## 1. 背景

当前，CORBA、DCOM、RMI等RPC中间件技术已广泛应用于各个领域。但是面对规模和复杂度都越来越高的分布式系统，这些技术也显示出其局限性：

（1）同步通信：客户发出调用后，必须等待服务对象完成处理并返回结果后才能继续执行；（2）客户和服务对象的生命周期紧密耦合：客户进程和服务对象进程都必须正常运行；

如果由于服务对象崩溃或者网络故障导致客户的请求不可达，客户会接收到异常；

（3）点对点通信：客户的一次调用只发送给某个单独的目标对象。

面向消息的中间件（Message Oriented Middleware，MOM）较好的解决了以上问题。发送者将消息发送给消息服务器，消息服务器将消息存放在若干队列中，在合适的时候再将消息转发给接收者。这种模式下，

1）发送和接收是异步的，发送者无需等待；

2）二者的生命周期未必相同：发送消息的时候接收者不一定运行，接收消息的时候发送者

也不一定运行；

3）一对多通信：对于一个消息可以有多个接收者。

已有的 MOM系统包括 IBM的 MQSeries、Microsoft的MSMQ和BEA的MessageQ等。

由于没有一个通用的标准，这些系统很难实现互操作和无缝连接。

Java Message Service（JMS）是 SUN提出的旨在统一各种 MOM系统接口的规范，它包含点对点（Point to Point，PTP）和发布 /订阅（Publish/Subscribe，pub/sub）两种消息模型，提供可靠消息传输、事务和消息过滤等机制。

## 2．JMS概述

### 2.1 JMS规范

JAVA 消息服务(JMS)定义了 Java 中访问消息中间件的接口。JMS 只是接口，并没有给予实现，实现 JMS 接口的消息中间件称为 JMS Provider，例如 ActiveMQ。

### 2.2 术语

JMS Provider：实现JMS 接口的消息中间件（provider:供应者）；

PTP：Point to Point，即点对点的消息模型；

Pub/Sub：Publish/Subscribe，即发布/订阅的消息模型；

Queue：队列目标； Topic：主题目标；Transaction：事务。

ConnectionFactory：连接工厂，JMS 用它创建连接；

Connection：JMS 客户端到JMS Provider 的连接；

Destination：消息的目的地；Acknowledge：签收；

Session：会话，一个发送或接收消息的线程；

MessageProducer：由 Session 对象创建的用来发送消息的对象；

MessageConsumer：由 Session 对象创建的用来接收消息的对象;

### 2.3 JMS编程模型

在 JMS 编程模型中，JMS 客户端（组件或应用程序）通过 JMS 消息服务交换消息。

消息生产者将消息发送至消息服务，消息消费者则从消息服务接收这些消息。这些消息传送操作是使用一组实现 JMS 应用编程接口 (API) 的对象（由 JMS Provide提供）来执行的。在 JMS 编程模型中，JMS 客户端使用 ConnectionFactory 对象创建一个连接，向消息服务发送消息以及从消息服务接收消息均是通过此连接来进行。Connection 是客户端与消息服务的活动连接。创建连接时，将分配通信资源以及验证客户端。这是一个相当重要的对象，大多数客户端均使用一个连接来进行所有的消息传送。连接用于创建会话。Session 是一个用于生成和使用消息的单线程上下文。它用于创建发送的生产者和接收消息的消费者，并为所发送的消息定义发送顺序。会话通过大量确认选项或通过事务来支持可靠传送。客户端使用 MessageProducer 向指定的物理目标（在 API 中表示为目标身份对象）发送消息。生产者可指定一个默认传送模式（持久性消息与非持久性消息）、优先级和有效期值，以控制生产者向物理目标发送的所有消息。同样，客户端使用 MessageConsumer 对象从指定的物理目标（在 API 中表示为目标对象）接收消息。消费者可使用消息选择器，借助它，消息服务可以只向消费者发送与选择标准匹配的那些消息。消费者可以支持同步或异步消息接收。异步使用可通过向消费者注册 MessageListener 来实现。当会话线程调用 MessageListener 对象的 onMessage()方法时，客户端将使用消息。

### 2.4 JMS编程域

JMS 支持两种截然不同的消息传送模型：PTP（即点对点模型）和 Pub/Sub（即发布 /订阅模型），分别称作：PTP Domain 和 Pub/Sub Domain。

PTP（使用 Queue即队列目标）消息从一个生产者传送至一个消费者。在此传送模型中，目标是一个队列。消息首先被传送至队列目标，然后根据队列传送策略，从该队列将消息传送至向此队列进行注册的某一个消费者，一次只传送一条消息。可以向队列目标发送消息的生产者的数量没有限制，但每条消息只能发送至、并由一个消费者成功使用。如果没有已经向队列目标注册的消费者，队列将保留它收到的消息，并在某个消费者向该队列进行注册时将消息传送给该消费者。

Pub/Sub（使用Topic即主题目标）消息从一个生产者传送至任意数量的消费者。在此传送模型中，目标是一个主题。消息首先被传送至主题目标，然后传送至所有已订阅此主题的活动消费者。可以向主题目标发送消息的生产者的数量没有限制，并且每个消息可以发送至任意数量的订阅消费者。主题目标也支持持久订阅的概念。

持久订阅表示消费者已向主题目标进行注册，但在消息传送时此消费者可以处于非活动状态。当此消费者再次处于活动状态时，它将接收此信息。如果没有已经向主题目标注册的消费者，主题不保留其接收到的消息，除非有非活动消费者注册了持久订阅。这两种消息传送模型使用表示不同编程域的 API 对象（其语义稍有不同）进行处理，如下所示：

基本类型

|  |  |  |
| --- | --- | --- |
| 统一域 | PTP域 | Pub/Sub域 |
| ConnectionFactory | QueueConnectionFactory | TopicConnectionFactory |
| Connection | QueueConnection | TopicConnection |
| Session | QueueSession | TopicPublisher |
| Destination（Queue或 Topic） | Queue | Topic |
| MessageProducer | QueueSender |  |
| MessageConsumer | QueueReceiver QueueBrowser | TopicSubscriber |

