小飞侠 RP 1
2019-08-17 发布

消息中间件之ActiveMQ入门

1. JMS与消息中间件

1.1 jms介绍

jms是java消息服务接口规范，主要包含四大元素：**生产者、消费者、消息、消息服务。**

- 生产者：创建消息，并把消息发动到消息服务；
- 消费者：从消息服务接收消息；
- 消息服务：即MQ消息服务（broker），而生产者与消费者相对其均为客服端；
- 消息：整个消息服务的传输对象，消息包含消息头、消息属性、消息体；

> **常用消息头属性**：JMSDestination(消息目的地，如果生产者指定了目的地，在发送时会改为生产者绑定的目的地)、JMSDeliveryMode(是持久还是非持久)、JMSExpiration(过期时间，默认永久)、JMSPriority(优先级，0-9，数值越大优先级越高，默认为4)、JMSMessageId(唯一的消息ID)；
> **消息属性**：可视为消息头属性的扩展，通过setXxxProperty(k,v)设置；
> **消息体**：封装消息的具体数据，发送与接收的消息体类型必须一致，消息体类型总共有5种，TextMessage、Mapmessage、BytesMessage、StreamMessage、ObjectMessage；

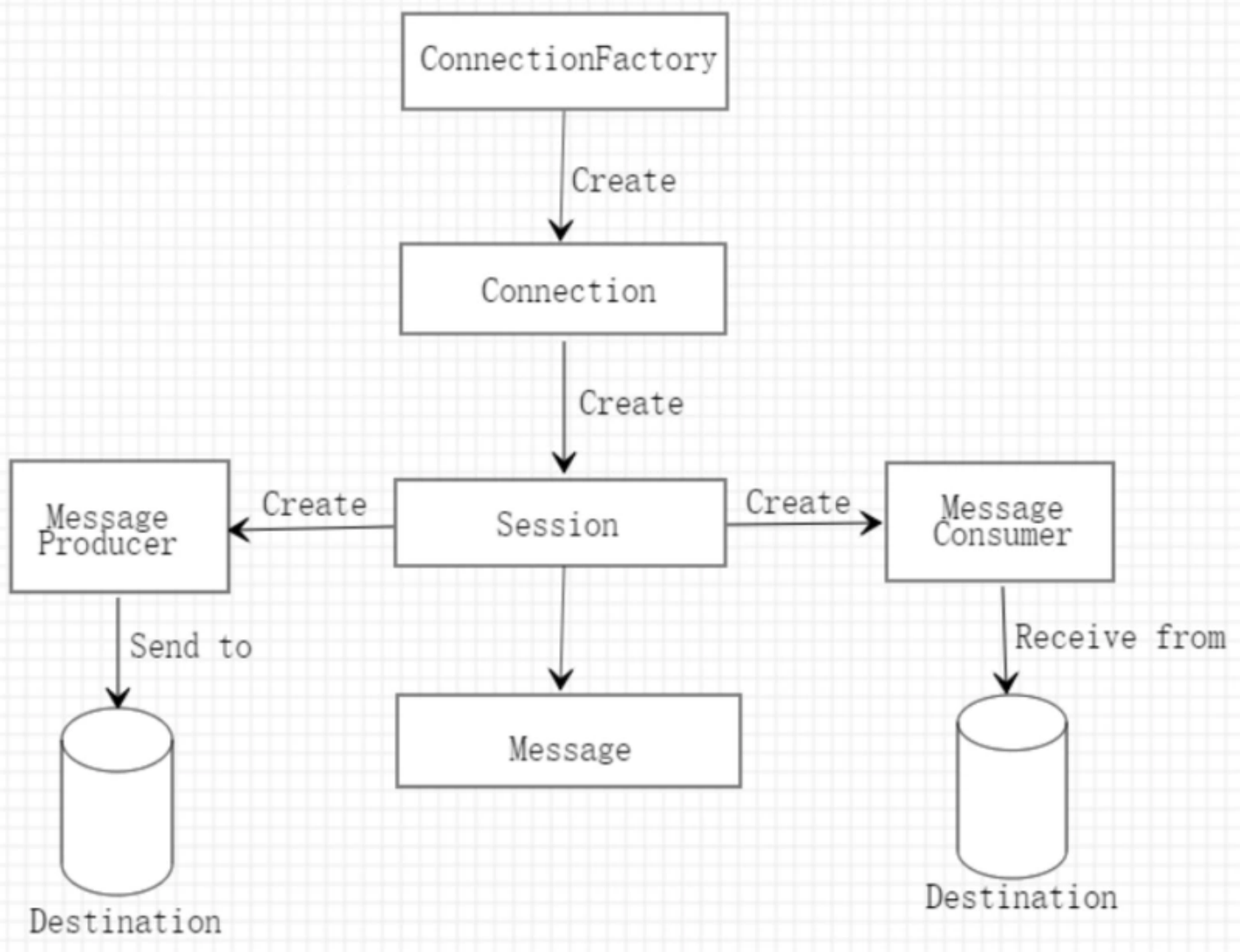
1.2 jms消息传递模式

jms消息传递模式有如下两种，
点对点消息传递模式（P2P）：消息发送到一个特殊队列(queue), 消费者从队列获取消息，一条消息只能被只能被一个消费者消费；
发布/订阅消息传递模式(publish-subscribe)：消息被发送到一个主题上(topic)，所有订阅了该主题的消费者，都能接收到消息。

1.3 jms编码总体架构

JMS应用程序由如下基本模块组成，

2. 连接对象，创建回话对象(session);
3. 会话对象，创建生产者对象(producer)、消费者对象(consumer)以及消息对象(message)；
4. 目的地(queue/topic)，点对点模式下目的地是队列(queue)，发布/订阅模式下目的地是主题(topic)，生产者把消息发送到目的地，消费者从目的地接收消息



1.4 消息中间件

