ActiveMQ in Action

作者: whitesock http://whitesock.javaeye.com

我的博客文章精选

目 录

1. EE

1.1 ActiveMQ in Action(1)

发表时间: 2008-02-25 关键字: activemq

1 JMS

在介绍ActiveMQ之前，首先简要介绍一下JMS规范。

1．1 JMS的基本构件

1．1．1 连接工厂

连接工厂是客户用来创建连接的对象，例如ActiveMQ提供的ActiveMQConnectionFactory。

1．1．2 连接

JMS Connection封装了客户与JMS提供者之间的一个虚拟的连接。

1．1．3 会话

JMS Session是生产和消费消息的一个单线程上下文。会话用于创建消息生产者（producer）、消息消费者（consumer）和消息（message）等。会话提供了一个事务性的上下文，在这个上下文中，一组发送和接收被组合到了一个原子操作中。

1．1．4 目的地

目的地是客户用来指定它生产的消息的目标和它消费的消息的来源的对象。JMS1.0.2规范中定义了两种消息传递域：点对点（PTP）消息传递域和发布/订阅消息传递域。

点对点消息传递域的特点如下：

- 每个消息只能有一个消费者。
- 消息的生产者和消费者之间没有时间上的相关性。无论消费者在生产者发送消息的时候是否处于运行状态，它都可以提取消息。

发布/订阅消息传递域的特点如下：

- 每个消息可以有多个消费者。
- 生产者和消费者之间有时间上的相关性。订阅一个主题的消费者只能消费自它订阅之后发布的消息。JMS规范允许客户创建持久订阅，这在一定程度上放松了时间上的相关性要求。持久订阅允许消费者消费它在未处于激活状态时发送的消息。

在点对点消息传递域中，目的地被成为队列（queue）；在发布/订阅消息传递域中，目的地被成为主题（topic）。

1．1．5 消息生产者

消息生产者是由会话创建的一个对象，用于把消息发送到一个目的地。

1．1．6 消息消费者

消息消费者是由会话创建的一个对象，它用于接收发送到目的地的消息。消息的消费可以采用以下两种方法之一：

- 同步消费。通过调用消费者的receive方法从目的地中显式提取消息。receive方法可以一直阻塞到消息到达。
- 异步消费。客户可以为消费者注册一个消息监听器，以定义在消息到达时所采取的动作。

1．1．7 消息

JMS消息由以下三部分组成：

- 消息头。每个消息头字段都有相应的getter和setter方法。
- 消息属性。如果需要除消息头字段以外的值，那么可以使用消息属性。
- 消息体。JMS定义的消息类型有TextMessage、MapMessage、BytesMessage、StreamMessage和ObjectMessage。

1．2 JMS的可靠性机制

1．2．1 确认

JMS消息只有在被确认之后，才认为已经被成功地消费了。消息的成功消费通常包含三个阶段：客户接收消息、客户处理消息和消息被确认。

在事务性会话中，当一个事务被提交的时候，确认自动发生。在非事务性会话中，消息何时被确认取决于创建会话时的应答模式（acknowledgement mode）。该参数有以下三个可选值：

- Session.AUTO_ACKNOWLEDGE。当客户成功的从receive方法返回的时候，或者从MessageListener.onMessage方法成功返回的时候，会话自动确认客户收到的消息。
- Session.CLIENT_ACKNOWLEDGE。客户通过消息的acknowledge方法确认消息。需要注意的是，在这种模式中，确认是在会话层上进行：确认一个被消费的消息将自动确认所有已被会话消费的消息。例如，如果一个消息消费者消费了10个消息，然后确认第5个消息，那么所有10个消息都被确认。
- Session.DUPS_ACKNOWLEDGE。该选择只是会话迟钝第确认消息的提交。如果JMS provider失败，那么可能会导致一些重复的消息。如果是重复的消息，那么JMS provider必须把消息头的JMSRedelivered字段设置为true。

1．2．2 持久性

JMS 支持以下两种消息提交模式：

- PERSISTENT。指示JMS provider持久保存消息，以保证消息不会因为JMS provider的失败而丢失。

- NON_PERSISTENT。不要求JMS provider持久保存消息。

1．2．3 优先级

可以使用消息优先级来指示JMS provider首先提交紧急的消息。优先级分10个级别，从0（最低）到9（最高）。如果不指定优先级，默认级别是4。需要注意的是，JMS provider并不一定保证按照优先级的顺序提交消息。

1．2．4 消息过期

可以设置消息在一定时间后过期，默认是永不过期。

1．2．5 临时目的地

可以通过会话上的createTemporaryQueue方法和createTemporaryTopic方法来创建临时目的地。它们的存在时间只限于创建它们的连接所保持的时间。只有创建该临时目的地的连接上的消息消费者才能够从临时目的地中提取消息。

1．2．6 持久订阅

首先消息生产者必须使用PERSISTENT提交消息。客户可以通过会话上的createDurableSubscriber方法来创建一个持久订阅，该方法的第一个参数必须是一个topic。第二个参数是订阅的名称。

JMS provider会存储发布到持久订阅对应的topic上的消息。如果最初创建持久订阅的客户或者任何其它客户使用相同的连接工厂和连接的客户ID、相同的主题和相同的订阅名再次调用会话上的createDurableSubscriber方法，那么该持久订阅就会被激活。JMS provider会象客户发送客户处于非激活状态时所发布的消息。

持久订阅在某个时刻只能有一个激活的订阅者。持久订阅在创建之后会一直保留，直到应用程序调用会话上的unsubscribe方法。

1．2．7 本地事务

在一个JMS客户端，可以使用本地事务来组合消息的发送和接收。JMS Session接口提供了commit和rollback方法。事务提交意味着生产的所有消息被发送，消费的所有消息被确认；事务回滚意味着生产的所有消息被销毁，消费的所有消息被恢复并重新提交，除非它们已经过期。

事务性的会话总是牵涉到事务处理中，commit或rollback方法一旦被调用，一个事务就结束了，而另一个事务被开始。关闭事务性会话将回滚其中的事务。

需要注意的是，如果使用请求/回复机制，即发送一个消息，同时希望在同一个事务中等待接收该消息的回复，那么程序将被挂起，因为知道事务提交，发送操作才会真正执行。

需要注意的还有一个，消息的生产和消费不能包含在同一个事务中。

1．3 JMS 规范的变迁

JMS的最新版本的是1.1。它和同1.0.2版本之间最大的差别是，JMS1.1通过统一的消息传递域简化了消息传递。这不仅简化了JMS API，也有利于开发人员灵活选择消息传递域，同时也有助于程序的重用和维护。
以下是不同消息传递域的相应接口：

JMS 公共	点对点域	发布/订阅域
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
Connection	QueueConnection	TopicConnection
Destination	Queue	Topic
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver	TopicSubscriber

2 ActiveMQ
2．1 Broker
2．1．1 Running Broker
ActiveMQ5.0 的二进制发布包中bin目录中包含一个名为activemq的脚本，直接运行这个脚本就可以启动一个broker。
此外也可以通过Broker Configuration URI或Broker XBean URI对broker进行配置，以下是一些命令行参数的例子：

Example	Description
activemq	Runs a broker using the default 'xbean:activemq.xml' as the broker configuration file.
activemq xbean:myconfig.xml	Runs a broker using the file myconfig.xml as the broker configuration file that is located in the classpath.
activemq xbean:file:./conf/broker1.xml	Runs a broker using the file broker1.xml as the broker configuration file that is located in the relative file path ./conf/broker1.xml
activemq xbean:file:C:/ActiveMQ/conf/broker2.xml	Runs a broker using the file broker2.xml as the broker configuration file that is located in the absolute file path C:/ActiveMQ/conf/broker2.xml
activemq broker:(tcp://localhost:61616, tcp://localhost:5000)?useJmx=true	Runs a broker with two transport connectors and JMX enabled.

activemq broker:(tcp://localhost:61616, network:tcp://localhost:5000)?persistent=false	Runs a broker with 1 transport connector and 1 network connector with persistence disabled.

