. 面对大流量并发时,容易被冲垮.
3. 等待同步存在性能问题:RPC接口基本是同步调用,整体的服务性能遵循"木桶理论",即整体系统的耗时取决于链路中最慢的那个接口.

   主要能狗解决:解耦,流量削峰,异步处理

2.ActiveMQ的HelloWord



2.1.Queue

在点对点的消息传递中,目的地被称为队列(Queue)

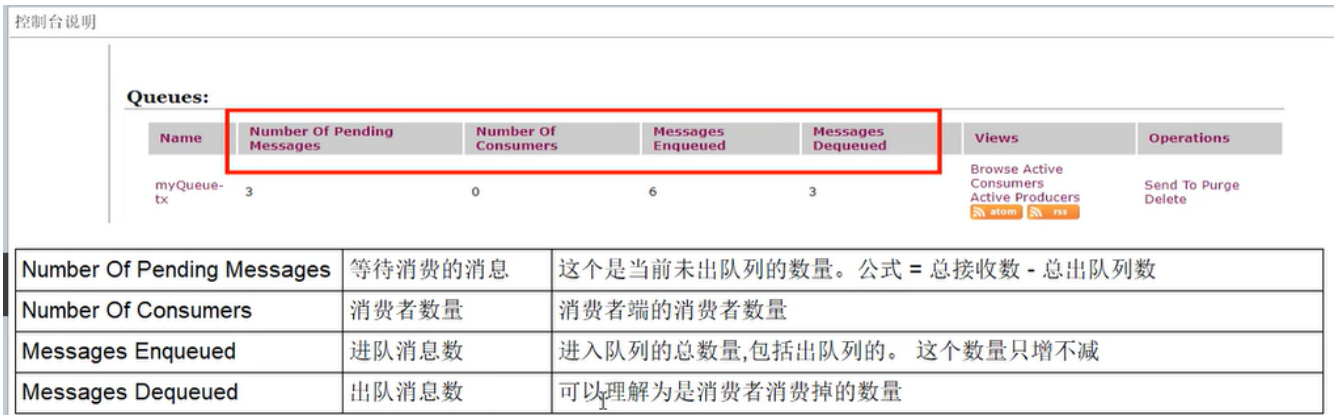
消息生产者程序

```
public class ActiveMQProducer {
    public static final String ACTIVEMQ_URL = "tcp://localhost:61616";
    public static final String QUEUE_NAME = "queue01";

    public static void main(String[] args) throws JMSException {
        //1.创建连接工厂,按照给定的url地址,采用默认用户名和密码
```

```
        ActiveMQConnectionFactory activeMQConnectionFactory = new
ActiveMQConnectionFactory(ACTIVEMQ_URL);
        //2.通过连接工厂,获得连接connection并启动访问
        Connection connection = activeMQConnectionFactory.createConnection();
        connection.start();
        //3.创建会话session,第一个参数是事务,第二个参数是签收
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        //4.创建目的地(具体是queue还是topic)
        Queue queue = session.createQueue(QUEUE_NAME);
        //5.创建消息的生产者
        MessageProducer producer = session.createProducer(queue);
        //6.通过使用messagerProducer生产3条消息发送到MQ的队列里面
        for (int i = 0; i < 3; i++) {
            //7.创建消息
            TextMessage textMessage = session.createTextMessage("msg---" + i);
            //8.通过messagerProducer发送给mq
            producer.send(textMessage);
        }
        //9.关闭资源
        producer.close();
        session.close();
        connection.close();
        System.out.println("---消息发布到MQ---");
    }
}
```

控制台对于Queue的说明



Number Of Pending Messages	等待消费的消息	这个是当前未出队列的数量。公式 = 总接收数 - 总出队列数
Number Of Consumers	消费者数量	消费者端的消费者数量
Messages Enqueued	进队消息数	进入队列的总数量,包括出队列的。 这个数量只增不减
Messages Dequeued	出队消息数	可以理解为是消费者消费掉的数量