消息中间件是实现了jms规范的落地产品，目前市场上主流的消息中间件有 ActiveMQ、Kafka、RocketMQ、RabbitMQ等。企业开发中使用消息中间件的主要目的是**解决耦合调用**、**抵御洪峰流量(削峰)**等。 以下主要讲解ActiveMQ的使用。

2. ActiveMQ安装并启动

具体安装步骤这里不再详述，可参考官网http://activemq.apache.org。安装成功后，进入安装目录，在bin目录下执行 `./activemq start` 命令，即可启动MQ服务，如果启动服务需要指定配置文件，命令为 `./activemq start xbean:file:./conf/myConfig.xml` 不指定默认为conf目录下的activemq.xml 修

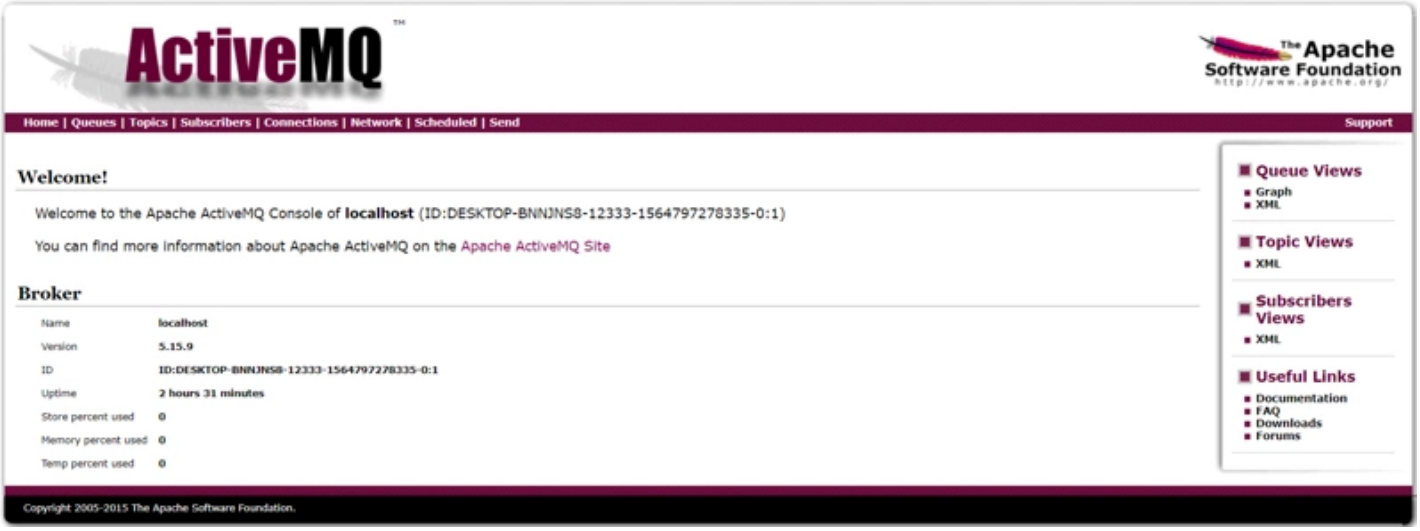
在conf目录下找到 `activemq.xml` 配置文件打开，里面包含如下内容，

```
<transportConnectors>
    <!-- DOS protection, limit concurrent connections to 1000 and frame size to 100MB -->
    <transportConnector name="openwire" uri="tcp://0.0.0.0:61616?maximumConnections=1000&amp;
    <transportConnector name="amqp" uri="amqp://0.0.0.0:5672?maximumConnections=1000&amp;wire
    <transportConnector name="stomp" uri="stomp://0.0.0.0:61613?maximumConnections=1000&amp;w
    <transportConnector name="mqtt" uri="mqtt://0.0.0.0:1883?maximumConnections=1000&amp;wire
    <transportConnector name="ws" uri="ws://0.0.0.0:61614?maximumConnections=1000&amp;wireFor
</transportConnectors>
```