### 2.5 JMS消息结构

JMS 消息由以下几部分组成：消息头，属性和消息体。

#### 2.5.1 消息头(Header)

消息头包含消息的识别信息和路由信息，消息头包含一些标准的属性

如：JMSDestination，JMSMessageID等。

|  |  |  |
| --- | --- | --- |
| 消息头 | 由谁设置 | 描述 |
| JMSDestination(目的地) | send方法 | 消息发送的目的地。 |
| JMSDeliveryMode(传送模式) | send方法 | 两种模式: PERSISTENT 和NON\_PERSISTENT，PERSISTENT 表示该消息一定要被送到目的地，否则会导致应用错误。NON\_PERSISTENT 表示偶 然丢失该消息是被允许的，这两种模式使开发者可以在消息传送的可靠性和吞吐量之间找到平衡点。 |
| JMSExpiration(消息过期时间) | send方法 | 等于 Destination 的send 方法中的timeToLive 值加上发送时刻的GMT 时间值。如果timeToLive 值等于零，则JMSExpiration 被设为零，表示该消息永不过期。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，则该消息被清除。 |
| JMSPriority(消息优先级) | send方法 | 从 0-9 十个级别，0-4 是普通消息，5-9 是加急消息。 |
| JMSMessageID(信息ID) | send方法 | 唯一识别每个消息的标识，由JMS Provider 产生。 |
| JMSTimestamp(时间戳) | send方法 | 一个消息被提交给JMS Provider 到消息被发出的时间。 |
| JMSCorrelationID(回复ID) | 客户 | 用来连接到另外一个消息，典型的应用是在回复消息中连接到原消息。 |
| JMSReplyTo（回复地址） | 客户 | 提供本消息回复消息的目的地址 |
| JMSType(类型) | 客户 | 消息类型的识别符。 |
| JMSRedelivered(再次发送) | JMS Provider | 如果一个客户端收到一个设置JMSRedelivered属性的消息，则表示可能客户端曾经在早些时候收到过该消息，但并没有签收(acknowledged)。 |

#### 2.5.2 属性(Properties)

除了消息头中定义好的标准属性外，JMS 提供一种机制增加新属性到消息头中，

这种新属性包含以下几种：

1.应用需要用到的属性;

2.消息头中原有的一些可选属性;

3. JMS Provider需要用到的属性。如:标准的JMS消息头包含以下属性：上图的10个属性

#### 2.5.3 消息体(Body)

JMS API 定义了 5种消息体格式，也叫消息类型，可以使用不同形式发送接收数据并可以兼容现有的消息格式，下面描述这 5种类型：

|  |  |
| --- | --- |
| 消息类型 | 消息体 |
| TextMessage | java.lang.String对象，如xml文件内容 |
| MapMessage | 键/值对的集合，键是 String对象，  值类型可以是Java任何基本类型 |
| BytesMessage | 字节流 |
| StreamMessage | Java中的输入输出流 |
| ObjectMessage | Java中的可序列化对象 |
| Message | 没有消息体，只有消息头和属性 |

### 2.6 PTP模型

PTP(Point-to-Point)模型是基于队列的，生产者发消息到队列，消费者从队列接收消息，队列的存在使得消息的异步传输成为可能。和邮件系统中的邮箱一样，队列可以包含各种消息，JMS Provider 提供工具管理队列的创建、删除。JMS PTP 模型定义了客户端如何向队列发送消息，从队列接收消息，浏览队列中的消息。下面描述 JMS PTP 模型中的主要概念和对象：

|  |  |
| --- | --- |
| 名称 | 描述 |
| ConnectionFactory | 客户端用 ConnectionFactory,创建Connection对象。 |
| Connection | 一个到 JMS Provider 的连接，客户端可以Connection 创建Session 来发送和接收消息。 |
| Session | 客户端用Session 创建MessageProducer 和MessageConsumer 对象。如果在Session 关闭时，有  一些消息已经被收到，但还没有被签(acknowledged)，那么，当消费者下次连接到相同的队列时，这些消息还会被再次接收。 |
| Destination（Queue或TemporaryQueue） | 客户端用 Session 创建Destination 对象。此处的目标为队列，队列由队列名识别。临时队列只能由创建它的Connection 所创建的消费者消费，但是任何生产者都可向临时队列发送消息。 |
| MessageProducer | 客户端用MessageProducer 发送消息到队列。 |
| MessageConsumer | 客户端用MessageConsumer 接收队列中的消息，如果用户在receive 方法中设定了消息选择条件，那么不符合条件的消息会留在队列中，不会被接收到。 |
| Reliability (可靠性) | 队列可以长久地保存消息直到消费者收到消息。消费者不需要因为担心消息会丢失而时刻和队列保持激活的连接状态，充分体现了异步传输模式的优势。 |

### 2.7 PUB/SUB模型

JMS Pub/Sub 模型定义了如何向一个内容节点发布和订阅消息，这些节点被称作主题(topic)。

主题可以被认为是消息的传输中介，发布者(publisher)发布消息到主题，订阅者(subscribe) 从主题订阅消息。主题使得消息订阅者和消息发布者保持互相独立，不需要接触即可保证消息的传送。下面描述 JMS Pub/Sub 模型中的主要概念和对象：

