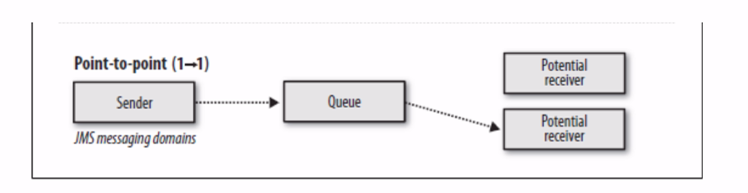
# 快速入门

## 点对点



消息生产者：

package code.lsh.activemq;  
  
public class JmsProducer  
{  
  
 private static final String *ACTIVEMQ\_URL* = "tcp://118.25.112.103:";  
  
 private static final String *Queue\_NANE* = "queue\_01" ;  
  
 public static void main(String[] args)  
 {  
 ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory(*ACTIVEMQ\_URL*);  
  
 Connection connection = factory.creaetConnection();  
  
 connection.start();  
  
 Session session = connection.createSession(false,Session.AUTO\_ACKNOWLEDGE);  
  
 Queue queue = session.createQueue(*Queue\_NANE*);  
  
 MessageProducer producer = session.creaetProducer(queue);  
  
 TextMessage message = session.createTextMessage("hello world");  
  
 producer.send(message);  
  
 producer.close();  
  
 session.close();  
  
 connection.close();  
   
  
 }  
}

这里

Number Of Pending Messages：等待消费的消息，这个是当前未出队列的数量

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。这个数量只增不减

Messages Dequeued：出队消息数，可以理解为消费者消费掉的数量。

Views：

Operations：

消息消费者：

public class JmsConsumer  
{  
 private static final String *ACTIVEMQ\_URL* = "tcp://118.25.112.103:";  
  
 private static final String *Queue\_NANE* = "queue\_01" ;  
  
 public static void main(String[] args)  
 {  
 ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory(*ACTIVEMQ\_URL*);  
  
 Connection connection = factory.creaetConnection();  
  
 connection.start();  
  
 Session session = connection.createSession(false,Session.AUTO\_ACKNOWLEDGE);  
  
 Queue queue = session.createQueue(*Queue\_NANE*);  
  
 MessageConsumer consumer = session.createConsumer(queue);  
 while(true)  
 {  
 TextMessage message = (TextMessage)consumer.receive();  
 (null!=message):(System.*out*.println("Consumer consume message:"+message.getText()))?break;  
 }  
 consumer.close();  
 session.close();  
 connection.close();  
 }  
}

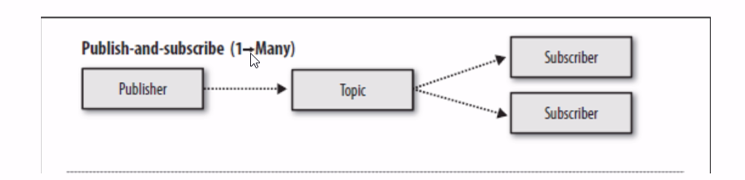
消息消费者第二种写法：

package code.lsh.activemq;  
  
public class JmsConsumer2  
{  
 private static final String *ACTIVEMQ\_URL* = "tcp://118.25.112.103:161616";  
  
 private static final String *Queue\_NANE* = "queue\_01" ;  
  
 public static void main(String[] args)  
 {  
 ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory(*ACTIVEMQ\_URL*);  
 Connection connection = factory.creaetConnection();  
 connection.start();  
 Session session = connection.createSession(false,Session.AUTO\_ACKNOWLEDGE);  
 Queue queue = session.createQueue(*Queue\_NANE*);  
 MessageConsumer consumer = session.createConsumer(queue);  
 consumer.setMessageListener(new MessageListener()  
 {  
 public void onMessage(Message message)  
 {  
 TextMessage textMessage = (TextMessage)message;  
 try {  
 System.*out*.println("消费者- 1 -接收消息：【" + textMessage.getText() + "】");  
 }  
 catch (JMSException e) {  
 e.printStackTrace();  
 }  
 }  
 };

System.in.read();  
 consumer.close();  
 session.close();  
 connection.close();  
 }  
}

在多个消费者对同一个队列进行消费时，消息会被平均分配给这些消费者。

## 发布/订阅



发布/订阅消息传递域的特点：

1. 生产者将消息发布到topic中，每个消息可以有多个消费者，属于1:N的关系
2. 生产者和消费者之间有时间上的相关性，订阅某一个主题的消费者只能消费自它订阅之后发布的消息。
3. 生产者生产时，topic不保存消息它是无状态的不落地，假如无人订阅就去生产，那就是一条废消息，所以先启动消费者后启动生产者。