2．1．2 Embedded Broker

可以通过在应用程序中以编码的方式启动broker，例如：

```
BrokerService broker = new BrokerService();
broker.addConnector("tcp://localhost:61616");
broker.start();
```

如果需要启动多个broker，那么需要为broker设置一个名字。例如：

```
BrokerService broker = new BrokerService();
broker.setName("fred");
broker.addConnector("tcp://localhost:61616");
broker.start();
```

如果希望在同一个JVM内访问这个broker，那么可以使用VM Transport，URI是：vm://brokerName。关于更多的broker属性，可以参考Apache的官方文档。

此外，也可以通过BrokerFactory来创建broker，例如：

```
BrokerService broker = BrokerFactory.createBroker(new URI(someURI));
```

someURI的可选值如下：

URI scheme	Example	Description
xbean:	xbean:activemq.xml	Searches the classpath for an XML document with the given URI (activemq.xml in this case) which will then be used as the Xml Configuration
file:	file:foo/bar/activemq.xml	Loads the given file (in this example foo/bar/activemq.xml) as the Xml Configuration
broker:	broker:tcp://localhost:61616	Uses the Broker Configuration URI to configure the broker

当使用XBean的配置方式的时候，需要指定一个xml配置文件，例如：

```
BrokerService broker = BrokerFactory.createBroker(new URI("xbean:com/test/activemq.xml"));
```

使用Spring的配置方式如下：

```
<bean id="broker" class="org.apache.activemq.xbean.BrokerFactoryBean">
  <property name="config" value="classpath:org/apache/activemq/xbean/activemq.xml" />
  <property name="start" value="true" />
</bean>
```

2．1．3 Monitoring Broker

2．1．3．1 JMX

在使用JMX监控broker之前，首先要启用broker的JMX监控功能，例如在配置文件中设置useJmx="true"，如下：

```
<broker useJmx="true" brokerName="broker1>
  <managementContext>
     <managementContext createConnector="true"/>
  </managementContext>
  ...
</broker>
```

接下来运行JDK自带的jconsole。在运行了jconsole后，它会弹出对话框来选择需要连接到的agent。如果是在启动broker的主机上运行jconsole，那么ActiveMQ broker会出现在jconsole的Local 标签中。如果要连接到远程的broker，那么可以在Advanced标签中指定JMX URL，以下是一个连接到本机的JMX URL：

service:jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi

在jconsole的MBeans标签中，可以查看详细信息，也可以执行相应的operation。需要注意的是，在jconsole连接到broker的时候，并不需要输入用户名和密码，如果这存在潜在的安全问题，那么就需要为JMX Connector配置密码保护（需要使用1.5以上版本的JDK）。

首先要禁止ActiveMQ创建自己的connector，例如：

```
<broker xmlns="http://activemq.org/config/1.0" brokerName="localhost"useJmx="true">
  <managementContext>
     <managementContext createConnector="false"/>
  </managementContext>
</broker>
```

然后在ActiveMQ的conf目录下创建一个访问控制文件和密码文件，如下：

conf/jmx.access：

The "monitorRole" role has readonly access.
The "controlRole" role has readwrite access.
monitorRole readonly
controlRole readwrite

conf/jmx.password：
The "monitorRole" role has password "abc123".
The "controlRole" role has password "abcd1234".
monitorRole abc123
controlRole abcd1234

然后修改ActiveMQ的bin目录下activemq的启动脚本，查找包含"SUNJMX="的一行如下：

REM set SUNJMX=-Dcom.sun.management.jmxremote.port=1616 -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false

把它替换成

set SUNJMX=-Dcom.sun.management.jmxremote.port=1616 -Dcom.sun.management.jmxremote.authenticate=true -Dcom.sun.management.jmxremote.ssl=false -Dcom.sun.management.jmxremote.password.file=%ACTIVEMQ_BASE%/conf/jmx.password -Dcom.sun.management.jmxremote.access.file=%ACTIVEMQ_BASE%/conf/jmx.access

最后重启ActiveMQ和jconsole，这时候需要强制login。如果在启动activemq的过程中出现以下错误，那么需要为这个文件增加访问控制。Windows平台上的具体解决方法请参考如下网址：http://java.sun.com/j2se/1.5.0/docs/guide/management/security-windows.html

Error: Password file read access must be restricted: D:\apache-activemq-5.0.0\bin\../conf/jmx.password

2．1．3．2 Web Console

Web Console被集成到了ActiveMQ的二进制发布包中，因此缺省访问http://localhost:8161/admin即可访问Web Console。

在配置文件中，可以通过修改nioConnector的port属性来修改Web console的缺省端口：

```
<jetty xmlns="http://mortbay.com/schemas/jetty/1.0">
  <connectors>
    <nioConnector port="8161" />
```

```
  </connectors>
  ...
</jetty>
```

出于安全性或者可靠性的考虑，Web Console 可以被部署到不同于ActiveMQ的进程中。例如把activemq-web-console.war部署到一个单独的web容器中（Tomcat，Jetty等）。在ActiveMQ5.0的二进制发布包中不包含activemq-web-console.war，因此需要下载ActiveMQ的源码，然后进入到${activemq.base}/src/activemq-web-console目录中执行mvn instanll。如果一切正常，那么缺省会在${activemq.base}/src/activemq-web-console/target目录中生成activemq-web-console-5.0.0.war。然后将activemq-web-console-5.0.0.war拷贝到Tomcat的webapps目录中，并重命名成activemq-web-console.war。

需要注意的是，要将activemq-all-5.0.0.jar拷贝到WEB-INF\lib目录中（可能还需要拷贝jms.jar）。还要为Tomcat设置以下五个系统属性（修改catalina.bat文件）：

```
set JAVA_OPTS=%JAVA_OPTS% -Dwebconsole.type="properties"
set JAVA_OPTS=%JAVA_OPTS% -Dwebconsole.jms.url="tcp://localhost:61616"
set JAVA_OPTS=%JAVA_OPTS% -Dwebconsole.jmx.url="service:jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi"
set JAVA_OPTS=%JAVA_OPTS% -Dwebconsole.jmx.role=""
set JAVA_OPTS=%JAVA_OPTS% -Dwebconsole.jmx.password=""
```

如果JMX没有配置密码保护，那么webconsole.jmx.role和webconsole.jmx.password设置成""即可。如果broker被配置成了Master/Slave模式，那么可以配置成使用failover transport，例如：

```
-Dwebconsole.jms.url=failover:(tcp://serverA:61616,tcp://serverB:61616)
```

顺便说一下，由于webconsole.type 属性是properties，因此实际上起作用的Web Console的配置文件是WEB-INF/ webconsole-properties.xml。最后启动被监控的ActiveMQ，访问http://localhost:8080/activemq-web-console/，查看显示是否正常。

2．1．3．3 Advisory Message

ActiveMQ 支持Advisory Messages，它允许你通过标准的JMS 消息来监控系统。目前的Advisory Messages支持：

- consumers, producers and connections starting and stopping
- temporary destinations being created and destroyed
- messages expiring on topics and queues
- brokers sending messages to destinations with no consumers.

- connections starting and stopping

Advisory Messages可以被想象成某种的管理通道，通过它你可以得到关于JMS provider、producers、consumers和destinations的信息。Advisory topics都使用ActiveMQ.Advisory.这个前缀，以下是目前支持的topics：

Client based advisories

Advisory Topics	**Description**
ActiveMQ.Advisory.Connection	Connection start & stop messages
ActiveMQ.Advisory.Producer.Queue	Producer start & stop messages on a Queue
ActiveMQ.Advisory.Producer.Topic	Producer start & stop messages on a Topic
ActiveMQ.Advisory.Consumer.Queue	Consumer start & stop messages on a Queue
ActiveMQ.Advisory.Consumer.Topic	Consumer start & stop messages on a Topic

在消费者启动/停止的Advisory Messages的消息头中有个consumerCount属性，他用来指明目前desination上活跃的consumer的数量。

Destination and Message based advisories

Advisory Topics	**Description**
ActiveMQ.Advisory.Queue	Queue create & destroy
ActiveMQ.Advisory.Topic	Topic create & destroy
ActiveMQ.Advisory.TempQueue	Temporary Queue create & destroy
ActiveMQ.Advisory.TempTopic	Temporary Topic create & destroy
ActiveMQ.Advisory.Expired.Queue	Expired messages on a Queue
ActiveMQ.Advisory.Expired.Topic	Expired messages on a Topic
ActiveMQ.Advisory.NoConsumer.Queue	No consumer is available to process messages being sent on a Queue
ActiveMQ.Advisory.NoConsumer.Topic	No consumer is available to process messages being sent on a Topic

以上的这些destnations都可以用来作为前缀，在其后面追加其它的重要信息，例如topic、queue、clientID、producderID和consumerID等。这令你可以利用Wildcards 和 Selectors 来过滤Advisory Messages（关于Wildcard和Selector会在稍后介绍）。