|  |  |
| --- | --- |
| 名称 | 描述 |
| subscription (订阅) | 消息订阅分为非持久订阅(non-durablesubscription)和持久订阅(durablesubscrip-tion)，非持久订阅只有当客户端处于激活状态，也就是和JMS Provider 保持连接状态才能收到发送到某个主题的消息，而当客户端处于离线状态，这个时间段发到主题的消息将会丢失，永远不会收到。持久订阅时，客户端向JMS 注册一个识别自己身份的ID，当这个客户端处于离线时，JMS Provider 会为这个ID 保存所有发送到主题的消息，当客户再次连接到JMS Provider 时，会根据自己的ID 得到所有当自己处于离线时发送到主题的消息。 |
| ConnectionFactory | 客户端用 ConnectionFactory 创建Connection对象。 |
| Connection | 一个到 JMS Provider 的连接，客户端可以用  Connection 创建Session 来发送和接收消息。 |
| Session | 客户端用Session 创建MessageProducer 和  MessageConsumer 对象。它还提供持久订阅主题，或使用unsubscribe 方法取消消息的持久订阅。 |
| Destination（Topic和TemporaryTopic） | 客户端用Session 创建Destination对象。此处的目标为主题，主题由主题名识别。临时主题只能由创建它的Connection 所创建的消费者消费。临时主题不能提供持久订阅功能。JMS 没有给出主题的组织和层次结构的定义，由JMS Provider 自己定义。 |
| MessageProducer | 客户端用MessageProducer 发布消息到主题。 |
| MessageConsumer | 客户端用MessageConsumer 接收发布到主题上  的消息。可以在receive 中设置消息过滤功能，这样，  不符合要求的消息不会被接收。 |
| Recovery and Redelivery  (恢复和重新派送) | 非持久订阅状态下，不能恢复或重新派送一个未  签收的消息。只有持久订阅才能恢复或重新派送一个未签收的消息。 |
| Reliability (可靠性) | 当所有的消息必须被接收，则用持久订阅模式。  当丢失消息能够被容忍，则用非持久订阅模式。 |

### 2.8 JMS支持并发

|  |  |
| --- | --- |
| JMS对象 | 是否支持并发 |
| Destination | 是 |
| ConnectionFactory | 是 |
| Connection | 是 |
| Session | 否 |
| MessageProducer | 否 |
| MessageConsumer | 否 |

## 3. ActiveMQ安装

### 3.1下载ActiveMQ

去官方网站下载：<http://activemq.apache.org/>

### 3.2运行ActiveMQ

解压缩apache-activemq-5.9.0-bin.zip，然后双击apache-activemq-5.9.0\bin\activemq.bat运行ActiveMQ程序。启动ActiveMQ以后，登陆：<http://localhost:8161/admin/>，

创建一个Queue，命名为FirstQueue。

## 4. ActiveMQ编程

### 4.1ActiveMQ特色：

本文档仅描述最基本的使用方法，实际使用过程中请以官方文档为准。

1．ActiveMQ官方网站：<http://activemq.apache.org>

2．JMS官方网站：<http://java.sun.com/products/jms>

### 4.2 开发 JSM的步骤

广义上说，一个JMS应用是几个 JMS 客户端交换消息，开发 JMS 客户端应用由以下几步构成：

1.用 JNDI 得到ConnectionFactory对象；

2.用 ConnectionFactory创建 Connection对象；

3.用 Connection 对象创建一个或多个JMS Session；

4.用JNDI 得到目标队列或主题对象，即Destination 对象；

5.用Session 和Destination 创建MessageProducer和MessageConsumer；

6.通知Connection开始传送消息。

### 4.3 编程模版

#### 4.3.1 ConnectionFactory（支持并发）

要初始化JMS，则需要使用连接工厂。客户端通过创建ConnectionFactory建立到ActveMQ的连接，一个连接工厂封装了一组连接配置参数，这组参数在配置ActiveMQ时已经定义，例如brokerURL参数，此参数传入的是ActiveMQ服务地址和端口，支持openwire协议的默认连接为tcp://localhost:61616，支持stomp协议的默认连接为tcp://localhost:61613。

注：由于C++客户端暂时仅支持stomp协议，所以需要使用 tcp://localhost:61613。

Java客户端：

ActiveMQConnectionFactory构造方法：

ActiveMQConnectionFactory();

ActiveMQConnectionFactory(String brokerURL);

ActiveMQConnectionFactory(String userName, String password, String brokerURL) ;

ActiveMQConnectionFactory(String userName, String password, URI brokerURL) ;

ActiveMQConnectionFactory(URI brokerURL); 其中brokerURL为ActiveMQ服务地址和端口。

例如：ActiveMQConnectionFactory connectionFactory =

new ActiveMQConnectionFactory("tcp://192.168.0.135:61616");

或者 ActiveMQConnectionFactory connectionFactory = new ActiveMQConnectionFactory();

connectionFactory. setBrokerURL("tcp://192.168.0.135:61616");

#### 4.3.2 Connection（支持并发）

在成功创建正确的ConnectionFactory后，下一步将是创建一个连接，它是JMS定义的一个接口。ConnectionFactory负责返回可以与底层消息传递系统进行通信的 Connection实现。通常客户端只使用单一连接。根据JMS文档，Connection的目的是“利用JMS提供者封装开放的连接”，以及表示“客户端与提供者服务例程之间的开放TCP /IP套接字”。该文档还指出 Connection应该是进行客户端身份验证的地方，除了其他一些事项外，客户端还可以指定惟一标志符。当一个 Connection被创建时，它的传输默认是关闭的，必须使用start()方法开启。一个 Connection可以建立一个或多个的Session。当一个程序执行完成后，必须关闭之前创建的Connection，否则ActiveMQ不能释放资源，关闭一个Connection同样也关闭Session，MessageProducer和MessageConsumer。

##### 4.3.2.1 创建Connection

Java客户端：

ActiveMQConnectionFactory方法：

Connection createConnection()；

Connection createConnection(String userName, String password);

例如：Connection connection = connectionFactory.createConnection();

##### 4.3.2.2 开启Connection

Java客户端：

ActiveMQConnection方法：

void start();

例如：Connection.start();

##### 4.3.2.3 关闭Connection

Java客户端：

ActiveMQConnection方法：

void close();

例如：Connection.close();

#### 4.3.3 Session

一旦从ConnectionFactory中获得一个Connection，就必须从Connection中创建一个或者多个Session。Session是一个发送或接收消息的线程，可以使用Session创建MessageProducer，MessageConsumer和Message。Session可以被事务化，也可以不被事务化，通常，可以通过向 Connection上的适当创建方法传递一个布尔参数对此进行设置。

Java客户端：

ActiveMQConnection方法：

Session createSession(boolean transacted, int acknowledgeMode);

其中transacted为使用事务标识，acknowledgeMode为签收模式。

例如：Session session = connection.createSession(false, Session.AUTO\_ACKNOWLEDGE);