这里配置的是MQ服务的各种传输协议连接和默认端口。再往下会发现这行内容 `<import resource="jetty.xml"/>` ，activemq.xml文件中导入了一个名为 `jetty.xml` 的配置文件，在conf目录下找到jetty.xml文件打开，里面配置了访问MQ服务web控制台的一些信息，

```
<bean id="jettyPort" class="org.apache.activemq.web.WebConsolePort" init-method="start">
    <!-- the default port number for the web console -->
    <property name="host" value="0.0.0.0"/>
    <property name="port" value="8161"/>
</bean>
```

其中8161为web控制台端口，MQ服务启动后，浏览器中访问http://localhost:8161/admin，输入用户名和密码，默认都为admin，即可看到如下页面，



3. 编码实战

ActiveMQ服务启动成功后，可以编写生产者客户端往MQ服务发送消息，消费者客户端从MQ服务获取消息。项目建好之后需要先引入ActiveMQ相关依赖，以gradle为例：

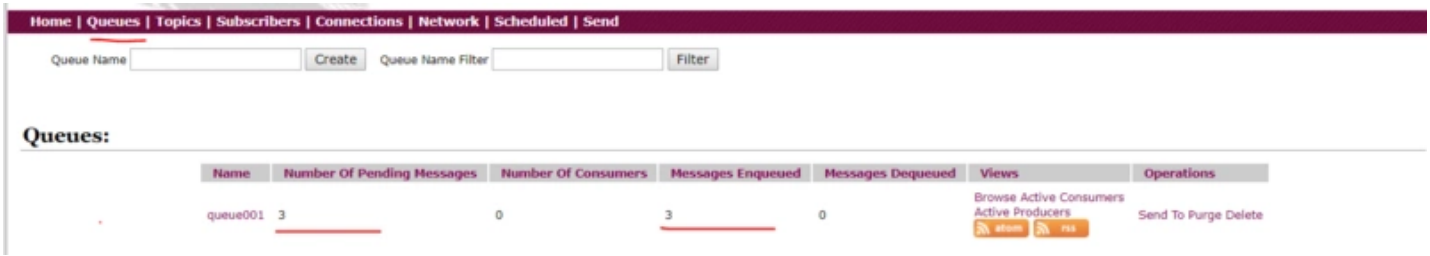
```
compile group: 'org.apache.activemq', name: 'activemq-all', version: '5.15.9'
```

3.1 点对点消息

3.1.1 生产者

```
package com.taicw.code.activemq.start.queue;

import org.apache.activemq.ActiveMQConnectionFactory;

import javax.jms.*;

/**
 * Created by taichangwei on 2019/6/22.
 */
public class QueueProducer {

    private static final String BROKER_URL = "tcp://localhost:61616";
    private static final String QUEUE_NAME = "queue001";

    public static void main(String[] args) throws JMSException, InterruptedException {

        //1、创建连接工厂。这里传入ActiveMQ消息服务连接地址，并使用默认用户名和密码。
        // 也可使用ActiveMQConnectionFactory()构造器或者ActiveMQConnectionFactory(String user
        ConnectionFactory connectionFactory = new ActiveMQConnectionFactory(BROKER_URL);
        //2、通过工厂对象创建连接
        Connection connection = connectionFactory.createConnection();
        //3、通过连接对象创建会话。第一个参数是否开启事务，第二参数指定签收类型
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
```