消息消费者程序

```
public class ActiveMQConsumer {
    public static final String ACTIVEMQ_URL = "tcp://localhost:61616";
    public static final String QUEUE_NAME = "queue01";

    public static void main(String[] args) throws JMSException {
        //1.创建连接工厂,按照给定的url地址,采用默认用户名和密码
        ActiveMQConnectionFactory activeMQConnectionFactory = new
ActiveMQConnectionFactory(ACTIVEMQ_URL);
        //2.通过连接工厂,获得连接connection并启动访问
        Connection connection = activeMQConnectionFactory.createConnection();
```

```
        connection.start();
        //3.创建会话session,第一个参数是事务,第二个参数是签收
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        //4.创建目的地(具体是queue还是topic)
        Queue queue = session.createQueue(QUEUE_NAME);
        //5.创建消费者
        MessageConsumer consumer = session.createConsumer(queue);
        while(true){
            TextMessage textMessage = (TextMessage) consumer.receive();
            if (textMessage!=null){
                System.out.println(textMessage.getText());
            }else{
                break;
            }
        }
        consumer.close();
        session.close();
        connection.close();
    }
}
```

receive( ):这是一种同步阻塞方式:订阅者或接收者调用MessageConsumer的receive( )方法来接受消息,receive方法能够接收到消息之前(或超时之前)将一直阻塞.接受到一个消息或者超时,就会关闭.

```
public class ActiveMQConsumer {
    public static final String ACTIVEMQ_URL = "tcp://localhost:61616";
    public static final String QUEUE_NAME = "queue01";

    public static void main(String[] args) throws JMSException, IOException {
        //1.创建连接工厂,按照给定的url地址,采用默认用户名和密码
        ActiveMQConnectionFactory activeMQConnectionFactory = new
ActiveMQConnectionFactory(ACTIVEMQ_URL);
        //2.通过连接工厂,获得连接connection并启动访问
        Connection connection = activeMQConnectionFactory.createConnection();
        connection.start();
        //3.创建会话session,第一个参数是事务,第二个参数是签收
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        //4.创建目的地(具体是queue还是topic)
        Queue queue = session.createQueue(QUEUE_NAME);
        //5.创建消费者
        MessageConsumer consumer = session.createConsumer(queue);
        //通过监听的方式来获取消息.
        consumer.setMessageListener(new MessageListener() {
            @Override
            public void onMessage(Message message) {
                if (message!=null && message instanceof TextMessage){
                    TextMessage textMessage = (TextMessage) message;
                    try {
                        System.out.println(textMessage.getText());
                    } catch (JMSException e) {
                        e.printStackTrace();
                    }
                }
```

```
            }
        });
        //如果不加这句话,有可能消费不掉消息就会关闭.
        System.in.read();
    }
}
```

onMessage():异步非阻塞方式,订阅者或接收者通过MessageConsumer的setMessageListener(MessageListener listener)注册一个消息监听器,当消息到达后,系统自动调用监听器MessageListener的onMessage(Message message)方法.

JMS开发的基本步骤

1. 创建一个connectionFactory
2. 通过connectionFactory来创建connection
3. 启动connection
4. 通过connection创建session
5. 创建destination
6. 创建producer或者创建message并设置destination
7. 创建consumer或者注册一个message listener
8. 发送或者接受message(s)
9. 关闭所有的JMS资源.

2.2.Topic

1. 消息生产者程序

```
public class ActiveMQProducer {
    public static final String ACTIVEMQ_URL = "tcp://localhost:61616";
    public static final String TOPIC_NAME = "topic01";

    public static void main(String[] args) throws JMSException {
        //1.创建连接工厂,按照给定的url地址,采用默认用户名和密码
        ActiveMQConnectionFactory activeMQConnectionFactory = new
ActiveMQConnectionFactory(ACTIVEMQ_URL);
        //2.通过连接工厂,获得连接connection并启动访问
        Connection connection = activeMQConnectionFactory.createConnection();
        connection.start();
        //3.创建会话session,第一个参数是事务,第二个参数是签收
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        //4.创建目的地(具体是queue还是topic)
        Topic topic = session.createTopic(TOPIC_NAME);
        //5.创建消息的生产者
        MessageProducer producer = session.createProducer(topic);
        //6.通过使用messagerProducer生产3条消息发送到MQ的队列里面
        for (int i = 0; i < 3; i++) {
            //7.创建消息
            TextMessage textMessage = session.createTextMessage("msg---" + i);
            //8.通过messagerProducer发送给mq
            producer.send(textMessage);
        }
```

```
        //9.关闭资源
        producer.close();
        session.close();
        connection.close();
        System.out.println("---消息发布到MQ---");
    }
}
```