#### 4.3.4 Destination

Destination是一个客户端用来指定生产消息目标和消费消息来源的对象。

在PTP模式中，Destination被称作Queue即队列；

在Pub/Sub模式，Destination被称作Topic即主题。在程序中可以使用多个 Queue和Topic。

Java客户端：

ActiveMQSession方法：

Queue createQueue(String queueName);

TemporaryQueue createTemporaryQueue();

Topic createTopic(String topicName);

TemporaryTopic createTemporaryTopic();

例如：Destination destination = session.createQueue("TEST.FOO");

或者 Destination destination = session.createTopic("TEST.FOO");

#### 4.3.5 MessageProducer

MessageProducer是一个由Session创建的对象，用来向Destination发送消息。

##### 4.2.5.1 创建 MessageProducer

Java客户端：

ActiveMQSession方法：

MessageProducer createProducer(Destination destination);

例如：MessageProducer producer = session.createProducer(destination);

##### 4.3.5.2 发送消息

Java客户端：

ActiveMQMessageProducer方法：

void send(Destination destination, Message message);

void send(Destination destination, Message message, int deliveryMode, int priority,long timeToLive);

void send(Message message);

void send(Message message, int deliveryMode, int priority, long timeToLive);

其中deliveryMode为传送模式，priority为消息优先级，timeToLive为消息过期时间。

例如：producer.send(message);

#### 4.3.6 MessageConsumer

MessageConsumer是一个由Session创建的对象，用来从Destination接收消息。

##### 4.2.6.1 创建 MessageConsumer

Java客户端：

ActiveMQSession方法：

MessageConsumer createConsumer(Destination destination);

MessageConsumer createConsumer(Destination destination, String messageSelector);

MessageConsumer createConsumer(Destination destination, String messageSelector, boolean noLocal);

TopicSubscriber createDurableSubscriber(Topic topic, String name);

TopicSubscriber createDurableSubscriber(Topic topic, String name, String messageSelector,

boolean noLocal);

其中messageSelector为消息选择器；noLocal标志默认为false，当设置为true时限制消费者只能接收和自己相同的连接（Connection）所发布的消息，此标志只适用于主题，不适用于队列；name标识订阅主题所对应的订阅名称，持久订阅时需要设置此参数。

例如：MessageConsumer consumer = session.createConsumer(destination);

##### 4.3.6.2消息的同步和异步接收

消息的同步接收是指客户端主动去接收消息，客户端可以采用MessageConsumer的receive()方法去接收下一个消息。消息的异步接收是指当消息到达时，ActiveMQ主动通知客户端。客户端可以通过注册一个实现MessageListener 接口的对象到MessageConsumer。MessageListener只有一个必须实现的方法 onMessage()，它只接收一个参数，即Message。在为每个发送到Destination的消息实现onMessage时，将调用该方法。

Java客户端：

ActiveMQMessageConsumer方法：

Message receive()

Message receive(long timeout)

Message receiveNoWait()

其中timeout为等待时间，单位为毫秒。或者实现MessageListener接口，每当消息到达时，ActiveMQ会调用MessageListener中的onMessage()函数。

例如：Message message = consumer.receive();

##### 4.3.6.3消息选择器

JMS提供了一种机制，使用它，消息服务可根据消息选择器中的标准来执行消息过滤。生产者可在消息中放入应用程序特有的属性，而消费者可使用基于这些属性的选择标准来表明对消息是否感兴趣。这就简化了客户端的工作，并避免了向不需要这些消息的消费者传送消息的开销。然而，它也使得处理选择标准的消息服务增加了一些额外开销。

消息选择器是用于MessageConsumer的过滤器，可以用来过滤传入消息的属性和消息头部分（但不过滤消息体），并确定是否将实际消费该消息。按照JMS文档的说法，消息选择器是一些字符串，它们基于某种语法，而这种语法是SQL-92的子集。可以将消息选择器作为MessageConsumer创建的一部分。

Java客户端：

例如：public final String SELECTOR = “JMSType = ‘TOPIC\_PUBLISHER’”;

该选择器检查了传入消息的 JMSType属性，并确定了这个属性的值是否等于

TOPIC \_PUBLISHER。如果相等，则消息被消费；如果不相等，那么消息会被忽略。

#### 4.3.7 Message

JMS程序的最终目的是生产和消费的消息能被其他程序使用，JMS的Message是一个既简单又不乏灵活性的基本格式，允许创建不同平台上符合非JMS程序格式的消息。Message由以下几部分组成：消息头，属性和消息体。

Java客户端：

ActiveMQSession方法：

BlobMessage createBlobMessage(File file)

BlobMessage createBlobMessage(InputStream in)

BlobMessage createBlobMessage(URL url)

BlobMessage createBlobMessage(URL url, boolean deletedByBroker)

BytesMessage createBytesMessage()

MapMessage createMapMessage()

Message createMessage()

ObjectMessage createObjectMessage()

ObjectMessage createObjectMessage(Serializable object)

TextMessage createTextMessage()

TextMessage createTextMessage(String text)

例如：

下例演示创建并发送一个TextMessage到一个队列：

TextMessage message = queueSession.createTextMessage();

message.setText(msg\_text); // msg\_text is a String

queueSender.send(message);

下例演示接收消息并转换为合适的消息类型：

Message m = queueReceiver.receive();

if (m instanceof TextMessage) {

TextMessage message = (TextMessage) m;

System.out.println("Reading message: " + message.getText());

} else {

// Handle error

}

### 4.4 可靠性机制

发送消息最可靠的方法就是在事务中发送持久性的消息，ActiveMQ默认发送持久性消息。结束事务有两种方法：提交或者回滚。当一个事务提交，消息被处理。如果事务中有一个步骤失败，事务就回滚，这个事务中的已经执行的动作将被撤销。接收消息最可靠的方法就是在事务中接收信息，不管是从PTP模式的非临时队列接收消息还是从Pub/Sub模式持久订阅中接收消息。对于其他程序，低可靠性可以降低开销和提高性能，例如发送消息时可以更改消息的优先级或者指定消息的过期时间。消息传送的可靠性越高，需要的开销和带宽就越多。性能和可靠性之间的折衷是设计时要重点考虑的一个方面。可以选择生成和使用非持久性消息来获得最佳性能。另一方面，也可以通过生成和使用持久性消息并使用事务会话来获得最佳可靠性。在这两种极端之间有许多选择，这取决于应用程序的要求。