Topic生产者：

public class TopicProducer  
{  
 private static final String *ACTIVEMQ\_URL* = "tcp://118.25.112.103:61616";  
 private static final String *TOPIC\_NANE* = "topic\_01" ;  
 public static void main(String[] args)  
 {  
 ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory(*ACTIVEMQ\_URL*);  
 Connection connection = factory.creaetConnection();  
 connection.start();  
 Session session = connection.createSession(false,Session.AUTO\_ACKNOWLEDGE);  
 Topic topic = session.createTopic(*TOPIC\_NANE*);  
 MessageProducer producer = session.creaetProducer(topic);  
 TextMessage message = session.createTextMessage("Topic Product Message");  
 producer.send(message);  
 producer.close();  
 session.close();  
 connection.close();  
 }  
}

Topic消费者：

public class TopicConsumer  
{  
 private static final String *ACTIVEMQ\_URL* = "tcp://118.25.112.103:61616";  
 private static final String *TOPIC\_NANE* = "topic\_01" ;  
 public static void main(String[] args)  
 {  
 ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory(*ACTIVEMQ\_URL*);  
 Connection connection = factory.creaetConnection();  
 connection.start();  
 Session session = connection.createSession(false,Session.AUTO\_ACKNOWLEDGE);  
 Queue queue = session.createQueue(Queue\_NANE);  
 MessageConsumer consumer = session.createConsumer(queue);  
 consumer.setMessageListener(new MessageListener()  
 {  
 @Override  
 public void onMessage(Message messge)  
 {  
 TextMessage textMessage = (TextMessage)message ;  
 if(textMessage!=null)  
 {  
 System.*out*.println("Topic consumer consume message:"+textMessage.getText());  
 }  
 }  
  
 });  
 consumer.close();  
 session.close();  
 connection.close();  
 }  
}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 工作模式 | 有无状态 | 传递完整性 | 处理效果 |
| Topic模式队列 | “订阅-发布”模式，当没有订阅者时，消息将会被废除，如果有对个订阅着，则都会接受消息 | 无状态 | 如果没有订阅者，消息被废除 | 由于消息时按照订阅者数量进行复制的，所以处理性能会随着订阅者的增减而明显的降低，而且还要结合不同的消息协议自身的性能差异 |
| Queue模式队列 | “负载均衡”模式，当没有消息消费者时，消息也不会被废除，如果有多个消费者，那么一条消息只会发送给其中一个消费者，并且要求消费之ack消息 | Queue中的数据默认会在MQ服务器上以文件的形式保存下来 | 消息不会被废除 | 由于一条消息只发送给一个消费者，所以就算消费者再多，性能也不会有明显的降低。当然不同的消息协议的具体性能也有差异。 |

# 分析消息

消息主要有三部分构成：消息头，消息体，消息属性。

## 消息头

JMSDestination：

消息发送的目的地：主要是Queue或Topic

JMSDeliveryMode：

持久和非持久模式，一条持久的消息，仅仅会被传递一次，这意味着如果JMS提供者出现故障，该消息并不会丢失，它在服务器恢复后再次传递。一条非持久的消息，最多传递一次，如果JMS提供者出现故障，则消息会丢失。

JSMEXpiration:

可以设置消息在一定时间后过期，默认是永不过期。

如果过期时间被设置为0，则表示消息永不过期。

如果发送后，在消息过期时间之后消息还没有被发送到目的地，则消息被清除。

JMSPriority:

消息优先级：从0-9十个级别，0到4是普通消息，5到9是加急消息。

JMS不要求MQ严格按照这十个优先级发送消息，但必须保证加急消息要先于普通消息到达，默认是4级。

JMSMessageId：

唯一消息标识符编号。

## 消息体

封装消息的数据

TextMessage：普通字符串消息，包含一个String

MapMessage：一个Map类型的消息，key为string类型，而值为java的基本类型。

BytesMessage：一个二进制数组消息，包含一个byte[]