2. 消息消费者程序

```
public class ActiveMQConsumer {
    public static final String ACTIVEMQ_URL = "tcp://localhost:61616";
    public static final String TOPIC_NAME = "topic01";

    public static void main(String[] args) throws JMSException, IOException {
        System.out.println("我是2号消费者");
        //1.创建连接工厂,按照给定的url地址,采用默认用户名和密码
        ActiveMQConnectionFactory activeMQConnectionFactory = new
ActiveMQConnectionFactory(ACTIVEMQ_URL);
        //2.通过连接工厂,获得连接connection并启动访问
        Connection connection = activeMQConnectionFactory.createConnection();
        connection.start();
        //3.创建会话session,第一个参数是事务,第二个参数是签收
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        //4.创建目的地(具体是queue还是topic)
        Topic topic = session.createTopic(TOPIC_NAME);
        //5.创建消费者
        MessageConsumer consumer = session.createConsumer(topic);
        //通过监听的方式来获取消息.
        consumer.setMessageListener(new MessageListener() {
            @Override
            public void onMessage(Message message) {
                if (message!=null && message instanceof TextMessage){
                    TextMessage textMessage = (TextMessage) message;
                    try {
                        System.out.println(textMessage.getText());
                    } catch (JMSException e) {
                        e.printStackTrace();
                    }
                }
            }
        });
        System.in.read();
    }
}
```

2.3.Queue与Topic的对比

比较项目	Topic 模式队列	Queue 模式队列
工作模式	“订阅-发布”模式，如果当前没有订阅者，消息将会被丢弃。如果有多个订阅者，那么这些订阅者都会收到消息	“负载均衡”模式，如果当前没有消费者，消息也不会丢弃；如果有多个消费者，那么一条消息也只会发送给其中一个消费者，并且要求消费者ack信息。
有无状态	无状态	Queue数据默认会在mq服务器上以文件形式保存，比如Active MQ一般保存在$AMQ_HOME\data\kr-store\data下面。也可以配置成DB存储。
传递完整性	如果没有订阅者，消息会被丢弃	消息不会丢弃
处理效率	由于消息要按照订阅者的数量进行复制，所以处理性能会随着订阅者的增加而明显降低，并且还要结合不同消息协议自身的性能差异	由于一条消息只发送给一个消费者，所以就算消费者再多，性能也不会有明显降低。当然不同消息协议的具体性能也是有差异的

3.JMS

3.1.JMS产品的比较

特性	ActiveMQ	RabbitMQ	Kafka	RocketMQ
PRODUCER-COMSUMER	支持	支持	支持	支持
PUBLISH-SUBSCRIBE	支持	支持	支持	支持
REQUEST-REPLY	支持	支持	-	支持
API完备性	高	高	高	低（静态配置）
多语言支持	支持，JAVA优先	语言无关	支持，JAVA优先	支持
单机吞吐量	万级	万级	十万级	单机万级
消息延迟	-	微秒级	毫秒级	-
可用性	高（主从）	高（主从）	非常高（分布式）	高
消息丢失	-	低	理论上不会丢失	-
消息重复	-	可控制	理论上会有重复	-
文档的完备性	高	高	高	中
提供快速入门	有	有	有	无
首次部署难度	-	低	中	高

JMS组成的四大元素

1. JMS provider:实现 JMS 接口规范的消息中间间,也就是我们的MQ服务器
2. JMS producer:消息生产者,创建和发送JMS 消息的客户端应用
3. JMS consumer:消息消费者,接收和处理JMS消息的客户端应用
4. JMS message:消息头,消息属性,消息体

3.2.Message

1.消息头

1. JMSDestination:目的地,是Queue还是Topic
2. JMSDeliveryMode:持久和非持久模式.
3. JMSExpiration:设置消息在一定时间后过期,默认是永不过期.
4. JMSPriority:消息优先级,0-9十个级别,0-4是普通消息,5-9是加急消息,默认级别是4级.
5. JMSMessageID:唯一识别,每个消息的标识由MQ产生.