#### 4.4.1 基本可靠性机制

##### 4.4.1.1 控制消息的签收（Acknowledgment）

客户端成功接收一条消息的标志是这条消息被签收。成功接收一条消息一般包括

如下三个阶段：

1. 客户端接收消息；2．客户端处理消息；3．消息被签收。

签收可以由ActiveMQ发起，也可以由客户端发起，取决于Session签收模式的设置。在带事务的Session中，签收自动发生在事务提交时。如果事务回滚，所有已经接收的消息将会被再次传送。在不带事务的Session中，一条消息何时和如何被签收取决于Session的设置。

1．Session.AUTO\_ACKNOWLEDGE

当客户端从receive或onMessage成功返回时，Session自动签收客户端的这条消息的收条。在AUTO\_ACKNOWLEDGE的Session中，同步接收receive是上述三个阶段的一个例外，在这种情况下，收条和签收紧随在处理消息之后发生。

2．Session.CLIENT\_ACKNOWLEDGE

客户端通过调用消息的acknowledge方法签收消息。在这种情况下，签收发生在Session层面：签收一个已消费的消息会自动地签收这个Session所有已消费消息的收条。

3．Session.DUPS\_OK\_ACKNOWLEDGE

此选项指示Session不必确保对传送消息的签收。它可能引起消息的重复，但是降低了Session的开销，所以只有客户端能容忍重复的消息，才可使用（如果ActiveMQ再次传送同一消息那么消息头中的JMSRedelivered将被设置为true）。

Java客户端：

签收模式分别为：

1．Session.AUTO\_ACKNOWLEDGE

2．Session.CLIENT\_ACKNOWLEDGE

3．Session.DUPS\_OK\_ACKNOWLEDGE

ActiveMQConnection方法：

Session createSession(boolean transacted, int acknowledgeMode);

例如：Session session = connection.createSession(false, Session.AUTO\_ACKNOWLEDGE);

对队列来说，如果当一个Session终止时它接收了消息但是没有签收，那么ActiveMQ将保留这些消息并将再次传送给下一个进入队列的消费者。对主题来说，如果持久订阅用户终止时，它已消费未签收的消息也将被保留，直到再次传送给这个用户。对于非持久订阅，AtiveMQ在用户Session关闭时将删除这些消息。如果使用队列和持久订阅，并且Session没有使用事务，那么可以使用Session的recover方法停止Session，再次启动后将收到它第一条没有签收的消息，事实上，重启后Session一系列消息的传送都是以上一次最后一条已签收消息的下一条为起点。如果这时有消息过期或者高优先级的消息到来，那么这时消息的传送将会和最初的有所不同。对于非持久订阅用户，重启后，ActiveMQ有可能删除所有没有签收的消息。

##### 4.4.1.2 指定消息传送模式

ActiveMQ支持两种消息传送模式：PERSISTENT和NON\_PERSISTENT两种。

1．PERSISTENT（持久性消息）

这是ActiveMQ的默认传送模式，此模式保证这些消息只被传送一次和成使用一次。对于这些消息，可靠性是优先考虑的因素。可靠性的另一个重要方面是确保持久性消息传送至目标后，消息服务在向消费者传送它们之前不会丢失这些消息。这意味着在持久性消息传送至目标时，消息服务将其放入持久性数据存储。如果消息服务由于某种原因导致失败，它可以恢复此消息并将此消息传送至相应的消费者。虽然这样增加了消息传送的开销，但却增加了可靠性。

2．NON\_PERSISTENT（非持久性消息）

保证这些消息最多被传送一次。对于这些消息，可靠性并非主要的考虑因素。此模式并不要求持久性的数据存储，也不保证消息服务由于某种原因导致失败后消息不会丢失。有两种方法指定传送模式：

1．使用 setDeliveryMode方法，这样所有的消息都采用此传送模式；

2．使用 send方法为每一条消息设置传送模式；

Java客户端：

传送模式分别为：

1．DeliveryMode.PERSISTENT

2．DeliveryMode.NON\_PERSISTENT

ActiveMQMessageProducer方法：

void setDeliveryMode(int newDeliveryMode);

或者

void send(Destination destination, Message message, int deliveryMode, int priority, long timeToLive);

void send(Message message, int deliveryMode, int priority, long timeToLive);

其中deliveryMode为传送模式，priority为消息优先级，timeToLive为消息过期时间。

例如：producer.setDeliveryMode(DeliveryMode.NON\_PERSISTENT);

如果不指定传送模式，那么默认是持久性消息。如果容忍消息丢失，那么使用非持久性消息可以改善性能和减少存储的开销。

##### 4.3.1.3 设置消息优先级

通常，可以确保将单个会话向目标发送的所有消息按其发送顺序传送至消费者。然而，如果为这些消息分配了不同的优先级，消息传送系统将首先尝试传送优先级较高的消息。

有两种方法设置消息的优先级：

1．使用 setDeliveryMode方法，这样所有的消息都采用此传送模式；

2．使用 send方法为每一条消息设置传送模式；

Java客户端：

ActiveMQMessageProducer方法：

void setPriority(int newDefaultPriority);

或者

void send(Destination destination, Message message, int deliveryMode, int priority, long timeToLive);

void send(Message message, int deliveryMode, int priority, long timeToLive);

其中deliveryMode为传送模式，priority为消息优先级，timeToLive为消息过期时间。

例如：producer.setPriority(4);

消息优先级从0-9十个级别，0-4是普通消息，5-9是加急消息。如果不指定优先级，则默认为4。JMS不要求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。

##### 4.4.1.4 允许消息过期

默认情况下，消息永不会过期。如果消息在特定周期内失去意义，那么可以设置过期时间。

有两种方法设置消息的过期时间，时间单位为毫秒：

1．使用 setTimeToLive方法为所有的消息设置过期时间；

2．使用 send方法为每一条消息设置过期时间；

Java客户端：

ActiveMQMessageProducer方法：

void setTimeToLive(long timeToLive);