执行main()方法后，进入web控制台可以看到待消费消息有3条，入队消息有3条，说明消息已经成功发送至MQ服务器。



3.1.2 消费者

```
package com.taicw.code.activemq.start.queue;

import org.apache.activemq.ActiveMQConnectionFactory;
```

```
/**
 * Created by taichangwei on 2019/6/22.
 */
public class QueueConsumer {

    private static final String BROKER_URL = "tcp://localhost:61616";
    private static final String QUEUE_NAME = "queue001";

    public static void main(String[] args) throws JMSException {
        ConnectionFactory connectionFactory = new ActiveMQConnectionFactory(BROKER_URL);
        Connection connection = connectionFactory.createConnection();
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        Queue queue = session.createQueue(QUEUE_NAME);
        MessageConsumer consumer = session.createConsumer(queue);

        //在调用receive()方法之前必须要调用start()方法启动连接，否者receive()接收不到消息会被一直
```

消费者客户端编码过程与生产者基本一致，只不过一个是生产者发送调用 `send()` 方法，一个是消费者接收调用 `receive()` 方法。其中需要注意的是 `receive()` 方法是一个阻塞方法，接收不到消息会一直阻塞等待，并且调用 `receive()` 之前必须调用 `connection.start()` 启动连接，否者接收不到消息。
执行main()方法后，进入web控制台可以看到待消费消息变为0条，出队消息变为3条，并且有一个消费者，说明消息被消费成功。

3.1.3 消息监听器实现异步非阻塞消费消息

上面我们了解到 `MessageConsumer#receive()` 方法是个阻塞方法，实际开发中不可能一直去阻塞等待，可以为消费者对象设置消息监听器来实现异步非阻塞消费消息，修改消费者代码如下：

```
package com.taicw.code.activemq.start.queue;

import org.apache.activemq.ActiveMQConnectionFactory;

import javax.jms.*;

/**
 * Created by taichangwei on 2019/6/22.
 */
public class QueueConsumer {
```

```
    public static void main(String[] args) throws JMSException, InterruptedException {
        ConnectionFactory connectionFactory = new ActiveMQConnectionFactory(BROKER_URL);
        Connection connection = connectionFactory.createConnection();
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        Queue queue = session.createQueue(QUEUE_NAME);
        MessageConsumer consumer = session.createConsumer(queue);

        //在调用receive()方法之前必须要调用start()方法启动连接，否者receive()接收不到消息会被一直
//          connection.start();
```

`setMessageListener()` 方法需要传入一个 `MessageListener` 实例对象，并实现 `onMessage()` ，这里使用的是lambda表达式。

3.2 发布/订阅消息

发布订阅消息与上面点对点消息的生产者与消费者编码一致，唯一要改变的是把消息目的地由queue改为topic。

- 生产者

```
...省略...
Topic topic = session.createTopic(TOPIC_NAME);
MessageProducer producer = session.createProducer(topic);
...省略...
```

- 消费者

```
...省略...
Topic topic = session.createTopic(TOPIC_NAME);
MessageConsumer consumer = session.createConsumer(topic);
...省略...
```

> 1.在点对点消息示例中，当同时启动多个消费者时(即同时执行多次main()方法)，生产者发布的每条消息只能被其中一个消费者消费一次；
> 2.在发布/订阅消息示例中，消费者不能消费订阅主题之前的消息，当同时启动多个消费者时，生产者发布的每条消息可以同时被多个消费者消费；

4. 传输协议

4.1 ActiveMQ默认支持的传输协议

ActiveMQ出厂默认支持的传输协议有 `tcp` 、 `amqp` 、 `stomp` 、 `mqtt` 、 `ws` ，在 `activemq.xml` 配置文件可以找到这几种协议的配置，

```
<transportConnectors>
    <!-- DOS protection, limit concurrent connections to 1000 and frame size to 100MB -->
    <transportConnector name="openwire" uri="tcp://0.0.0.0:61616?maximumConnections=1000&amp;
    <transportConnector name="amqp" uri="amqp://0.0.0.0:5672?maximumConnections=1000&amp;wire
    <transportConnector name="stomp" uri="stomp://0.0.0.0:61613?maximumConnections=1000&amp;w
    <transportConnector name="mqtt" uri="mqtt://0.0.0.0:1883?maximumConnections=1000&amp;wire
    <transportConnector name="ws" uri="ws://0.0.0.0:61614?maximumConnections=1000&amp;wireFor
</transportConnectors>
```

对于java开发后四种协议不经常使用，这里主要说一次tcp协议。tcp协议的client监听端口默认是61616，在网络上传输数据，必须序列化数据，消息是通过一个write protocol来序列化为字节流。默认情况ActiveMQ会把wire protocol叫做Open Wire，它的目的是促使网络上的效率和数据快速交互。
tcp传输的优点：

1. 传输可靠性高、稳定性强
2. 高效性：字节流方式传递，效率高
3. 有效性、可用性：应用广泛，支持任何平台

tcp连接的URL形式如：**tcp://hostname:port?key=value**。更多协议的可配置参数请参考 http://activemq.apache.org/tc...