例如，如果你希望订阅FOO.BAR这个queue上Consumer的start/stop的消息，那么可以订阅ActiveMQ.Advisory.Consumer.Queue.FOO.BAR；如果希望订阅所有queue上的start/stop消息，那么可以订阅ActiveMQ.Advisory.Consumer.Queue.>；如果希望订阅所有queue或者topic上的start/stop消息，那么可以订阅ActiveMQ.Advisory.Consumer. >。

org.apache.activemq.advisory.AdvisorySupport类上有如下的helper methods，用来在程序中得到advisory destination objects。

```
AdvisorySupport.getConsumerAdvisoryTopic()
AdvisorySupport.getProducerAdvisoryTopic()
AdvisorySupport.getDestinationAdvisoryTopic()
AdvisorySupport.getExpiredTopicMessageAdvisoryTopic()
AdvisorySupport.getExpiredQueueMessageAdvisoryTopic()
AdvisorySupport.getNoTopicConsumersAdvisoryTopic()
AdvisorySupport.getNoQueueConsumersAdvisoryTopic()
```

以下是段使用Advisory Messages的程序代码：

```
Destination advisoryDestination = AdvisorySupport.getProducerAdvisoryTopic(destination)
MessageConsumer consumer = session.createConsumer(advisoryDestination);
consumer.setMessageListener(this);
...
public void onMessage(Message msg){
    if (msg instanceof ActiveMQMessage){
        try {
             ActiveMQMessage aMsg =  (ActiveMQMessage)msg;
             ProducerInfo prod = (ProducerInfo) aMsg.getDataStructure();
        } catch (JMSException e) {
            log.error("Failed to process message: " + msg);
        }
    }
}
```

2．1．3．4 Command Agent

在介绍Command Agent前首先简要介绍一下XMPP(Jabber)协议，XMPP是一种基于XML的即时通信协议，它由Jabber软件基金会开发。在配置文件中通过增加transportConnector来支持XMPP协议：

```
<broker xmlns="http://activemq.org/config/1.0">
  <transportConnectors>
     ...
     <transportConnector name="xmpp"     uri="xmpp://localhost:61222"/>
```

```
  </transportConnectors>
</broker>
```

ActiveMQ提供了ActiveMQ messages和XMPP之间的双向桥接：

- 如果客户加入了一个聊天室，那么这个聊天室的名字会被映射到一个JMS topic。
- 尝试在聊天室内发送消息会导致一个JMS消息被发送到这个topic。
- 呆在一个聊天室中意味着这将保持一个对相应JMS topic的订阅。因此发送到这个topic的JMS消息也会被发送到聊天室。

推荐XMPP客户端Spark(http://www.igniterealtime.org/)。

从4.2版本起，ActiveMQ支持Command Agent。在配置文件中，通过设置commandAgent来启用Command Agent：

```
<beans>
  <broker useJmx="true" xmlns="http://activemq.org/config/1.0">
    ...
  </broker>
  <commandAgent xmlns="http://activemq.org/config/1.0"/>
</beans>
```

启用了Command Agent的broker上会有一个来自Command Agent的连接，它同时订阅topic：ActiveMQ.Agent。在你启动XMPP客户端，加入到ActiveMQ.Agent聊天室后，就可以同broker进行交谈了。通过在XMPP客户端中键入help，可以得到帮助信息。

需要注意的是，ActiveMQ5.0版本有个小bug，如果broker没有采用缺省的用户名和密码，那么Command Agent便无法正常启动。Apache官方文档说，此bug已经被修正，预定在5.2.0版本上体现。修改方式如下：

```
<commandAgent xmlns="http://activemq.org/config/1.0" brokerUser="user" brokerPassword="passwarc
```

2．1．3．5 Visualization plugin

ActiveMQ支持以broker插件的形式生成DOT文件(可以用agrviewer来查看)，以图表的方式描述connections、sessions、producers、consumers、destinations等信息。配置方式如下：

```
<broker xmlns="http://activemq.org/config/1.0" brokerName="localhost" useJmx="true">
      ...
      <plugins>
          <connectionDotFilePlugin  file="connection.dot"/>
```

```
          <destinationDotFilePlugin file="destination.dot"/>
      </plugins>
</broker>
```

需要注意的是，笔者认为ActiveMQ5.0版本的Visualization Plugin尚不稳定，存在诸多问题。例如：如果使用connectionDotFilePlugin，那么brokerName必须是localhost；如果使用destinationDotFilePlugin可能会导致ArrayStoreException。

1.2 ActiveMQ in Action(2)

发表时间: 2008-02-25 关键字: activemq

2．2 Transport

ActiveMQ目前支持的transport有：VM Transport、TCP Transport、SSL Transport、Peer Transport、UDP Transport、Multicast Transport、HTTP and HTTPS Transport、Failover Transport、Fanout Transport、Discovery Transport、ZeroConf Transport等。以下简单介绍其中的几种，更多请参考Apache官方文档。

2．2．1 VM Transport

VM transport允许在VM内部通信，从而避免了网络传输的开销。这时候采用的连接不是socket连接，而是直接地方法调用。 第一个创建VM 连接的客户会启动一个embed VM broker，接下来所有使用相同的broker name的VM连接都会使用这个broker。当这个broker上所有的连接都关闭的时候，这个broker也会自动关闭。

以下是配置语法：

vm://brokerName?transportOptions

例如：vm://broker1?marshal=false&broker.persistent=false

Transport Options的可选值如下：

Option Name	Default Value	Description
Marshal	false	If true, forces each command sent over the transport to be marshlled and unmarshlled using a WireFormat
wireFormat	default	The name of the WireFormat to use
wireFormat.*		All the properties with this prefix are used to configure the wireFormat
create	true	If the broker should be created on demand if it does not allready exist. Only supported in ActiveMQ 4.1
broker.*		All the properties with this prefix are used to configure the broker. See Configuring Wire Formats for more information

以下是高级配置语法：

vm:(broker:(tcp://localhost)?brokerOptions)?transportOptions

vm:broker:(tcp://localhost)?brokerOptions

例如：vm:(broker:(tcp://localhost:6000)?persistent=false)?marshal=false

Transport Options的可选值如下：

Option Name	Default Value	Description
marshal	false	If true, forces each command sent over the transport to be marshlled and unmarshlled using a WireFormat
wireFormat	default	The name of the WireFormat to use
wireFormat.*		All the properties with this prefix are used to configure the wireFormat

使用配置文件的配置语法：

vm://localhost?brokerConfig=xbean:activemq.xml

例如：vm:// localhost?brokerConfig=xbean:com/test/activemq.xml

使用Spring的配置：

```
<bean id="broker" class="org.apache.activemq.xbean.BrokerFactoryBean">
  <property name="config" value="classpath:org/apache/activemq/xbean/activemq.xml" />
  <property name="start" value="true" />
</bean>

<bean id="connectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory" depends-on='
  <property name="brokerURL" value="vm://localhost"/>
</bean>
```

如果persistent是true，那么ActiveMQ会在当前目录下创建一个缺省值是activemq-data的目录用于持久化保存数据。需要注意的是，如果程序中启动了多个不同名字的VM broker，那么可能会有如下警告：Failed to start jmx connector: Cannot bind to URL [rmi://localhost:1099/jmxrmi]: javax.naming.NameAlreadyBoundException...可以通过在transportOptions中追加broker.useJmx=false来禁用JMX来避免这个警告。

2．2．2 TCP Transport

TCP transport 允许客户端通过TCP socket连接到远程的broker。以下是配置语法：

tcp://hostname:port?transportOptions

Transport Options的可选值如下：

Option Name	Default Value	Description
minmumWireFormatVersion	0	The minimum version wireformat that is allowed
trace	false	Causes all commands that are sent over the transport to be logged
useLocalHost	true	When true, it causes the local machines name to resolve to "localhost".
socketBufferSize	64 * 1024	Sets the socket buffer size in bytes
soTimeout	0	sets the socket timeout in milliseconds
connectionTimeout	30000	A non-zero value specifies the connection timeout in milliseconds. A zero value means wait forever for the connection to be established. Negative values are ignored.
wireFormat	default	The name of the WireFormat to use
wireFormat.*		All the properties with this prefix are used to configure the wireFormat. See Configuring Wire Formats for more information

例如：tcp://localhost:61616?trace=false

2．2．3 Failover Transport

Failover Transport是一种重新连接的机制，它工作于其它transport的上层，用于建立可靠的传输。它的配置语法允许制定任意多个复合的URI。Failover transport会自动选择其中的一个URI来尝试建立连接。如果没有成功，那么会选择一个其它的URI来建立一个新的连接。以下是配置语法：

failover:(uri1,...,uriN)?transportOptions

failover:uri1,...,uriN

Transport Options的可选值如下：

Option Name	Default Value	Description
initialReconnectDelay	10	How long to wait before the first reconnect attempt (in ms)

maxReconnectDelay	30000	The maximum amount of time we ever wait between reconnect attempts (in ms)
useExponentialBackOff	true	Should an exponential backoff be used between reconnect attempts
backOffMultiplier	2	The exponent used in the exponential backoff attempts
maxReconnectAttempts	0	If not 0, then this is the maximum number of reconnect attempts before an error is sent back to the client
randomize	true	use a random algorithm to choose the URI to use for reconnect from the list provided
backup	false	initialize and hold a second transport connection - to enable fast failover