或者

void send(Destination destination, Message message, int deliveryMode, int priority, long timeToLive);

void send(Message message, int deliveryMode, int priority, long timeToLive);

其中deliveryMode为传送模式，priority为消息优先级，timeToLive为消息过期时间。

例如：producer.setTimeToLive(1000);

消息过期时间，send方法中的timeToLive 值加上发送时刻的GMT时间值。如果 timeToLive值等于零，则JMSExpiration 被设为零，表示该消息永不过期。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，则该消息被清除。

##### 4.4.1.5 创建临时目标

ActiveMQ通过createTemporaryQueue和createTemporaryTopic创建临时目标，这些目标持续到创建它的Connection关闭。只有创建临时目标的Connection所创建的客户端才可以从临时目标中接收消息，但是任何的生产者都可以向临时目标中发送消息。如果关闭了创建此目标的Connection，那么临时目标被关闭，内容也将消失。

Java客户端：

ActiveMQSession方法：

TemporaryQueue createTemporaryQueue();

TemporaryTopic createTemporaryTopic();

某些客户端需要一个目标来接收对发送至其他客户端的消息的回复。这时可以使用临时目标。Message的属性之一是JMSReplyTo属性，这个属性就是用于这个目的的。可以创建一个临时的Destination，并把它放入Message的JMSReplyTo属性中，收到该消息的消费者可以用它来响应生产者。

Java客户端：

如下所示代码段，将创建临时的Destination，并将它放置在TextMessage的JMSReplyTo属性中：Destination tempQueue = session.createTemporaryQueue();

msg.setJMSReplyTo(tempQueue);

消费者接收这条消息时，会从JMSReplyTo字段中提取临时Destination，并且会通过应用程序构造一个MessageProducer，以便将响应消息发送回生产者。这展示了如何使用 JMS Message的属性，并显示了私有的临时Destination的有用之处。它还展示了客户端可以既是消息的生产者，又可以是消息的消费者。

Destination tempQueue = msg.getJMSReplyTo();

MessageProducer Sender = session. createProducer(tempQueue);

TextMessage msg = session.createTextMessage();

msg.setText(REPLYTO\_TEXT);

sender.send(msg);

#### 4.4.2 高级可靠性机制

##### 4.4.2.1 创建持久订阅

通过为发布者设置PERSISTENT传送模式，为订阅者时使用持久订阅，这样可以保证Pub/Sub程序接收所有发布的消息。消息订阅分为非持久订阅(non-durable subscription)和持久订阅(durable subscription)，非持久订阅只有当客户端处于激活状态，也就是和ActiveMQ保持连接状态才能收到发送到某个主题的消息，而当客户端处于离线状态，这个时间段发到主题的消息将会丢失，永远不会收到。持久订阅时，客户端向ActiveMQ注册一个识别自己身份的ID，当这个客户端处于离线时，ActiveMQ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ActiveMQ时，会根据自己的ID得到所有当自己处于离线时发送到主题的消息。持久订阅会增加开销，同一时间在持久订阅中只有一个激活的用户。

建立持久订阅的步骤：

1．为连接设置一个客户ID；

2．为订阅的主题指定一个订阅名称；上述组合必须唯一。

###### 4.4.2.1.1 创建持久订阅

Java客户端：

ActiveMQConnection方法：

void setClientID(String newClientID)和ActiveMQSession方法：

TopicSubscriber createDurableSubscriber(Topic topic, String name)

TopicSubscriber createDurableSubscriber(Topic topic, String name, String messageSelector, boolean noLocal)

其中messageSelector为消息选择器；noLocal标志默认为false，当设置为true时限制消费者只能接收和自己相同的连接（Connection）所发布的消息，此标志只适用于主题，不适用于队列;name标识订阅主题所对应的订阅名称，持久订阅时需要设置此参数。

###### 4.4.2.1.2 删除持久订阅

Java客户端：

ActiveMQSession方法：

void unsubscribe(String name);

##### 4.4.2.2 使用本地事务

在事务中生成或使用消息时，ActiveMQ跟踪各个发送和接收过程，并在客户端发出提交事务的调用时完成这些操作。如果事务中特定的发送或接收操作失败，则出现异常。客户端代码通过忽略异常、重试操作或回滚整个事务来处理异常。在事务提交时，将完成所有成功的操作。在事务进行回滚时，将取消所有成功的操作。本地事务的范围始终为一个会话。也就是说，可以将单个会话的上下文中执行的一个或多个生产者或消费者操作组成一个本地事务。

不但单个会话可以访问Queue 或 Topic（任一类型的 Destination ），而且单个会话实例可以用来操纵一个或多个队列以及一个或多个主题，一切都在单个事务中进行。这意味着单个会话可以（例如）创建队列和主题中的生产者，然后使用单个事务来同时发送队列和主题的消息。因为单个事务跨越两个目标，所以，要么队列和主题的消息都得到发送，要么都未得到发送。类似地，单个事务可以用来接收队列中的消息并将消息发送到主题上，反过来也可以。由于事务的范围只能为单个的会话，因此不存在既包括消息生成又包括消息使用的端对端事务。（换句话说，至目标的消息传送和随后进行的至客户端的消息传送不能放在同一个事务中。）

###### 4.4.2.2.1 使用事务

Java客户端：

ActiveMQConnection方法：

Session createSession(boolean transacted, int acknowledgeMode);

其中transacted为使用事务标识，acknowledgeMode为签收模式。

例如：Session session = connection.createSession(true, Session.AUTO\_ACKNOWLEDGE);

###### 4.3.2.2.2 提交

Java客户端：

ActiveMQSession方法：

void commit()；

例如：

try {

producer.send(consumer.receive());

session.commit();

} catch (JMSException ex) {

session.rollback();

}

###### 4.3.2.2.3 回滚

Java客户端：

ActiveMQSession方法：

void rollback();