4.2 使用NIO传输协议提供更好的性能

使用tcp协议，每一个连接都会创建一个线程，当client连接较多时需要大量的系统开销，nio支持多个连接使用同一个线程，相比tcp需要更少的线程数。

nio协议基于tcp协议之上进行了扩展和优化。要使ActiveMQ支持nio协议，只需要做少量的修改即可。打开 `activemq.xml` 配置文件，在 `<transportConnectors>` 节点内添加， `<transportConnector name="nio" uri="nio://0.0.0.0:61617"/>` ，同时客户端代码url连接形式要改为 `nio://hostname:port?key=value` ，后面的可选参数与tcp协议一致。

> tcp协议也好nio协议也好，都绑定了特定的端口，如何实现一个端口可以支持多种协议呢？
> ActiveMQ提供了一个auto协议，类似于一个适配器协议，在不改变端口的情况下可以切换协议。详细配置参考官方文档 http://activemq.apache.org/auto

其他协议配置参见官网文档 http://activemq.apache.org/co...

5 ActiveMQ消息高可用

5.1 消息持久化

5.1.1 持久化编码

如果生产者把消息发送到了MQ消息服务，消费者还没有来得及消费，此时MQ服务停止或意外宕机，那么这些未被消费的消息改怎么处理呢？分为消息非持久化和消息持久化两种情况，**消息非持久化**这些未被处理的消息直接丢失，**消息持久化**会把这些未被消费的消息暂时存储起来，当MQ消息服务重新启动时恢复这些消息，消费者可以继续消费。

- **队列消息持久化**

基于上面的示例代码，只需要为生产者客户端代码添加一行通过MessageProducer对象设置就可以了。（队列消息默认开启持久化这一行实际上可以省略）

```
producer.setDeliveryMode(DeliveryMode.PERSISTENT);
```

- **主题消息持久化**

主题消息默认不持久化，支持主题消息持久化，只需要修改消费者客户端代码如下：

```
 ...
connection.setClientID("client_0001");
 ...
Topic topic = session.createTopic(TOPIC_NAME);
TopicSubscriber subscriber =  session.createDurableSubscriber(topic, "remark...");
connection.start();
subscriber.setMessageListener(message -> {
        ...
});
```

首先必须要通过 `connection.setClientID("client_0001")` 指定订阅者ID，因为如果不指定唯一ID,订阅者（非持久化订阅者）每次连接时都会随机创建一个ID，在消息持久化状态下，订阅者需要保证从离线到重新在线ClientID唯一不变，这样MQ消息服务才能确定主题消息是否被所有持久化订阅者消费了（如果MQ服务停止或宕机时，主题消息未被所有持久化订阅者消费的会被存储起来，已经被所有持久化订阅者消费的主题消息会直接丢弃）。

然后通过 `session.createDurableSubscriber(topic, "remark...")` 创建一个TopicSubscriber对象，告诉MQ服务其订阅的此主题消息要做持久化处理。

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再试图将消息发送给接受者，成功则将消息从存储中删除，失败则继续尝试发送。MQ消息服务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继续发送出去。下面分别介绍一下KahaDB与JDBC持久化机制。

- **KahaDB存储**

KahaDB是一个基于文件的持久性数据库，消息存储使用一个事务日志和仅仅用一个索引文件来存储它所有的地址。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。在 `activemq.xml` 配置文件可查看其配置信息，更多的配置信息可参见官网 http://activemq.apache.org/ka...

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

`directory` 这里指明了kahadb数据存储路径，默认为ActiveMQ安装目录下 `/data/kahadb` ，其中主要包含4类文件和一个lock：

-

1. **db-<number>.log**：kahaDB存储消息到预定大小（默认32M）的数据记录文件中，文件命名为db-<number>.log，当数据文件已满时，一个新的文件会随之创建，number数值也会随之递增，当不再有引用到数据文件中的消息时，文件会被删除或者归档；
2. **db.data**：改文件包含了持久化的BTree索引，它是消息的索引文件，使用BTree作为索引指向db-<nubmer>.log里面存储的消息；
3. **db.free**：记录当前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;
4. **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；
5. **lock**：文件锁，表示当前获得kahaDB读写权限的broker；

- **JDBC存储**

如果采用JDBC机制存储，需要准备一个第三方数据库，这里以MySql数据库为例，更多信息参考 http://activemq.apache.org/jd...