例如：failover:(tcp://localhost:61616,tcp://remotehost:61616)?initialReconnectDelay=100

2．2．4 Discovery transport

Discovery transport是可靠的tranport。它使用Discovery transport来定位用来连接的URI列表。以下是配置语法：

discovery:(discoveryAgentURI)?transportOptions

discovery:discoveryAgentURI

Transport Options的可选值如下：

Option Name	**Default Value**	**Description**
initialReconnectDelay	10	How long to wait before the first reconnect attempt
maxReconnectDelay	30000	The maximum amount of time we ever wait between reconnect attempts
useExponentialBackOff	true	Should an exponential backoff be used btween reconnect attempts
backOffMultiplier	2	The exponent used in the exponential backoff attempts
maxReconnectAttempts	0	If not 0, then this is the maximum number of reconnect attempts before an error is sent back to the client

例如：discovery:(multicast://default)?initialReconnectDelay=100

为了使用Discovery来发现broker，需要为broker启用discovery agent。 以下是XML配置文件中的一个例子：

```
<broker name="foo">
   <transportConnectors>
      <transportConnector uri="tcp://localhost:0" discoveryUri="multicast://default"/>
    </transportConnectors>
    ...
</broker>
```

在使用Failover Transport或Discovery transport等能够自动重连的transport的时候，需要注意的是：设想有两个broker，它们都启用AMQ Message Store作为持久化存储，有一个producer和一个consumer连接到某个queue。当因其中一个broker失效时而切换到另一个broker的时候，如果失效的broker的queue中还有未被consumer消费的消息，那么这个queue里的消息仍然滞留在失效broker的中，直到失效的broker被修复并重新切换回这个被修复的broker后，之前被保留的消息才会被consumer消费掉。如果被处理的消息有时序限制，那么应用程序就需要处理这个问题。另外也可以通过ActiveMQ集群来解决这个问题。

在transport重连的时候，可以在connection上注册TransportListener来获得回调，例如：

```
(ActiveMQConnection)connection).addTransportListener(new TransportListener() {
    public void onCommand(Object cmd) {
    }

    public void onException(IOException exp) {
    }

    public void transportInterupted() {
        // The transport has suffered an interruption from which it hopes to recover.
    }

    public void transportResumed() {
        // The transport has resumed after an interruption.
    }
});
```

1.3 ActiveMQ in Action(3)

发表时间: 2008-02-26 关键字: activemq

2．3 Persistence

2．3．1 AMQ Message Store

AMQ Message Store是ActiveMQ5.0缺省的持久化存储。Message commands被保存到transactional journal（由rolling data logs组成）。Messages被保存到data logs中，同时被reference store进行索引以提高存取速度。Date logs由一些单独的data log文件组成，缺省的文件大小是32M，如果某个消息的大小超过了data log文件的大小，那么可以修改配置以增加data log文件的大小。如果某个data log文件中所有的消息都被成功消费了，那么这个data log文件将会被标记，以便在下一轮的清理中被删除或者归档。以下是其配置的一个例子：

```
<broker brokerName="broker" persistent="true" useShutdownHook="false">
  <persistenceAdapter>
     <amqPersistenceAdapter directory="${activemq.base}/data" maxFileLength="32mb"/>
  </persistenceAdapter>
</broker>
```

Property name	Default value	Comments
directory	activemq-data	the path to the directory to use to store the message store data and log files
useNIO	true	use NIO to write messages to the data logs
syncOnWrite	false	sync every write to disk
maxFileLength	32mb	a hint to set the maximum size of the message data logs
persistentIndex	true	use a persistent index for the message logs. If this is false, an in-memory structure is maintained
maxCheckpointMessageAddSize	4kb	the maximum number of messages to keep in a transaction before automatically committing
cleanupInterval	30000	time (ms) before checking for a discarding/moving message data logs that are no longer used
indexBinSize	1024	default number of bins used by the index. The bigger the bin size - the better the relative performance of the index
indexKeySize	96	the size of the index key - the key is the message id
indexPageSize	16kb	the size of the index page - the bigger the page - the better the write performance of the index

directoryArchive	archive	the path to the directory to use to store discarded data logs
archiveDataLogs	false	if true data logs are moved to the archive directory instead of being deleted

2．3．2 Kaha Persistence

Kaha Persistence 是一个专门针对消息持久化的解决方案。它对典型的消息使用模式进行了优化。在Kaha中，数据被追加到data logs中。当不再需要log文件中的数据的时候，log文件会被丢弃。以下是其配置的一个例子：

```
<broker brokerName="broker" persistent="true" useShutdownHook="false">
      <persistenceAdapter>
        <kahaPersistenceAdapter directory="activemq-data" maxDataFileLength="33554432"/>
      </persistenceAdapter>
</broker>
```

2．3．3 JDBC Persistence

目前支持的数据库有Apache Derby, Axion, DB2, HSQL, Informix, MaxDB, MySQL, Oracle, Postgresql, SQLServer, Sybase。

如果你使用的数据库不被支持，那么可以调整StatementProvider 来保证使用正确的SQL方言（flavour of SQL）。通常绝大多数数据库支持以下adaptor：

- org.activemq.store.jdbc.adapter.BlobJDBCAdapter
- org.activemq.store.jdbc.adapter.BytesJDBCAdapter
- org.activemq.store.jdbc.adapter.DefaultJDBCAdapter
- org.activemq.store.jdbc.adapter.ImageJDBCAdapter

也可以在配置文件中直接指定JDBC adaptor，例如：

```
<jdbcPersistenceAdapter  adapterClass="org.apache.activemq.store.jdbc.adapter.ImageBasedJDBCAda
```

以下是其配置的一个例子：

```
<persistence>
      <jdbcPersistence dataSourceRef=" mysql-ds"/>
</persistence>

<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>
```

需要注意的是，如果使用MySQL，那么需要设置relaxAutoCommit 标志为true。

2．3．4 Disable Persistence

以下是其配置的一个例子：

```
<broker persistent="false">
</broker>
```

1.4 ActiveMQ in Action(4)

发表时间: 2008-02-26 关键字: activemq

2．4 Security

ActiveMQ支持可插拔的安全机制，用以在不同的provider之间切换。

2．4．1 Simple Authentication Plugin

Simple Authentication Plugin适用于简单的认证需求，或者用于建立测试环境。它允许在XML配置文件中指定用户、用户组和密码等信息。以下是ActiveMQ配置的一个例子：

```
<plugins>
  ...
  <simpleAuthenticationPlugin>
    <users>
      <authenticationUser username="system" password="manager" groups="users,admins"/>
          <authenticationUser username="user" password="password" groups="users"/>
      <authenticationUser username="guest" password="password" groups="guests"/>
        </users>
  </simpleAuthenticationPlugin>
</plugins>
```

2．4．2 JAAS Authentication Plugin

JAAS Authentication Plugin依赖标准的JAAS机制来实现认证。通常情况下，你需要通过设置java.security.auth.login.config系统属性来配置login modules的配置文件。如果没有指定这个系统属性，那么JAAS Authentication Plugin会缺省使用login.config作为文件名。以下是一个login.config文件的例子：

activemq-domain {

org.apache.activemq.jaas.PropertiesLoginModule required debug=true
org.apache.activemq.jaas.properties.user="users.properties"
org.apache.activemq.jaas.properties.group="groups.properties";
};

这个login.config文件中设置了两个属性：org.apache.activemq.jaas.properties.user和org.apache.activemq.jaas.properties.group分别用来指向user.properties和group.properties文件。需要注意的是，PropertiesLoginModule使用本地文件的查找方式，而且查找时采用的base directory是login.config文件所在的目录。因此这个login.config说明user.properties和group.properties文件存放在跟login.config文件相同的目录里。

以下是ActiveMQ配置的一个例子：

```
<plugins>
  ...
  <jaasAuthenticationPlugin configuration="activemq-domain" />
</plugins>
```

基于以上的配置，在JAAS的LoginContext中会使用activemq-domain中配置的PropertiesLoginModule来进行登陆。

ActiveMQ JAAS还支持LDAPLoginModule、CertificateLoginModule、TextFileCertificateLoginModule等login module。