### 4.5 高级特征

#### 4.5.1 异步发送消息

ActiveMQ支持生产者以同步或异步模式发送消息。使用不同的模式对send方法的反应时间有巨大的影响，反映时间是衡量ActiveMQ吞吐量的重要因素，使用异步发送可以提高系统的性能。在默认大多数情况下，AcitveMQ是以异步模式发送消息。例外的情况：在没有使用事务的情况下，生产者以PERSISTENT传送模式发送消息。在这种情况下，send方法都是同步的，并且一直阻塞直到ActiveMQ发回确认消息：消息已经存储在持久性数据存储中。这种确认机制保证消息不会丢失，但会造成生产者阻塞从而影响反应时间。高性能的程序一般都能容忍在故障情况下丢失少量数据。如果编写这样的程序，可以通过使用异步发送来提高吞吐量（甚至在使用PERSISTENT传送模式的情况下）。

Java客户端：

使用

Connection URI配置异步发送:

cf = new ActiveMQConnectionFactory("tcp://locahost:61616?jms.useAsyncSend=true");

在 ConnectionFactory层面配置异步发送:

((ActiveMQConnectionFactory)connectionFactory).setUseAsyncSend(true);

在 Connection层面配置异步发送，此层面的设置将覆盖ConnectionFactory层面的设置:

((ActiveMQConnection)connection).setUseAsyncSend(true);

#### 4.5.2 消费者特色

##### 4.5.2.1 消费者异步分派

在 ActiveMQ4中，支持ActiveMQ以同步或异步模式向消费者分派消息。这样的意义：

可以以异步模式向处理消息慢的消费者分配消息；以同步模式向处理消息快的消费者分配消息。ActiveMQ默认以同步模式分派消息，这样的设置可以提高性能。但是对于处理消息慢的消费者，需要以异步模式分派。

Java客户端：

在 ConnectionFactory层面配置同步分派:

((ActiveMQConnectionFactory)connectionFactory).setDispatchAsync(false);

在 Connection层面配置同步分派，此层面的设置将覆盖ConnectionFactory层面

的设置：((ActiveMQConnection)connection).setDispatchAsync(false);

在消费者层面以Destination URI配置同步分派，此层面的设置将覆盖ConnectionFactory和

Connection层面的设置:

queue = new ActiveMQQueue("TEST.QUEUE?consumer.dispatchAsync=false");

consumer = session.createConsumer(queue);

##### 4.5.2.2 消费者优先级

在ActveMQ分布式环境中，在有消费者存在的情况下，如果更希望

ActveMQ发送消息给消费者而不是其他的ActveMQ到ActveMQ的传送，可以如下设置：

Java客户端：

queue = new ActiveMQQueue("TEST.QUEUE?consumer.prority=10");

consumer = session.createConsumer(queue);

##### 4.5.2.3 独占的消费者

ActiveMQ维护队列消息的顺序并顺序把消息分派给消费者。但是如果建立了多个Session和MessageConsumer，那么同一时刻多个线程同时从一个队列中接收消息时就并不能保证处理时有序。有时候有序处理消息是非常重要的。ActiveMQ4支持独占的消费。ActiveMQ挑选一个MessageConsumer，并把一个队列中所有消息按顺序分派给它。如果消费者发生故障，那么ActiveMQ将自动故障转移并选择另一个消费者。可以如下设置：

Java客户端：

queue = new ActiveMQQueue("TEST.QUEUE?consumer.exclusive=true");

consumer = session.createConsumer(queue);

##### 4.5.2.4 再次传送策略

在以下三种情况中，消息会被再次传送给消费者：

1．在使用事务的Session中，调用rollback()方法；

2．在使用事务的Session中，调用commit()方法之前就关闭了Session;

3．在Session中使用CLIENT\_ACKNOWLEDGE签收模式，并且调用了recover()方法。可以通过设置ActiveMQConnectionFactory和ActiveMQConnection来定制想要的再次传送策略。

|  |  |
| --- | --- |
| 属性 | 默认值 |
| collisionAvoidanceFactor | 0.15 |
| maximumRedeliveries | 6 |
| initialRedeliveryDelay | 1000L |
| useCollisionAvoidance | false |
| useExponentialBackOff | false |
| backOffMultiplier | 5 |

#### 4.5.3 目标特色

##### 4.5.3.1 复合目标

在1.1版本之后，ActiveMQ支持混合目标技术。它允许在一个JMS目标中使用一组 JMS目标。例如可以利用混合目标在同一操作中用向12个队列发送同一条消息或者在同一操作中向一个主题和一个队列发送同一条消息。在混合目标中，通过“,”来分隔不同的目标。

Java客户端：

例如：Queue queue = new ActiveMQQueue("FOO.A,FOO.B,FOO.C");

producer.send(queue, someMessage);

如果在一个目标中混合不同类别的目标，可以通过使用“queue://”和“topic://”前缀来识别不同的目标。

例如：Queue queue = new ActiveMQQueue("FOO.A,topic://NOTIFY.FOO.A");

producer.send(queue, someMessage);

##### 4.5.3.2 目标选项

Java客户端：

例如：

|  |  |
| --- | --- |
| 属性 | 默认值 |
| consumer.prefetchSize | variable |
| consumer.maximumPendingMess  ageLimit | 0 |
| consumer.noLocal | false |
| consumer.dispatchAsync | false |
| consumer.retroactive | fasle |
| consumer.selector | null |
| consumer.exclusive | false |
| consumer.priority | 0 |

Queue =

new ActiveMQQueue("TEST.QUEUE?consumer.dispatchAsync=false&consumer.prefetchSize=10");

consumer = session.createConsumer(queue);

#### 4.5.4 消息预取

ActiveMQ的目标之一就是高性能的数据传送，所以ActiveMQ使用“预取限制”来控制有多少消息能及时的传送给任何地方的消费者。一旦预取数量达到限制，那么就不会有消息被分派给这个消费者直到它发回签收消息（用来标识所有的消息已经被处理）。可以为每个消费者指定消息预取。如果有大量的消息并且希望更高的性能，那么可以为这个消费者增大预取值。如果有少量的消息并且每条消息的处理都要花费很长的时间，那么可以设置预取值为1，这样同一时间，ActiveMQ只会为这个消费者分派一条消息。