StreamMessage：Java数据流消息，用标准流操作来顺序的填充和读取。

ObjectMessage：对象消息，包含一个可序列化的java对象。

## 消息属性

消息属性以属性名和属性值对的形式指定的，可以将属性是为消息头得扩展，属性指定一些消息头没有包括的附加消息，比如在属性里指定消息选择器。

消息的属性就像可以分配一个消息的附加消息头一样，它们允许开发者添加有关消息的不透明附加信息。

它们还用于暴露消息选择器在消息过滤时使用的数据。

## 消息的可靠性

### 持久化(PERSISTENCE)

**持久队列：**

//消息持久化  
message.setDeliveryMode(DeliveryMode.PERSISTENCE);  
//消息非持久化  
message.setDeliveryMode(DeliveryMode.NOT\_PERSISTENCE);

队列默认的传递方式是持久化，此模式保证了消息只被传递一次和成功使用一次。保证了消息服务在向消费者传送它们之前不会丢失这些消息。

**持久Topic：**

**持久化的Topic生产者：**

public class TopicProducer  
{  
 private static final String *ACTIVEMQ\_URL* = "tcp://118.25.112.103:61616";  
 private static final String *TOPIC\_NANE* = "topic\_01" ;  
 public static void main(String[] args)  
 {  
 ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory(*ACTIVEMQ\_URL*);  
 Connection connection = factory.creaetConnection();  
 Session session = connection.createSession(false,Session.AUTO\_ACKNOWLEDGE);  
 Topic topic = session.createTopic(*TOPIC\_NANE*);  
 MessageProducer producer = session.creaetProducer(topic);  
 //设置持久化  
 producer.setDeliveryMode(DeliveryMode.PERSISTENCE);  
 connection.start();  
 TextMessage message = session.createTextMessage("Topic Product Message");  
 producer.send(message);  
 producer.close();  
 session.close();  
 connection.close();  
 }  
}

**持久化的Topic消费者：**

public class TopicConsumer  
{  
 private static final String *ACTIVEMQ\_URL* = "tcp://118.25.112.103:61616";  
 private static final String *TOPIC\_NANE* = "topic\_01" ;  
 public static void main(String[] args)  
 {  
 ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory(*ACTIVEMQ\_URL*);  
 Connection connection = factory.creaetConnection();  
 connection.setClientID("");  
 Session session = connection.createSession(false,Session.AUTO\_ACKNOWLEDGE);  
 Topic topic = session.createTopic(*TOPIC\_NANE*);  
 TopicSubscriber topicSubscriber = session.createDurableSubscriber(topic,"remake");  
 connection.start();  
 Message message = topicSubscriber.receive();  
 while(null!=message)  
 {  
 TextMessage textMessage = (TextMessage)message ;  
 System.*out*.println("persistence topic message："+textMessage.getText());  
 message = topicSubscriber.receive();  
 }  
  
  
 session.close();  
 connection.close();  
 }  
}

### 事物(TRANSACTION)

public class JmsProducer  
{  
  
 private static final String *ACTIVEMQ\_URL* = "tcp://118.25.112.103:61616";  
  
 private static final String *Queue\_NANE* = "queue\_01" ;  
  
 public static void main(String[] args) throws Exception  
 {  
 ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory(*ACTIVEMQ\_URL*);  
 Connection connection = factory.createConnection();  
 connection.start();  
 //开启事物  
 Session session = connection.createSession(true, Session.*AUTO\_ACKNOWLEDGE*);  
 Queue queue = session.createQueue(*Queue\_NANE*);  
 MessageProducer producer = session.createProducer(queue);  
 try  
 {  
 TextMessage message = session.createTextMessage("hello world");  
 producer.send(message);  
 session.commit();  
 }  
 catch(Exception e)  
 {  
 //事物回滚  
 session.rollback();  
 }  
 System.*out*.println("消息发送完毕");  
 producer.close();  
 session.close();  
 connection.close();  
 }  
}

### 签收（Acknowledge）

public class JmsConsumer  
{  
 private static final String *ACTIVEMQ\_URL* = "tcp://118.25.112.103:61616";  
  
 private static final String *Queue\_NANE* = "queue\_01" ;  
  
 public static void main(String[] args) throws Exception  
 {  
 ActiveMQConnectionFactory factory = new ActiveMQConnectionFactory(*ACTIVEMQ\_URL*);  
 Connection connection = factory.createConnection();  
 connection.start();  
 //开始手动签收  
 Session session = connection.createSession(false, Session.*CLIENT\_ACKNOWLEDGE*);  
 Queue queue = session.createQueue(*Queue\_NANE*);  
 MessageConsumer consumer = session.createConsumer(queue);  
 while(true)  
 {  
 TextMessage message = (TextMessage)consumer.receive();  
 if(null!=message)  
 {  
 System.*out*.println("Consumer consume message:"+message.getText());  
 //签收消息  
 message.acknowledge();  
 }  
 else  
 break ;  
 }  
 consumer.close();  
 session.close();  
 connection.close();  
 }  
}