2.消息体

1. TextMessage:普通的字符串消息,包含一个string.
2. MapMessage:一个Map类型的消息,key是String类型,而值是java的基本类型
3. BytesMessage:二进制数组消息,包含一个byte[]
4. StreamMessage:Java数据流消息,用标准流操作来顺序的填充和读取.
5. ObjectMessage:对象消息,包含一个可序列化的Java对象.

   发送和接受的消息体的类型必须一致

3.消息属性

他们是以属性名和属性值对的形式制定的,可以将属性为消息头的扩展,属性指定一些消息头没有包括的附加信息,比如可以在属性里指定消息选择器.

3.3 JMS的可靠性

1.持久化(Persistent)

1.Queue的持久化

非持久化:messageProducer.setDeliveryModel(DeliveryMode.NON_PERSISTENT),当服务器宕机,消息不存在.

持久化:messageProducer.setDeliveryModel(DeliveryMode.PERSISTENT),当服务器宕机,消息依然存在.

Queue默认是持久化的

2.Topic的持久化

1. 生产者程序

```
public class ActiveMQProducer {
    public static final String ACTIVEMQ_URL = "tcp://localhost:61616";
    public static final String TOPIC_NAME = "topic01";

    public static void main(String[] args) throws JMSException {
        //1.创建连接工厂,按照给定的url地址,采用默认用户名和密码
        ActiveMQConnectionFactory activeMQConnectionFactory = new
ActiveMQConnectionFactory(ACTIVEMQ_URL);
        //2.通过连接工厂,获得连接connection并启动访问
        Connection connection = activeMQConnectionFactory.createConnection();
        //3.创建会话session,第一个参数是事务,第二个参数是签收
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        //4.创建目的地(具体是queue还是topic)
        Topic topic = session.createTopic(TOPIC_NAME);
        //5.创建消息的生产者
        MessageProducer producer = session.createProducer(topic);
        //6.生产者设置消息持久化
        producer.setDeliveryMode(DeliveryMode.PERSISTENT);
        //7.连接启动
        connection.start();
        //8.通过使用messagerProducer生产3条消息发送到MQ的队列里面
        for (int i = 0; i < 3; i++) {
            //9.创建消息
            TextMessage textMessage = session.createTextMessage("msg---" + i);
            //10.通过messagerProducer发送给mq
            producer.send(textMessage);
        }
        //11.关闭资源
        producer.close();
        session.close();
        connection.close();
        System.out.println("---消息发布到MQ---");
    }
```

```
}
```

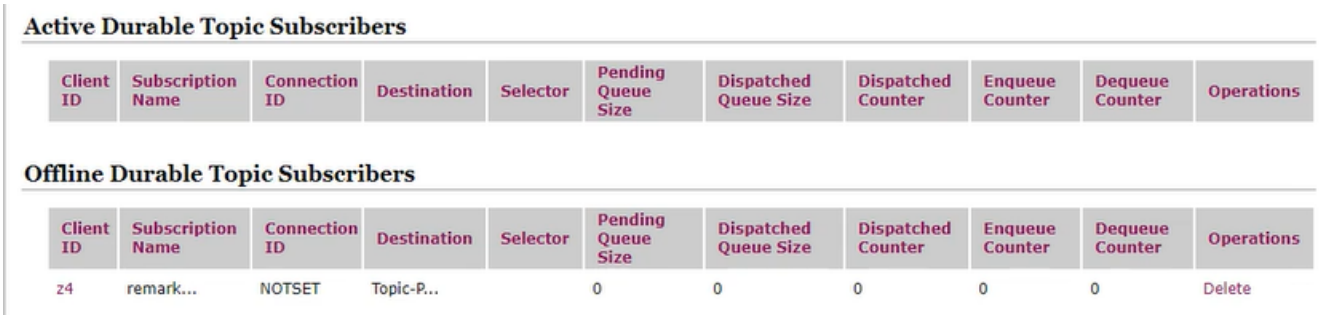
2. 消费者程序

```
public class ActiveMQConsumer {
    public static final String ACTIVEMQ_URL = "tcp://localhost:61616";
    public static final String TOPIC_NAME = "topic01";

    public static void main(String[] args) throws JMSException, IOException {
        System.out.println("我是1号消费者");
        //1.创建连接工厂,按照给定的url地址,采用默认用户名和密码
        ActiveMQConnectionFactory activeMQConnectionFactory = new
ActiveMQConnectionFactory(ACTIVEMQ_URL);
        //2.通过连接工厂,获得连接connection并启动访问
        Connection connection = activeMQConnectionFactory.createConnection();
        //3.谁订阅,需要给
        connection.setClientID("Z1");
        //4.创建会话session,第一个参数是事务,第二个参数是签收
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        //5.创建目的地(具体是queue还是topic)
        Topic topic = session.createTopic(TOPIC_NAME);
        //6.创建持久化的订阅,这边的"remake"就是SubscriptionName
        TopicSubscriber topicSubscriber = session.createDurableSubscriber(topic, "remark");
        connection.start();
        //7.当receive没有收到消息的时候,就会一直堵塞在这边
        Message message = topicSubscriber.receive();
        while(message!=null){
            TextMessage textMessage = (TextMessage) message;
            System.out.println(textMessage.getText());
            //8.等待5s,如果没有收到消息,就会关闭,那么message就会为null,
            message = topicSubscriber.receive(5000L);
        }
        session.close();
        connection.close();
    }
}
```