1.首先将mysql数据库的驱动包 `mysql-connector-java-5.1.41.jar` 添加到 `ActiveMQ安装目录/lib` 目录下，用于连接mysql数据库；

2.打开 `activemq.xml` 配置文件，找到 `<beans>` 节点添加数据库连接池配置信息 `dataSource` bean，

```
<bean id="mysql-ds" class="org.apache.commons.dbcp2.BasicDataSource" destroy-method="close">
    <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
    <property name="url" value="jdbc:mysql://localhost:3306/activemq?relaxAutoCommit=true"/>
    <property name="username" value="root"/>
    <property name="password" value="123456"/>
    <property name="poolPreparedStatements" value="true"/>
</bean>
```

3. activemq.xml 配置文件中找到 <persistenceAdapter> 节点，修改为如下，

```
<persistenceAdapter>
    <jdbcPersistenceAdapter dataSource="#mysql-ds" createTablesOnStartup="true"/>
</persistenceAdapter>
```

mysql-ds 为上一步配置的beanId， createTablesOnStartup 是否在启动的时候自动创建数据表，默认值是true，一般是第一次启动的时候设置为true之后再改为false。

上诉三步都配置完后，启动ActiveMQ服务会自动创建三张表，分别为 activemq_msgs 消息表，用于保存queue和topic消息， 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保存， activemq_lock 在集群环境中才有用，保证只有一个borker可以获取消息，用于记录哪个breker是当前的master broker。

JDBC每次消息过来都需要去写库和读库，ActiveMQ Journal使用高速缓存写入技术大大提高了性能，克服了JDBC Store的不足。当消费者的消费速度能够及时跟上生产者消息的生产速度时，journal文件能够大大减少需要写入到DB中的消息，比如生产者生产了1000条消息，这1000条消息会先保存到journal文件，如果消费者的消费速度很快的情况下，在journal文件还没有同步到DB之前，消费者已经消费了90%的消息，那么这个时候只需要同步剩余的10%的消息到DB。

使用高效的Journal，需要修改持久化配置，打开 activemq.xml 配置文件，找到 <persistenceAdapter> 节点，修改为如下：

```
<persistenceFactory>
    <journalPersistenceAdapterFactory journalLogFiles="5" dataDirectory="activemq-data" dataS
</persistenceFactory>
```

5.2 事务

在上面的示例代码中，创建session时传了两个参数， createSession(false, Session.AUTO_ACKNOWLEDGE) ，第一个参数表示**是否开启事务**，第二个参数表示**签收方式**。

当开启事务，即第一个参数为 `true` 时，对于生产者而言执行 `send()` 方法后，消息不会直接进入消息队列中（没有真正发送到MQ服务），只有执行 `session.commit()` 消息才会真正发送成功进入消息队列中；对于消费者而言，消费完消息后，只有执行了 `session.commit()` 消息才会从消息队列中出队，如果不执行 `session.commit()` 会导致消息被重复消费。

事务开启的意义在于，对于多条必须同批次传输的消息，如果有一条传输失败，可以将事务回滚，再次传输，保证数据的完整性。

5.3 签收（ack）

签收和事务起到的作用是一样的，事务的优先级高于签收，即如果开启了事务，签收方式不管是哪种都是不起作用的，一般事务倾向于生产者使用，签收倾向于消费者使用。

签收方式总共有4种， `AUTO_ACKNOWLEDGE` 自动签收， `CLIENT_ACKNOWLEDGE` 手动签收， `DUPS_OK_ACKNOWLEDGE` 可重复的签收（不常用）， `SESSION_TRANSACTED` 一般表示开启了事务设置任何签收方式是无效的。

如果签收方式为 `CLIENT_ACKNOWLEDGE` 手动签收，必须执行 `message.acknowledge()` ，消息才能被真正的消费或者发送。

6 高级特性

6.1 异步投递

ActiveMQ支持以同步或异步模式向borker发送消息，所使用的模式对发送调用的延迟有很大的影响。由于延迟通常是生产者可以实现的吞吐量中的一个重要因素，因此使用异步发送可以显著提高系统的性能。

ActiveMQ默认以异步模式发送消息，以同步模式发送的情况是除非明确指定使用同步发送或者**事务外部发送持久消息**（即未使用事务的前提下发送持久化消息）。如果不使用事务，而是发送持久消息，那么每次发送都会同步并阻塞，直到broker向生产者发送确认消息已安全持久存储到磁盘为止，此确认机制提供了消息不会丢失的保证，但由于客户端被阻塞需要付出巨大的延迟代价。