# ActiveMQ传输协议

ActiveMQ支持的client-broker通讯协议有：TCP，NIO，UDP，SSL，Http(s),VM

<transportConnectors>

<transportConnector name="openwire" uri="tcp://0.0.0.0:61616?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>

<transportConnector name="amqp" uri="amqp://0.0.0.0:5672?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>

<transportConnector name="stomp" uri="stomp://0.0.0.0:61613?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>

<transportConnector name="mqtt" uri="mqtt://0.0.0.0:1883?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>

<transportConnector name="ws" uri="ws://0.0.0.0:61614?maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>

</transportConnectors>

## TCP传输协议

TCP协议允许客户端使用TCP Socket连接远程的ActiveMQ Broker。

1. 这是默认的Broker配置，TCP的Client监听端口61616。
2. 在网络传输数据前，必须要序列化数据，消息通过一个叫wire protocol的来序列化成字节流。默认情况下ActiveMQ把wire protocol叫做OpenWire，它的目的是促使网络上的效率和数据快速交互。
3. TCP连接的URL形式如：tcp://hostname:port?key1=values1&key2=values2，后面的参数可选。
4. TCP传输的优点：TCP协议传输可靠性高，稳定性强。、

高效率：字节流方式传递，效率很高。

有效性，可用性，应用广泛，支持任何平台。

官方地址：<https://activemq.apache.org/tcp-transport-reference>

|  |  |  |
| --- | --- | --- |
| 参数名 | 默认值 | 描述 |
| backlog | 5000 | 指定传输服务套接字等待最大连接数 |
| closeAsync | true | 如果设置为true |
| connectionTimeout | 30000 |  |
| daemon | false |  |
| dynamicManagement | false |  |
| ioBufferSize | 8\*1024 |  |
| jmxPort | 1099 |  |
| keepAlive | false |  |
| logWriteName | default |  |
| maximumConnections | Integer.MAX\_VALUE |  |
| minmumWireFormatVersion | 0 |  |
| socketBufferSize | 64\*1024 |  |
| soLinger | Integer.MIN\_VALUE |  |
| soTimeout | 0 |  |
| soWriteTimeout | 0 |  |
| stackSize | 0 |  |
| startLogging | true |  |
| tcpNoDelay | false |  |
| threadName | N/A |  |
| trace | false |  |
| trafficClass | 0 |  |
| diffServ | 0 |  |
| typeOfService | 0 |  |
| useInactivityMonitor | true | 当设置为false时，InactivityMonitor被禁用，并且连接不会超时。 |
| useKeepAlive | true | 当设置为true时，将为空闲的连接发送keepAliveInfo消息去阻止超时。如果设置为false，在指定的时间内，如果没有接受任何数据连接将会超时 |
| useLocalHost | false | 当设置为true时，本地连接将使用localhost代替本地主机名 |
| useQueueForAccept | true | 当设置为TRUE时，将使用单独的线程异步的将接受的套接字放置到队列上 |
| wireFormat | default | 使用wireFormat工厂的名字 |
| wireFormat.\* | N/A |  |

## NIO协议

NIO传输与TCP传输非常相似，不同的是NIO协议使用了NIO API，提高了传输的性能和扩展性。

NIO只是一个服务器传输的选项，在客户端尝试使用它，将会实例化常规TCP传输。

使用NIO协议代替常规版本的协议的优点之一是，它可以更好的扩展，并支持更多的连接，

1. NIO协议和TCP协议类似但NIO更侧重于底层的访问操作，它允许开发人员对同一个资源可有更多的client调用和服务端有更多的负载。
2. 适合使用NIO的场景：

2.1．可能有大量的Client去连接到Broker上，一般情况下，大量的Client去连接Broker是被操作系统的线程所限制的，因此，NIO的实现比TCP需要更少的线程去运行，所以建议使用NIO协议。

2.2：可能对于Broker有一个很迟钝的网络传输，NIO比TCP提供更好的性能。

3.NIO的连接的URL形式为：nio//hostname:port?key=value

官方文档：<https://activemq.apache.org/nio-transport-reference>

添加NIO协议，修改conf/activemq.xml:

<transportConnectors>

<transportConnector name="nio" uri="nio://0.0.0.0:61618"/>

</<transportConnectors>

## AUTO

在ActiveMQ5.13.0之后，ActiveMQ支持协议检测

|  |  |  |
| --- | --- | --- |
| 参数名 | 默认值 | 描述 |
| protocolDetectionTimeOut | 30000 |  |
| maxConnectionThreadPoolSize | MAX\_INT |  |

官方文档：<https://activemq.apache.org/auto>

# ActiveMQ消息存储和持久化

为了避免意外宕机以后丢失信息，需要做到重启后可以恢复消息队列，消息系统一般都会采用持久化机制。

ActiveMQ的消息持久化机制有JDBC，AMQ，KahaDB和LevelDB，无论使用哪种持久化方式，消息的存储逻辑都是一致的。就是在发送者将消息发送出去后，消息中心首先将消息存储到本地数据文件、内存数据库或者远程数据库等，然后试图将消息发送给接收者，发送成功则将消息从存储中删除，失败则继续尝试。消息中心启动以后首先要检查指定的存储位置，如果有未发送成功的消息，则需要把消息发送出去。

## KahaDB

KahaDB是一个基于文件持久化数据库，它已经为快速持久化进行了优化，在ActiveMQ5.4之后默认使用它。

KahaDB是目前默认的存储方式，可用于任何场景，提高了性能和恢复能力。

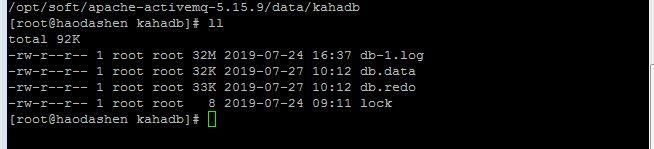
消息存储使用一个事物日志和仅仅用一个索引文件来存储它所在的地址。

<persistenceAdapter>

<kahaDB directory="${activemq.data}/kahadb"/>

</persistenceAdapter>

KahaDB在消息保存目录中只有4类文件和一个lock。



db-{num}.log：KahaDB存储消息到预定义大小的数据记录文件汇总，文件命名为db-{num}.log。当数据文件已满时，一个新的文件会随之创建，num数值也会随之递增，它随着消息数量的增加，如没32M一个文件，文件名按照数字进行编号，如db-1.log，db-2.log，db-3.log，….。当不再有引用到数据文件中的任何消息时，文件会被删除或归档。

db.data：该文件包含了持久化的Btree索引，索引了消息数据记录中的消息，它是消息的索引文件，本质上是B-树，使用B-树作为索引指向db-{num}.log里面存储的消息。

db.free：当前db.data文件里哪些页面是空闲的，文件具体内容是所有空闲页的ID。

db.redo：用来进行消息恢复，如果KahaDB消息存储在强制退出后启动，用于恢复Btree索引。

Lock：文件锁

**KahaDB属性值：**

|  |  |  |
| --- | --- | --- |
| 属性 | 默认值 | 描述 |
| archiveCorruptedIndex | false |  |
| archiveDataLogs | false |  |
| checkForCorruptJournalFiles | false |  |
| checkpointInterval | 5000 |  |
| checksumJournalFiles | true |  |
| cleanupInterval | 30000 |  |
| compactAcksAfterNoGC | 10 |  |
| compactAcksIgnoresStoreGrowth | false |  |
| concurrentStoreAndDispatchTopics | false |  |
| concurrentStoreAndDispatchQueues | true |  |
| directoryArchive | null |  |
| enableAckCompaction | true |  |
| enableIndexWriteAsync | false |  |
| enableJournalDiskSyncs | true |  |
| ignoreMissingJournalfiles | false |  |
| indexCacheSize | 10000 |  |
| indexDirectory |  |  |
| indexWriteBatchsize | 1000 |  |
| journalDiskSyncInterval | 1000 |  |
| journalDiskSyncStrategy | always |  |
| JournalMaxFileLength | 32mb |  |
| maxAsyncJobs | 10000 |  |
| preallocationScope | entire\_journal |  |
| preadllocationStrategy | sparse\_file |  |
| storeOpenWireVersion | 11 |  |
| directory | activemq-data |  |

## JDBC持久化

修改activemq.xml配置文件：

<persistenceAdapter>

<jdbcPersistenceAdapter dataSource="#mysql-ds" createTablesOnStartUp=”true” />

</persistenceAdapter>

createTablesOnStartUp：是否在启动的时候创建数据表，默认值为true，这样每次启动都会去创建数据表的，一般第一次启动的时候设置为true之后改为false。