Java客户端：

在ConnectionFactory层面为所有消费者配置预取值:

tcp://localhost:61616?jms.prefetchPolicy.all=50

在 ConnectionFactory层面为队列消费者配置预取值:

tcp://localhost:61616?jms.prefetchPolicy.queuePrefetch=1

使用“目标选项”为一个消费者配置预取值:

queue = new ActiveMQQueue("TEST.QUEUE?consumer.prefetchSize=10");

consumer = session.createConsumer(queue);

#### 4.5.5 配置连接 URL

ActiveMQ支持通过Configuration URI明确的配置连接属性。

例如：当要设置异步发送时，可以通过在

Configuration URI中使用 jms.$PROPERTY来设置。

tcp://localhost:61616?jms.useAsyncSend=true

以下的选项在URI必须以“jms.”为前缀。

|  |  |
| --- | --- |
| 属性 | 默认值 |
| alwaysSessionAsync | true |
| clientID | null |
| closeTimeout | 15000(毫秒) |
| copyMessageOnSend | true |
| disableTimeStampsByDefault | fasle |
| dispatchAsync | fasle |
| nestedMapAndListEnabled | true |
| objectMessageSerializationDe  fered | fasle |
| optimizeAcknowledge | false |
| optimizedMessageDispatch | true |
| useAsyncSend | false |
| useCompression | false |
| useRetroactiveConsumer | false |

### 4.6 优化

优化部分请参阅：<http://devzone.logicblaze.com/site/how-to-tune-activemq.html>

## 6. ActiveMQ配置

### 6.1 配置文件

ActiveMQ配置文件：$AcrtiveMQ/conf/activemq.xml

### 6.2 配置

ActiveMQ服务IP和端口

<transportConnectors>

<transportConnector name="openwire" uri="tcp://localhost:61616"

discoveryUri="mu lticast://default"/>

<transportConnector name="ssl" uri="ssl://localhost:61617"/>

<transportConnector name="stomp" uri="stomp://localhost:61613"/>

</transportConnectors>

在transportConnectors标识中配置ActiveMQ服务IP和端口，其中name属性指定协议的名称，uri属性指定协议所对应的协议名，IP地址和端口号。上述IP地址和端口可以根据实际需要指定。Java客户端默认使用openwire协议，所以ActiveMQ服务

地址为tcp://localhost:61616；

目前C++客户端仅支持stomp协议，所以ActiveMQ服务地址为tcp://localhost:61613。

### 6.3 分布式部署

分布式部署请参阅：<http://activemq.apache.org/networks-of-brokers.html>

### 6.4 监控

ActiveMQ本节将使用JXM和JXM控制台（JDK1.5控制台）监控ActiveMQ。

#### 6.4.1 配置 JXM

<broker brokerName="emv219" useJmx="true" xmlns="http://activemq.org/config/1.0">

<managementContext>

<managementContext connectorPort="1099" jmxDomainName="org.apache.activemq"/>

</managementContext>

</broker>

配置JXM步骤如下：

1．设置broker标识的useJmx属性为true；

2．取消对managementContext标识的注释（系统默认注释 managementContext标识），监控的默认端口为1099。

#### 6.4.2 在Windows平台监控

进入%JAVA\_HOME%/bin，双击jconsole.exe即出现如下画面，在对话框中输入ActiveMQ服务主机的地址，JXM的端口和主机登陆帐号。

## 7. 目前存在问题

### 7.1 C++客户端丢失消息问题

ActiveMQ版本：ActiveMQ 4.1.1SNAPSHOT

C++客户端版本：ActiveMQ CPP 1.1 Release

测试中发现，当 C++客户端异常退出时（即没有正常调用close函数关闭连接），ActiveMQ并不能检测到C++客户端的连接已经中断，这时如果向队列中发送消息，那么第一条消息就会丢失，这时ActiveMQ才能检测到这个连接是中断的。在ActiveMQ论坛反应此问题后，开发人员答复并建议使用CLIENT\_ACKNOWLEDGE签收模式。但是此模式会造成消息重复接收。测试ActiveMQ 4.2SNAPSHOT时并未发现上述问题。

### 7.2 队列消息堆积过多后有可能阻塞程序

默认activemq.xml中配置的内存是20M，这就意味着当消息堆积超过20M后，程序可能出现问题。在mial list中其他用户对此问题的描述是：send方法会阻塞或抛出异常。

### 7.3 目前版本的 C++客户端仅支持 stomp协议

目前版本的 C++客户端程序（ActiveMQ CPP 1.1 Release）仅支持stomp协议，因此传输消息的速度应该没有使用openwire协议的Java客户端快。ActiveMQ网站显示不久将会有支持openwire协议的C++客户端程序发布。

### 7.4 分布式部署问题

ActiveMQ版本：ActiveMQ 4.1.1SNAPSHOT和ActiveMQ 4.2SNAPSHOT

测试选用上述两个未正式发布的版本，未选用正式发布的ActiveMQ 4.1.0 Release版本是因为此版本bug较多。在测试中发现，如果重启其中一台机器上的ActiveMQ，其他机器的

ActiveMQ有可能会打印：java.io.EOFException

at java.io.DataInputStream.readInt(DataInputStream.java:358)

at org.apache.activemq.openwire.OpenWireFormat.unmarshal(OpenWireFormat.java:267)

at org.apache.activemq.transport.tcp.TcpTransport.readCommand(TcpTransport.java:156)

at org.apache.activemq.transport.tcp.TcpTransport.run(TcpTransport.java:136)

at java.lang.Thread.run(Thread.java:595)

WARN TransportConnection - Unexpected extra broker info com

mand received: BrokerInfo {commandId = 6, responseRequired = false,

brokerId = ID:emv219n-33945-1174458770157-1:0, brokerURL = tcp://emv219n:61616, sl

aveBroker = false, masterBroker = false, faultTolerantConfiguration = false,

networkConnection = false, duplexConnection = false, peerBrokerInfos = [],

brokerName = emv219, connectionId = 0}.

INFO FailoverTransport - Transport failed, attempting to

automatically reconnect due to: java.io.EOFException。

这时分布式的消息传输就会出现问题，此问题目前还没找到原因。