异步投递可以最大化producer端的发送效率。通常在发送消息量比较密集的情况下使用异步发送，它可以很大的提升producer的吞吐量，不过这也带来了额外的问题，就是需要消耗很多的client端内存的同时也会导致broker端性能消耗增加，此外**不能有效的确保消失的发送成功**。在使用异步投递的情况下客户端需要容忍消息丢失的可能。

- **开启异步投递的三种方式**

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

2.通过ConnectionFactory对象属性

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

3.通过Connection对象属性，在此级别配置将覆盖ConnectionFactory级别的设置

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

- **如何保证一部投递情况下消息不丢失**

异步发送消息丢失的情况场景是，UseAsyncSend为true，使用 `producer.send(message)` 持续发送消息，消息不会阻塞，生产者会认为所有的消息均会被发送到了MQ服务，如果MQ服务突然宕机，此时生产者端尚未同步到MQ服务的消息均会丢失。所以，正确的异步发送方法需要接收回调的。

同步发送和异步发送的区别就在于，同步发送 `send()` 不阻塞就代表消息发送成功，异步发送需要接收回调并由客户端再判断一次是否发送。异步投递编码如下，

```
...省略...
TextMessage textMessage = session.createTextMessage("队列消息：message" + i);
//producer.send(textMessage);
//异步投递并确认消息发送结果
textMessage.setJMSMessageID(UUID.randomUUID().toString());
String msgId = textMessage.getJMSMessageID();
((ActiveMQMessageProducer) producer).send(textMessage, new AsyncCallback() {
    @Override
    public void onSuccess() {
        System.out.println("消息：" + msgId + "发送成功");

    }

    @Override
    public void onException(JMSException exception) {
        exception.printStackTrace();
        System.out.println("消息：" + msgId + "发送失败");
        // 对于失败消息后续可以进行重新发送
    }
});
...省略...
```

6.2 定时与延时投递

ActiveMQ开启定时与延迟投递，首先编辑 `activemq.xml` 配置文件， `<broker>` 标签内添加属性 `schedulerSupport` 并且设置为 `true`

```
<broker xmlns="http://activemq.apache.org/schema/core" brokerName="localhost"
dataDirectory="${activemq.data}" schedulerSupport="true">
```

代码中生产者端消息对象 `message` ，需要设置时间调度相关属性，主要属性如下：

属性名称	类型	描述
AMQ_SCHEDULED_DELAY	long	延迟投递时间
AMQ_SCHEDULED_PERIOD	long	重复投递时间间隔
AMQ_SCHEDULED_REPEAT	int	重复投递次数
AMQ_SCHEDULED_CRON	String	cron表达式

```
...
            TextMessage textMessage = session.createTextMessage("队列消息：message" + i);
            //消息延迟3秒，每隔4秒重复发送，重复5次
            textMessage.setLongProperty(ScheduledMessage.AMQ_SCHEDULED_DELAY, 3000L);
            textMessage.setLongProperty(ScheduledMessage.AMQ_SCHEDULED_PERIOD, 4000L);
            textMessage.setIntProperty(ScheduledMessage.AMQ_SCHEDULED_REPEAT, 5); //（加上第一
            producer.send(textMessage);
...
```

更多介绍参考官网 http://activemq.apache.org/de...

6.3 消费者消息重试策略

当下列任何一种情况发生时，borker会将消息重新传送至消费端:

- 使用事务会话并调用 `rollback()` ；
- 使用事务会话调用 `commit()` 之前关闭已处理的会话；
- 在手动签收 `CLIENT_ACKNOWLEDGE` 传递模式下调用 `session.recover()` ；
- 客户机连接超时(可能正在执行的代码比配置的超时时间更长)。

默认重发时间间隔为1秒总共重发6次，超过6次即最大重发次数后，消费端会给broker返送一个 `poison ack` 表示这个消息有毒，告诉broker不要再发了，这个时候broker会把这个消息放到DLQ（死信队列），以便稍后对其进行分析并人工干预处理。

```
...
        ActiveMQConnectionFactory connectionFactory = new ActiveMQConnectionFactory(BROKER_UR
        RedeliveryPolicy policy = new RedeliveryPolicy();
        policy.setInitialRedeliveryDelay(0);
        policy.setRedeliveryDelay(1000);
        policy.setUseExponentialBackOff(false);
        policy.setMaximumRedeliveries(2);
        connectionFactory.setRedeliveryPolicy(policy);
...
```