2．4．3 Custom Authentication Implementation

可以通过编码的方式为ActiveMQ增加认证功能。例如编写一个类继承自XBeanBrokerService。

```
package com.yourpackage;

import java.net.URI;
import java.util.HashMap;
import java.util.Map;

import org.apache.activemq.broker.Broker;
import org.apache.activemq.broker.BrokerFactory;
import org.apache.activemq.broker.BrokerService;
import org.apache.activemq.security.SimpleAuthenticationBroker;
import org.apache.activemq.xbean.XBeanBrokerService;

public class SimpleAuthBroker extends XBeanBrokerService {
    //
    private String user;
    private String password;

    @SuppressWarnings("unchecked")
    protected Broker addInterceptors(Broker broker) throws Exception {
        broker = super.addInterceptors(broker);
        Map passwords = new HashMap();
        passwords.put(getUser(), getPassword());
        broker = new SimpleAuthenticationBroker(broker, passwords, new HashMap());
```

```
        return broker;
    }

    public String getUser() {
            return user;
    }

    public void setUser(String user) {
            this.user = user;
    }

    public String getPassword() {
            return password;
    }

    public void setPassword(String password) {
            this.password = password;
    }
}
```

以下是ActiveMQ配置文件的一个例子：

```
<beans>
  …
  <auth:SimpleAuthBroker
        xmlns:auth="java://com.yourpackage"
    xmlns="http://activemq.org/config/1.0" brokerName="SimpleAuthBroker1" user="user" password=

    <transportConnectors>
      <transportConnector uri="tcp://localhost:61616"/>
    </transportConnectors>
  </auth:SimpleAuthBroker>
  …
</beans>
```

在这个配置文件中增加了一个namespace auth，用于指向之前编写的哪个类。同时为SimpleAuthBroker注入了两个属性值user和password，因此在被SimpleAuthBroker改写的addInterceptors方法里，可以使用这

两个属性进行认证了。ActiveMQ提供的SimpleAuthenticationBroker类继承自BrokerFilter（可以简单的看成是Broker的Adaptor），它的构造函数中的两个Map分别是userPasswords和userGroups。SimpleAuthenticationBroker在 addConnection方法中使用userPasswords进行认证，同时会把userGroups的信息保存到ConnectionContext中 。

2．4．4 Authorization Plugin

可以通过Authorization Plugin为认证后的用户授权，以下ActiveMQ配置文件的一个例子：

```
<plugins>
  <jaasAuthenticationPlugin configuration="activemq-domain"/>

  <authorizationPlugin>
    <map>
      <authorizationMap>
        <authorizationEntries>
          <authorizationEntry queue=">" read="admins" write="admins" admin="admins" />
          <authorizationEntry queue="USERS.>" read="users" write="users" admin="users" />
          <authorizationEntry queue="GUEST.>" read="guests" write="guests,users" admin="guests,

          <authorizationEntry topic=">" read="admins" write="admins" admin="admins" />
          <authorizationEntry topic="USERS.>" read="users" write="users" admin="users" />
          <authorizationEntry topic="GUEST.>" read="guests" write="guests,users" admin="guests,

          <authorizationEntry topic="ActiveMQ.Advisory.>" read="guests,users" write="guests,use
        </authorizationEntries>
      </authorizationMap>
    </map>
  </authorizationPlugin>
</plugins>
```

1.5 ActiveMQ in Action(5)

发表时间: 2008-02-26 关键字: activemq

2．5 Clustering

ActiveMQ从多种不同的方面提供了集群的支持。

2．5．1 Queue consumer clusters

ActiveMQ支持订阅同一个queue的consumers上的集群。如果一个consumer失效，那么所有未被确认（unacknowledged）的消息都会被发送到这个queue上其它的consumers。如果某个consumer的处理速度比其它consumers更快，那么这个consumer就会消费更多的消息。

需要注意的是，笔者发现AcitveMQ5.0版本的Queue consumer clusters存在一个bug：采用AMQ Message Store，运行一个producer，两个consumer，并采用如下的配置文件：

```
<beans>
  <broker xmlns="http://activemq.org/config/1.0" brokerName="BugBroker1" useJmx="true">

    <transportConnectors>
      <transportConnector uri="tcp://localhost:61616"/>
    </transportConnectors>

    <persistenceAdapter>
      <amqPersistenceAdapter directory="activemq-data/BugBroker1" maxFileLength="32mb"/>
    </persistenceAdapter>

  </broker>
</beans>
```

那么经过一段时间后可能会报出如下错误：

ERROR [ActiveMQ Transport: tcp:///127.0.0.1:1843 - RecoveryListenerAdapter.java:58 - RecoveryListenerAdapter] Message id ID:versus-1837-1203915536609-0:2:1:1:419 could not be recovered from the data store!

Apache官方文档说，此bug已经被修正，预定在5.1.0版本上体现。

2．5．2 Broker clusters

一个常见的场景是有多个JMS broker，有一个客户连接到其中一个broker。如果这个broker失效，那么客户会自动重新连接到其它的broker。在ActiveMQ中使用failover:// 协议来实现这个功能。ActiveMQ3.x版本的reliable://协议已经变更为failover://。

如果某个网络上有多个brokers而且客户使用静态发现（使用Static Transport或Failover Transport）或动态发现（使用Discovery Transport），那么客户可以容易地在某个broker失效的情况下切换到其它的brokers。然而，stand alone brokers并不了解其它brokers上的consumers，也就是说如果某个broker上没有consumers，那么这个broker上的消息可能会因得不到处理而积压起来。目前的解决方案是使用Network of brokers，以便在broker之间存储转发消息。ActiveMQ在未来会有更好的特性，用来在客户端处理这个问题。

从ActiveMQ1.1版本起，ActiveMQ支持networks of brokers。它支持分布式的queues和topics。一个broker会相同对待所有的订阅（subscription）：不管他们是来自本地的客户连接，还是来自远程broker，它都会递送有关的消息拷贝到每个订阅。远程broker得到这个消息拷贝后，会依次把它递送到其内部的本地连接上。有两种方式配置Network of brokers，一种是使用static transport，如下：

```
<broker brokerName="receiver" persistent="false" useJmx="false">
  <transportConnectors>
    <transportConnector uri="tcp://localhost:62002"/>
  </transportConnectors>
  <networkConnectors>
    <networkConnector uri="static:( tcp://localhost:61616,tcp://remotehost:61616)"/>
  </networkConnectors>
  …
</broker>
```

另外一种是使用multicast discovery，如下：

```
<broker name="sender" persistent="false" useJmx="false">
  <transportConnectors>
    <transportConnector uri="tcp://localhost:0" discoveryUri="multicast://default"/>
  </transportConnectors>
  <networkConnectors>
    <networkConnector uri="multicast://default"/>
  </networkConnectors>
  ...
</broker>
```

Network Connector有以下属性：

Property	Default Value	Description
name	bridge	name of the network - for more than one network connector between the same two brokers - use different names

dynamicOnly	false	if true, only forward messages if a consumer is active on the connected broker
decreaseNetworkConsumerPriority	false	decrease the priority for dispatching to a Queue consumer the further away it is (in network hops) from the producer
networkTTL	1	the number of brokers in the network that messages and subscriptions can pass through
conduitSubscriptions	true	multiple consumers subscribing to the same destination are treated as one consumer by the network
excludedDestinations	empty	destinations matching this list won't be forwarded across the network
dynamicallyIncludedDestinations	empty	destinations that match this list will be forwarded across the network n.b. an empty list means all destinations not in the excluded list will be forwarded
staticallyIncludedDestinations	empty	destinations that match will always be passed across the network - even if no consumers have ever registered an interest
duplex	false	if true, a network connection will be used to both produce AND Consume messages. This is useful for hub and spoke scenarios when the hub is behind a firewall etc.