非持久订阅:只有当客户端处于激活状态,也就是和MQ保持连接状态才能收到发送到某个主题的消息,如果消费者处于离线状态,生产者发送的主题消息将会丢失作废,消费者永远不会收到.一句话:**先要订阅注册才能收到发布,只给订阅者发布消息**.

持久化订阅:客户端先想MQ注册一个自己的身份ID识别号,当这个客户端处于离线的时候,生产者会为这个id保存所有发送到主题的消息,当客户端再次连接到MQ时会根据消费者的ID得到所有当自己处于离线时候发送到主题的消息.



Active Durable Topic Subscribers

Client ID	Subscription Name	Connection ID	Destination	Selector	Pending Queue Size	Dispatched Queue Size	Dispatched Counter	Enqueue Counter	Dequeue Counter	Operations

Offline Durable Topic Subscribers

Client ID	Subscription Name	Connection ID	Destination	Selector	Pending Queue Size	Dispatched Queue Size	Dispatched Counter	Enqueue Counter	Dequeue Counter	Operations
z4	remark...	NOTSET	Topic-P...		0	0	0	0	0	Delete

2.事务(transaction)

1. 当事务是false的时候:只要执行send,就会进入到队列中.
2. 当事务是true的时候,先执行send再执行**session.commit( )**,消息才会被真正的提交到队列中.如果在一个事务中出现异常,可以使用**session.rollback( )**进行事务回滚.

3.签收(Acknowledge)

1.非事务模式下的消费者签收介绍:

1. 自动签收(默认): Session.AUTO_ACKNOWLEDGE
2. 手动签收:Session.CLIENT_ACKNOWLEDGE:需要客户端调用acknowledge方法手动签收.

需要使用**textMessage.acknowledge( )**;来确认签收,如果不确认签收,会出现重复消费的情况.

2.事务模式下的消费者签收介绍:

当消费者事务是true,即当开启事务的时候.那么,可以这么认为:事务会接管签收,当事务commit,正常,当事务不commit.会出现重复消费的情况.

3.签收和事务的关系:

在事务性会话中,当一个事务被成功提交则消息被自动签收,如果事务回滚,那么消息会被再次传送.

非事务性会话中,消息合适被确认取决于创建会话时的应答模式(acknowledgement mode).

4.ActiveMQ的Broker

相当于一个ActiveMQ服务器实例,说白了,Broker其实就是实现了用代码的形式启动ActiveMQ将MQ嵌入到Java代码中,以便随时用随时启动,在用的时候再去启动这样能节省了资源,也保证了可靠性.

```
public class EmbedBroker {
    public static void main(String[] args) throws Exception {
        //ActiveMQ也支持在vm中通信基于嵌入式的broker
        BrokerService brokerService  = new BrokerService();
        brokerService.setUseJmx(true);
        brokerService.addConnector("tcp://localhost:61616");
        brokerService.start();
    }
}
```

5.Spring整合ActiveMQ(参考印象笔记)