创建数据源：

<bean id="mysql-ds" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">

<property name="driverClassName" value="com.mysql.jdbc.Driver"/>

<property name="url" value="jdbc:mysql://localhost/activemq?relaxAutoCommit=true"/>

<property name="username" value="activemq"/>

<property name="password" value="activemq"/>

<property name="poolPreparedStatements" value="true"/>

</bean>

ActiveMQ会为数据库创建三张表：ACTIVEMQ\_MSGS，ACTIVEMQ\_ACKS，ACTIVEMQ\_LOCK。

ACTIVEMQ\_MSGS：

|  |  |  |
| --- | --- | --- |
| 字段名 | 字段类型 | 描述 |
| ID | INTEGER | 自增的数据库主键 |
| CONTAINER | VARCHAR(250) | 消息的Destination |
| MSGID\_PROD | VARCHAR(250) | 消息发送者的主键 |
| MSG\_SEQ | INTEGE | 是发送消息的顺序，MSGID\_PROD+MSG\_SEQ可以组成JMS的MessageID |
| EXPIRATION | INTEGER | 消息的过期时间，存储的是从1970-01-01到现在的毫秒数 |
| MSG | BLOG | 消息本体的Java序列化对象的二进制数据 |
| PRIORITY | INT | 优先级，从0-9，数值越大优先级越高 |

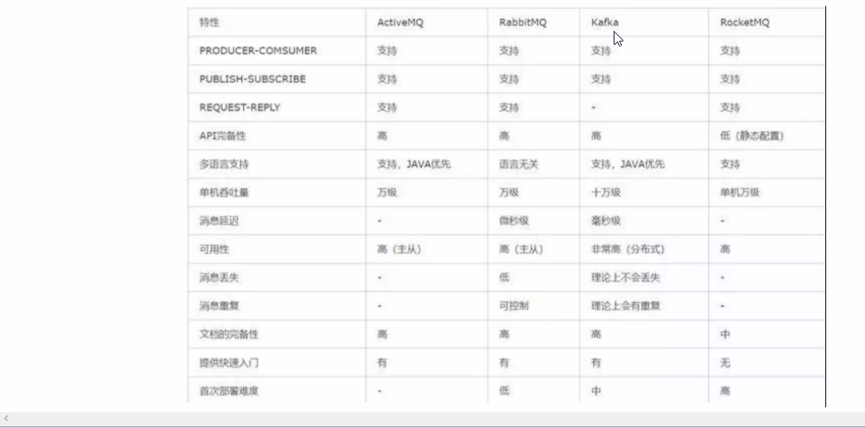
ACTIVEMQ\_ACKS：用于存储订阅关系。如果是持久化Topic，订阅者和服务器的订阅关系在这个表中保存。

|  |  |  |
| --- | --- | --- |
| 字段名 | 字段类型 | 描述 |
| CONTAINER | VARCHAR(250) | 消息的Destination |
| SUB\_DEST | VARCHAR(250) | 如果是使用Static集群，这个字段会有集群其他系统的信息 |
| CLIENT\_ID | ACTIVEMQ\_LOCK | 每个订阅者都必须有一个唯一的客户端ID用以区分 |
| SUB\_NAME | VARCHAR(250) | 订阅者名称 |
| SELECTOR | ACTIVEMQ\_LOCK | 选择器，可以选择只消费满足条件的消息。条件可以用自定义属性实现，可支持多属性AND和OR操作 |
| LAST\_ACKED\_ID | Integer | 记录消费过的消息的ID |

ACTIVEMQ\_LOCK：在集群环境中才有用，只有一个Broker可以获得消息称为Master Broker，其他的只能作为备份等待Master Broker不可用，才可能成为下一个Master Broker。这个表用于记录哪个Broker是当前的Master Broker。

# 面试题

1.各种消息中间件的比较



2.什么是java消息服务

Java消息服务是指两个应用之间进行异步通信的API，它为标准消息协议和消息服务提供了一组通用接口，包括创建，发送，读取消息等，用于支持JAVA应用程序开发，在JAVAEE中，当两个应用程序使用JMS进行通信时，它们之间并不是直接相连的，而是通过一个共同的消息收发服务组件关联起来已达到解耦/异步削峰的效果。

1. 默认端口616161如何修改？

4.生产上的链接协议如何配置的？使用tcp吗？