关于conduitSubscriptions属性，这里稍稍说明一下。设想有两个brokers，分别是brokerA和brokerB，它们之间用forwarding bridge连接。有一个consumer连接到brokerA并订阅queue：Q.TEST。有两个consumers连接到brokerB，也是订阅queue：Q.TEST。这三个consumers有相同的优先级。然后启动一个producer，它发送了30条消息到brokerA。如果conduitSubscriptions=true，那么brokerA上的consumer会得到15条消息， 另外15条消息会发送给brokerB。此时负载并不均衡，因为此时brokerA将brokerB上的两个consumers视为一个；如果conduitSubscriptions=false，那么每个consumer上都会收到10条消息。以下是关于NetworkConnector属性的一个例子：

```
<networkConnectors>
  <networkConnector uri="static://(tcp://localhost:61617)"
     name="bridge" dynamicOnly="false" conduitSubscriptions="true"
     decreaseNetworkConsumerPriority="false">
     <excludedDestinations>
       <queue physicalName="exclude.test.foo"/>
       <topic physicalName="exclude.test.bar"/>
```

```
    </excludedDestinations>
    <dynamicallyIncludedDestinations>
      <queue physicalName="include.test.foo"/>
      <topic physicalName="include.test.bar"/>
    </dynamicallyIncludedDestinations>
    <staticallyIncludedDestinations>
      <queue physicalName="always.include.queue"/>
      <topic physicalName="always.include.topic"/>
    </staticallyIncludedDestinations>
  </networkConnector>
</networkConnectors>
```

2．5．3 Master Slave

在一个网络内运行多个brokers或者stand alone brokers时存在一个问题，这就是消息在物理上只被一个broker持有，因此当某个broker失效，那么你只能等待直到它重启后，这个broker上的消息才能够被继续发送（如果没有设置持久化，那么在这种情况下，消息将会丢失）。Master Slave 背后的想法是，消息被复制到slave broker，因此即使master broker遇到了像硬件故障之类的错误，你也可以立即切换到slave broker而不丢失任何消息。

Master Slave是目前ActiveMQ推荐的高可靠性和容错的解决方案。以下是几种不同的类型：

Master Slave Type	Requirements	Pros	Cons
Pure Master Slave	None	No central point of failure	Requires manual restart to bring back a failed master and can only support 1 slave
Shared File System Master Slave	A Shared File system such as a SAN	Run as many slaves as required. Automatic recovery of old masters	Requires shared file system
JDBC Master Slave	A Shared database	Run as many slaves as required. Automatic recovery of old masters	Requires a shared database. Also relatively slow as it cannot use the high performance journal

2．5．3．1 Pure Master Slave

Pure Master Slave的工作方式如下：

- Slave broker消费master broker上所有的消息状态，例如消息、确认和事务状态等。只要slave broker连接到了master broker，它不会（也不被允许）启动任何network connectors或者transport connectors，所以唯一的目的就是复制master broker的状态。
- Master broker只有在消息成功被复制到slave broker之后才会响应客户。例如，客户的commit请求只有在master broker和slave broker都处理完毕commit请求之后才会结束。
- 当master broker失效的时候，slave broker有两种选择，一种是slave broker启动所有的network connectors和transport connectors，这允许客户端切换到slave broker；另外一种是slave broker停止。这种情况下，slave broker只是复制了master broker的状态。
- 客户应该使用failover transport并且应该首先尝试连接master broker。例如：failover://(tcp://masterhost:61616,tcp://slavehost:61616)?randomize=false 设置randomize为false就可以让客户总是首先尝试连接master broker（slave broker并不会接受任何连接，直到它成为了master broker）。

Pure Master Slave具有以下限制：

- 只能有一个slave broker连接到master broker。
- 在因master broker失效而导致slave broker成为master之后，之前的master broker只有在当前的master broker（原slave broker）停止后才能重新生效。
- Master broker失效后而切换到slave broker后，最安全的恢复master broker的方式是人工处理。首先要停止slave broker（这意味着所有的客户也要停止）。然后把slave broker的数据目录中所有的数据拷贝到master broker的数据目录中。然后重启master broker和slave broker。

Master broker不需要特殊的配置。Slave broker需要进行以下配置

```
<broker masterConnectorURI="tcp://masterhost:62001" shutdownOnMasterFailure="false">
    ...
    <transportConnectors>
          <transportConnector uri="tcp://slavehost:61616"/>
   </transportConnectors>
</broker>
```

其中的masterConnectorURI用于指向master broker，shutdownOnMasterFailure用于指定slave broker在master broker失效的时候是否需要停止。此外，也可以使用如下配置：

```
<broker brokerName="slave" useJmx="false"  deleteAllMessagesOnStartup="true"  xmlns="http://act
  ...
  <services>
    <masterConnector remoteURI= "tcp://localhost:62001" userName="user" password="password"/>
```

```
  </services>
</broker>
```

需要注意的是，笔者认为ActiveMQ5.0版本的Pure Master Slave仍然不够稳定。

2．5．3．2 Shared File System Master Slave

如果你使用SAN或者共享文件系统，那么你可以使用Shared File System Master Slave。基本上，你可以运行多个broker，这些broker共享数据目录。当第一个broker得到文件上的排他锁之后，其它的broker便会在循环中等待获得这把锁。客户端使用failover transport来连接到可用的broker。当master broker失效的时候会释放这把锁，这时候其中一个slave broker会得到这把锁从而成为master broker。以下是ActiveMQ配置的一个例子：

```
<broker useJmx="false"  xmlns="http://activemq.org/config/1.0">
   <persistenceAdapter>
      <journaledJDBC dataDirectory="/sharedFileSystem/broker"/>
   </persistenceAdapter>
   …
</broker>
```

2．5．3．3 JDBC Master Slave

JDBC Master Slave的工作原理跟Shared File System Master Slave类似，只是采用了数据库作为持久化存储。以下是ActiveMQ配置的一个例子：

```
<beans>
  <broker xmlns="http://activemq.org/config/1.0" brokerName="JdbcMasterBroker">
    ...
    <persistenceAdapter>
      <jdbcPersistenceAdapter dataSource="#mysql-ds"/>
    </persistenceAdapter>

  </broker>

  <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:3306/test?relaxAutoCommit=true"/>
    <property name="username" value="username"/>
    <property name="password" value="passward"/>
    <property name="poolPreparedStatements" value="true"/>
  </bean>
</beans>
```

需要注意的是，如果你使用MySQL数据库，需要首先执行以下三条语句：（Apache官方文档说，此bug已经被修正，预定在5.1.0版本上体现）

```
ALTER TABLE activemq_acks ENGINE = InnoDB;
ALTER TABLE activemq_lock ENGINE = InnoDB;
ALTER TABLE activemq_msgs ENGINE = InnoDB;
```

1.6 ActiveMQ in Action(6)

发表时间: 2008-02-26 关键字: activemq

2．6 Features

ActiveMQ包含了很多功能强大的特性，下面简要介绍其中的几个。

2．6．1 Exclusive Consumer

Queue中的消息是按照顺序被分发到consumers的。然而，当你有多个consumers同时从相同的queue中提取消息时，你将失去这个保证。因为这些消息是被多个线程并发的处理。有的时候，保证消息按照顺序处理是很重要的。例如，你可能不希望在插入订单操作结束之前执行更新这个订单的操作。

ActiveMQ从4.x版本起开始支持Exclusive Consumer （或者说Exclusive Queues）。 Broker会从多个consumers中挑选一个consumer来处理queue中所有的消息，从而保证了消息的有序处理。如果这个consumer失效，那么broker会自动切换到其它的consumer。

可以通过Destination Options 来创建一个Exclusive Consumer，如下：

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

顺便说一下，可以给consumer设置优先级，以便针对网络情况（如network hops）进行优化，如下：

```
queue = new ActiveMQQueue("TEST.QUEUE?consumer.exclusive=true &consumer.priority=10");
```

2．6．2 Message Groups

用Apache官方文档的话说，Message Groups rock！它是Exclusive Consumer功能的增强。逻辑上，Message Groups 可以看成是一种并发的Exclusive Consumer。跟所有的消息都由唯一的consumer处理不同，JMS 消息属性JMSXGroupID 被用来区分message group。Message Groups特性保证所有具有相同JMSXGroupID 的消息会被分发到相同的consumer（只要这个consumer保持active）。另外一方面，Message Groups特性也是一种负载均衡的机制。

在一个消息被分发到consumer之前，broker首先检查消息JMSXGroupID属性。如果存在，那么broker 会检查是否有某个consumer拥有这个message group。如果没有，那么broker会选择一个consumer，并将它关联到这个message group。此后，这个consumer会接收这个message group的所有消息，直到：

- Consumer被关闭。
- Message group被关闭。通过发送一个消息，并设置这个消息的JMSXGroupSeq为0。

从4.1版本开始，ActiveMQ支持一个布尔字段JMSXGroupFirstForConsumer 。当某个message group的第一个消息被发送到consumer的时候，这个字段被设置。如果客户使用failover transport连接到broker。在由

于网络问题等造成客户重新连接到broker的时候，相同message group的消息可能会被分发到不同与之前的consumer，因此JMSXGroupFirstForConsumer字段也会被重新设置。

以下是使用message groups的例子：

```
Mesasge message = session.createTextMessage("<foo>hey</foo>");
message.setStringProperty("JMSXGroupID", "IBM_NASDAQ_20/4/05");
...
producer.send(message);
```

2．6．3 JMS Selectors

JMS Selectors用于在订阅中，基于消息属性对进行消息的过滤。JMS Selectors由SQL92语法定义。以下是个Selectors的例子：

```
consumer = session.createConsumer(destination, "JMSType = 'car' AND weight > 2500");
```