从ActiveMQ v5.7.0开始，可以在每个目的地的基础上配置RedeliveryPolicy,

```
...
        Connection connection = connectionFactory.createConnection();
        Session session = connection.createSession(true, Session.CLIENT_ACKNOWLEDGE);
        Queue queue = session.createQueue(QUEUE_NAME);

        RedeliveryPolicy queuePolicy = new RedeliveryPolicy();
        queuePolicy.setInitialRedeliveryDelay(0);
        queuePolicy.setRedeliveryDelay(1000);
        queuePolicy.setUseExponentialBackOff(false);
        queuePolicy.setMaximumRedeliveries(2);
        RedeliveryPolicyMap redeliveryPolicyMap =((ActiveMQConnection)connection).getRedelive
        redeliveryPolicyMap.put((ActiveMQQueue)queue, queuePolicy);
...
```

常用重发策略配置如下：

属性名称	默认值	描述
collisionAvoidanceFactor	0.15	设置防止冲突范围的正负百分比，只有启用useCollisionAvoidance参数时才生效。也就是在延迟时间上再加一个时间波动范围
initialRedeliveryDelay	1000L	初始重发延迟时间
maximumRedeliveries	6	最大重发次数，达到最大重发次数后消息进入死信队列。为-1时不限制次数，为0时表示不进行重发
maximumRedeliveryDelay	-1	最大重发延迟时间，只有useExponentialBackOff为true时有效（v5.5）。假设首次重发间隔为10ms，倍数为2，那么第二次重发时间间隔为20ms，第三次时间间隔为40ms，当重发时间间隔的计划最大传送延迟时间，以后每次重发时间间隔都为最大传送

属性名称	默认值	描述
redeliveryDelay	1000L	重发延迟时间，当initialRedeliveryDelay=0生效
useCollisionAvoidance	false	启用防止冲突功能
useExponentialBackOff	false	启用指数倍数递增的方式增加延迟时间
backOffMultiplier	5	重发时间间隔递增倍数，只有值大于1和启用useExponentialBackOff参数时才生效

更多重发介绍参考官网 http://activemq.apache.org/re...

ActiveMQ中的默认死信队列名称为 `ActiveMQ.DLQ`，所有无法交付的消息都将被发送到这个队列，这可能很难管理，因此，你也可以在 `activemq.xml` 配置文件为每个目的地配置单独的死信队列，如下：

```
<!-- 单独为每个queue目的地设置一个死信队列，前缀为DLQ -->
<destinationPolicy>
  <policyMap>
    <policyEntries>
      <!-- Set the following policy on all queues using the '>' wildcard -->
      <policyEntry queue=">">
        <deadLetterStrategy>
          <!-- Use the prefix 'DLQ.' for the destination name, and make the DLQ a queue rat
          <individualDeadLetterStrategy queuePrefix="DLQ." useQueueForQueueMessages="true"/
        </deadLetterStrategy>
      </policyEntry>
    </policyEntries>
  </policyMap>
</destinationPolicy>
```

死信队列详细介绍参考官网 http://activemq.apache.org/me...

7 内嵌broker

使用spring boot开发web应用的时候，spring boot提供了内嵌的tomcat或者jetty服务器，使用内嵌服务器运行项目时就不用再单独启动一个servlet服务器了，类似的ActiveMQ也提供了一个内嵌broker，使用如下：

```
package com.taicw.code.activemq.start;

import org.apache.activemq.broker.BrokerService;
```

```
    public static void main(String[] args) throws Exception {
        BrokerService brokerService = new BrokerService();
        brokerService.setUseJmx(true);
        brokerService.addConnector("tcp://localhost:61616");
        brokerService.start();
    }
}
```

详细介绍参考官网 https://activemq.apache.org/h...

你可能感兴趣的

- **分布式消息中间件 — MQ** bali tomcat mysql nginx spring java
- **消息中间件企业级应用** itdragon activemq mq
- **基于JMS(Java Message Service)的消息中间件----ActiveMQ** james 负载均衡 apache maven eclipse activemq
- **Spring Boot-实现Apache ActiveMQ消息中间件** 王继红 消息中间件 springboot activemq
- **深入消息中间件选型分析** skyarthur rocketmq rabbitmq kafka 消息队列 消息中间件
- **深入理解阿里分布式消息中间件** bali java spring nginx mysql 缓存
- **消息中间件及ActiveMQ介绍** JTQian java activemq jmeter mq
- **redux 中间件入门** 爱琴海小屋 react.js

评论

默认排序 时间排序

文明社会，理性评论

发布评论