在JMS Selectors表达式中，可以使用IN、NOT IN、LIKE等，例如：

LIKE '12%3'（'123' true，'12993' true，'1234' false）

LIKE 'l_se'（'lose' true，'loose' false）

LIKE '_%' ESCAPE '\'（'_foo' true，'foo' false）

需要注意的是，JMS Selectors表达式中的日期和时间需要使用标准的long型毫秒值。另外表达式中的属性不会自动进行类型转换，例如：

```
myMessage.setStringProperty("NumberOfOrders", "2");
```

"NumberOfOrders > 1" 求值结果是false。关于JMS Selectors的详细文档请参考javax.jms.Message的javadoc。

上一小节介绍的Message Groups虽然可以保证具有相同message group的消息被唯一的consumer顺序处理，但是却不能确定被哪个consumer处理。在某些情况下，Message Groups可以和JMS Selector一起工作，例如：

设想有三个consumers分别是A、B和C。你可以在producer中为消息设置三个message groups分别是"A"、"B"和"C"。然后令consumer A使用"JMXGroupID = 'A'"作为selector。B和C也同理。这样就可以保证message group A的消息只被consumer A处理。需要注意的是，这种做法有以下缺点：

- producer必须知道当前正在运行的consumers，也就是说producer和consumer被耦合到一起。
- 如果某个consumer失效，那么应该被这个consumer消费的消息将会一直被积压在broker上。

2．6．4 Pending Message Limit Strategy

首先简要介绍一下prefetch机制。ActiveMQ通过prefetch机制来提高性能，这意味这客户端的内存里可能会

缓存一定数量的消息。缓存消息的数量由prefetch limit来控制。当某个consumer的prefetch buffer已经达到上限，那么broker不会再向consumer分发消息，直到consumer向broker发送消息的确认。可以通过在ActiveMQConnectionFactory或者ActiveMQConnection上设置ActiveMQPrefetchPolicy对象来配置prefetch policy。也可以通过connection options或者destination options来配置。例如：

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

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

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

prefetch size的缺省值如下：

- persistent queues (default value: 1000)
- non-persistent queues (default value: 1000)
- persistent topics (default value: 100)
- non-persistent topics (default value: Short.MAX_VALUE -1)

慢消费者会在非持久的topics上导致问题：一旦消息积压起来，会导致broker把大量消息保存在内存中，broker也会因此而变慢。未来ActiveMQ可能会实现磁盘缓存，但是这也还是会存在性能问题。目前ActiveMQ使用Pending Message Limit Strategy来解决这个问题。除了prefetch buffer之外，你还要配置缓存消息的上限，超过这个上限后，新消息到来时会丢弃旧消息。通过在配置文件的destination map中配置PendingMessageLimitStrategy，可以为不用的topic namespace配置不同的策略。目前有以下两种：

- ConstantPendingMessageLimitStrategy。这个策略使用常量限制。
  例如：<constantPendingMessageLimitStrategy limit="50"/>
- PrefetchRatePendingMessageLimitStrategy。这个策略使用prefetch size的倍数限制。
  例如：<prefetchRatePendingMessageLimitStrategy multiplier="2.5"/>

在以上两种方式中，如果设置0意味着除了prefetch之外不再缓存消息；如果设置-1意味着禁止丢弃消息。

此外，你还可以配置消息的丢弃策略，目前有以下两种：

- oldestMessageEvictionStrategy。这个策略丢弃最旧的消息。
- oldestMessageWithLowestPriorityEvictionStrategy。这个策略丢弃最旧的，而且具有最低优先级的消息。

以下是个ActiveMQ配置文件的例子：

```
<broker persistent="false" brokerName="${brokername}" xmlns="http://activemq.org/config/1.0">
    <destinationPolicy>
      <policyMap>
        <policyEntries>
          <policyEntry topic="PRICES.>">
```

```
            <!--  10 seconds worth -->
            <subscriptionRecoveryPolicy>
              <timedSubscriptionRecoveryPolicy recoverDuration="10000" />
            </subscriptionRecoveryPolicy>

            <!-- lets force old messages to be discarded for slow consumers -->
            <pendingMessageLimitStrategy>
              <constantPendingMessageLimitStrategy limit="10"/>
            </pendingMessageLimitStrategy>
          </policyEntry>
        </policyEntries>
      </policyMap>
    </destinationPolicy>
    ...
</broker>
```

2．6．5 Composite Destinations

从1.1版本起, ActiveMQ支持composite destinations。它允许用一个虚拟的destination 代表多个destinations。例如你可以通过composite destinations在一个操作中同时向12个queue发送消息。在composite destinations中，多个destination之间采用","分割。例如：

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

如果你希望使用不同类型的destination，那么需要加上前缀如queue:// 或topic://，例如：

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

以下是ActiveMQ配置文件进行配置的一个例子：

```
<destinationInterceptors>
  <virtualDestinationInterceptor>
    <virtualDestinations>
      <compositeQueue name="MY.QUEUE">
        <forwardTo>
          <queue physicalName="FOO" />
          <topic physicalName="BAR" />
```

```
        </forwardTo>
      </compositeQueue>
    </virtualDestinations>
  </virtualDestinationInterceptor>
</destinationInterceptors>
```

可以在转发前，先通过JMS Selector判断一个消息是否需要转发，例如：

```
<destinationInterceptors>
  <virtualDestinationInterceptor>
    <virtualDestinations>
      <compositeQueue name="MY.QUEUE">
        <forwardTo>
          <filteredDestination selector="odd = 'yes'" queue="FOO"/>
          <filteredDestination selector="i = 5" topic="BAR"/>
        </forwardTo>
      </compositeQueue>
    </virtualDestinations>
  </virtualDestinationInterceptor>
</destinationInterceptors>
```

2．6．6 Mirrored Queues

每个queue中的消息只能被一个consumer消费。然而，有时候你可能希望能够监视生产者和消费者之间的消息流。你可以通过使用Virtual Destinations 来建立一个virtual queue 来把消息转发到多个queues中。但是为系统中每个queue都进行如此的配置可能会很麻烦。

ActiveMQ支持Mirrored Queues。Broker会把发送到某个queue的所有消息转发到一个名称类似的topic，因此监控程序可以订阅这个mirrored queue topic。为了启用Mirrored Queues，首先要将BrokerService的useMirroredQueues属性设置成true，然后可以通过destinationInterceptors设置其它属性，如mirror topic的前缀，缺省是"VirtualTopic.Mirror."。以下是ActiveMQ配置文件的一个例子：

```
<broker xmlns="http://activemq.org/config/1.0" brokerName="MirroredQueuesBroker1" useMirroredQu

  <transportConnectors>
    <transportConnector uri="tcp://localhost:61616"/>
  </transportConnectors>
```

```
  <destinationInterceptors>
      <mirroredQueue copyMessage = "true" prefix="Mirror.Topic"/>
  </destinationInterceptors>
  ...
</broker>
```

假如某个producer向名为Foo.Bar的queue中发送消息，那么你可以通过订阅名为Mirror.Topic.Foo.Bar的topic来获得发送到Foo.Bar中的所有消息。

1.7 ActiveMQ in Action(7)

发表时间: 2008-02-27 关键字: activemq

2．6．7 Wildcards

Wildcards用来支持联合的名字分层体系（federated name hierarchies）。它不是JMS规范的一部分，而是ActiveMQ的扩展。ActiveMQ支持以下三种wildcards：

- "." 用于作为路径上名字间的分隔符。
- "*" 用于匹配路径上的任何名字。
- ">" 用于递归地匹配任何以这个名字开始的destination。

作为一种组织事件和订阅感兴趣那部分信息的一种方法，这个概念在金融市场领域已经流行了一段时间了。设想你有以下两个destination：

- PRICE.STOCK.NASDAQ.IBM （IBM在NASDAQ的股价）
- PRICE.STOCK.NYSE.SUNW （SUN在纽约证券交易所的股价）

订阅者可以明确地指定destination的名字来订阅消息，或者它也可以使用wildcards来定义一个分层的模式来匹配它希望订阅的destination。例如：

Subscription	Meaning
PRICE.>	Any price for any product on any exchange
PRICE.STOCK.>	Any price for a stock on any exchange
PRICE.STOCK.NASDAQ.*	Any stock price on NASDAQ
PRICE.STOCK.*.IBM	Any IBM stock price on any exchange

2．6．8 Async Sends

ActiveMQ支持以同步（sync）方式或者异步（async）方式向broker发送消息。 使用何种方式对send方法的延迟有巨大的影响。对于生产者来说，既然延迟是决定吞吐量的重要因素，那么使用异步发送方式会极大地提高系统的性能。

ActiveMQ缺省使用异步传输方式。但是按照JMS规范，当在事务外发送持久化消息的时候，ActiveMQ会强制使用同步发送方式。在这种情况下，每一次发送都是同步的，而且阻塞到收到broker的应答。这个应答保证了broker已经成功地将消息持久化，而且不会丢失。但是这样作也严重地影响了性能。

如果你的系统可以容忍少量的消息丢失，那么可以在事务外发送持久消息的时候，选择使用异步方式。以下是几种不同的配置方式：

```
cf = new ActiveMQConnectionFactory("tcp://locahost:61616?jms.useAsyncSend=true");
((ActiveMQConnectionFactory)connectionFactory).setUseAsyncSend(true);
((ActiveMQConnection)connection).setUseAsyncSend(true);
```

2．6．9 Dispatch Policies

2．6．9．1 Round Robin Dispatch Policy

在2．6．4小节介绍过ActiveMQ的prefetch机制，ActiveMQ的缺省参数是针对处理大量消息时的高性能和高吞吐量而设置的。所以缺省的prefetch参数比较大，而且缺省的dispatch policies会尝试尽可能快的填满prefetch缓冲。然而在有些情况下，例如只有少量的消息而且单个消息的处理时间比较长，那么在缺省的prefetch和dispatch policies下，这些少量的消息总是倾向于被分发到个别的consumer上。这样就会因为负载的不均衡分配而导致处理时间的增加。

Round robin dispatch policy会尝试平均分发消息，以下是ActiveMQ配置文件的一个例子：

```
<destinationPolicy>
  <policyMap>
    <policyEntries>
      <policyEntry topic="FOO.>">
        <dispatchPolicy>
          <roundRobinDispatchPolicy />
        </dispatchPolicy>
      </policyEntry>
    </policyEntries>
  </policyMap>
</destinationPolicy>
```

2．6．9．2 Strict Order Dispatch Policy

有时候需要保证不同的topic consumer以相同的顺序接收消息。通常ActiveMQ会保证topic consumer以相同的顺序接收来自同一个producer的消息。然而，由于多线程和异步处理，不同的topic consumer可能会以不同的顺序接收来自不同producer的消息。例如有两个producer，分别是P和Q。差不多是同一时间内，P发送了P1、P2和P3三个消息；Q发送了Q1和Q2两个消息。两个不同的consumer可能会以以下顺序接收到消息：

consumer1: P1 P2 Q1 P3 Q2

consumer2: P1 Q1 Q2 P2 P3

Strict order dispatch policy 会保证每个topic consumer会以相同的顺序接收消息，代价是性能上的损失。以下是采用了strict order dispatch policy后，两个不同的consumer可能以以下的顺序接收消息：

consumer1: P1 P2 Q1 P3 Q2
consumer2: P1 P2 Q1 P3 Q2

以下是ActiveMQ配置文件的一个例子：

```
<destinationPolicy>
  <policyMap>
    <policyEntries>
      <policyEntry topic=""FOO.>">
        <dispatchPolicy>
          <strictOrderDispatchPolicy />
        </dispatchPolicy>
      </policyEntry>
    </policyEntries>
  </policyMap>
</destinationPolicy>
```

2．6．10 Message Cursors

当producer发送的持久化消息到达broker之后，broker首先会把它保存在持久存储中。接下来，如果发现当前有活跃的consumer，而且这个consumer消费消息的速度能跟上producer生产消息的速度，那么ActiveMQ会直接把消息传递给broker内部跟这个consumer关联的dispatch queue；如果当前没有活跃的consumer或者consumer消费消息的速度跟不上producer生产消息的速度，那么ActiveMQ会使用Pending Message Cursors保存对消息的引用。在需要的时候，Pending Message Cursors把消息引用传递给broker内部跟这个consumer关联的dispatch queue。以下是两种Pending Message Cursors：

- VM Cursor。在内存中保存消息的引用。
- File Cursor。首先在内存中保存消息的引用，如果内存使用量达到上限，那么会把消息引用保存到临时文件中。

在缺省情况下，ActiveMQ 5.0根据使用的Message Store来决定使用何种类型的Message Cursors，但是你可以根据destination来配置Message Cursors。

对于topic，可以使用的pendingSubscriberPolicy 有vmCursor和fileCursor。可以使用的PendingDurableSubscriberMessageStoragePolicy有vmDurableCursor 和fileDurableSubscriberCursor。以下是ActiveMQ配置文件的一个例子：

```
<destinationPolicy>
  <policyMap>
    <policyEntries>
      <policyEntry topic="org.apache.>">
        <pendingSubscriberPolicy>
          <vmCursor />
        </pendingSubscriberPolicy>
        <PendingDurableSubscriberMessageStoragePolicy>
          <vmDurableCursor/>
        </PendingDurableSubscriberMessageStoragePolicy>
      </policyEntry>
    </policyEntries>
  </policyMap>
</destinationPolicy>
```

对于queue，可以使用的pendingQueuePolicy有vmQueueCursor 和 fileQueueCursor。以下是ActiveMQ配置文件的一个例子：

```
<destinationPolicy>
  <policyMap>
    <policyEntries>
      <policyEntry queue="org.apache.>">
        <pendingQueuePolicy>
          <vmQueueCursor />
        </pendingQueuePolicy>
      </policyEntry>
    </policyEntries>
  </policyMap>
</destinationPolicy>
```

2．6．11 Optimized Acknowledgement

ActiveMQ缺省支持批量确认消息。由于批量确认会提高性能，因此这是缺省的确认方式。如果希望在应用程序中禁止经过优化的确认方式，那么可以采用如下方法：

```
cf = new ActiveMQConnectionFactory ("tcp://locahost:61616?jms.optimizeAcknowledge=false");
((ActiveMQConnectionFactory)connectionFactory).setOptimizeAcknowledge(false);
((ActiveMQConnection)connection).setOptimizeAcknowledge(false);
```

2．6．12 Producer Flow Control

同步发送消息的producer会自动使用producer flow control ；对于异步发送消息的producer，要使用producer flow control，你先要为connection配置一个ProducerWindowSize参数，如下：

```
((ActiveMQConnectionFactory)cf).setProducerWindowSize(1024000);
```

ProducerWindowSize是producer在发送消息的过程中，收到broker对于之前发送消息的确认之前， 能够发送消息的最大字节数。你也可以禁用producer flow control，以下是ActiveMQ配置文件的一个例子：

```
<destinationPolicy>
  <policyMap>
    <policyEntries>
      <policyEntry topic="FOO.>" producerFlowControl="false">
        <dispatchPolicy>
          <strictOrderDispatchPolicy/>
        </dispatchPolicy>
      </policyEntry>
    </policyEntries>
  </policyMap>
</destinationPolicy>
```

2．6．13 Message Transformation

有时候需要在JMS provider内部进行message的转换。从4.2版本起，ActiveMQ 提供了一个MessageTransformer 接口用于进行消息转换，如下：

```
public interface MessageTransformer {
    Message producerTransform(Session session, MessageProducer producer, Message message) throw
    Message consumerTransform(Session session, MessageConsumer consumer, Message message)throws
}
```

通过在以下对象上通过调用setTransformer方法来设置MessageTransformer：

- ActiveMQConnectionFactory
- ActiveMQConnection
- ActiveMQSession
- ActiveMQMessageConsumer
- ActiveMQMessageProducer

MessageTransformer接口支持：

- 在消息被发送到JMS provider的消息总线前进行转换。通过producerTransform方法。
- 在消息到达消息总线后，但是在consumer接收到消息前进行转换。通过consumerTransform方法。

以下是个简单的例子：

```
public class SimpleMessage implements Serializable {
    //
    private static final long serialVersionUID = 2251041841871975105L;

    //
    private String id;
    private String text;

    public String getId() {
            return id;
    }
    public void setId(String id) {
            this.id = id;
    }
    public String getText() {
            return text;
    }
    public void setText(String text) {
            this.text = text;
    }
}
```

在producer内发送ObjectMessage，如下：

```
SimpleMessage sm = new SimpleMessage();
sm.setId("1");
sm.setText("this is a sample message");
ObjectMessage message = session.createObjectMessage();
message.setObject(sm);
producer.send(message);
```

在consumer的session上设置一个MessageTransformer用于将ObjectMessage转换成TextMessage，如下：

```
((ActiveMQSession)session).setTransformer(new MessageTransformer() {
public Message consumerTransform(Session session, MessageConsumer consumer, Message message) th
        ObjectMessage om = (ObjectMessage)message;
        XStream xstream = new XStream();
        xstream.alias("simple message", SimpleMessage.class);
        String xml = xstream.toXML(om.getObject());
        return session.createTextMessage(xml);
}

public Message producerTransform(Session session, MessageProducer consumer, Message message) th
        return null;
}
});
```