

ActiveMQ消息队列的使用及应用

这里就不说怎么安装了，直接解压出来就行了。

谢绝转载，作者保留所有权力

目录：



- 一：JMQL两种消息模式
 - 1.1：点对点的消息模式
 - 1.2：订阅模式
- 二：点对点的实现代码
 - 2.1：点对点的发送端
 - 2.2：点对点的接收端
- 三：订阅/发布模式的实现代码
 - 3.1：订阅模式的发送端
 - 3.2：订阅模式的接收端
- 四：发送消息的数据类型
 - 4.1：传递javabean对象
 - 4.2：发送文件
- 五：ActiveMQ的应用
 - 5.1：保证消息的成功处理
 - 5.2：避免消息队列的并发
 - 5.2.1：主动接收队列消息
 - 5.2.2：使用多个接收端
 - 5.3：消息有效期的管理
 - 5.4：过期消息，处理失败的消息如何处理
- 六：ActiveMQ的安全配置
 - 6.1：管理后台的密码设置
 - 6.2：生产消费者的连接密码



一：JMQL两种消息模式

消息队列有两种消息模式，一种是点对点的消息模式，还有一种就是订阅的模式。

目录：

- 一：JMQL两种消息模式
 - 1.1：点对点的消息模式
 - 1.2：订阅模式
- 二：点对点的实现代码
 - 2.1：点对点的发送端
 - 2.2：点对点的接收端
- 三：订阅/发布模式的实现代码
 - 3.1：订阅模式的发送端
 - 3.2：订阅模式的接收端
- 四：发送消息的数据类型
 - 4.1：传递javabean对象
 - 4.2：发送文件
- 五：ActiveMQ的应用
 - 5.1：保证消息的成功处理
 - 5.2：避免消息队列的并发
 - 5.3：消息有效期的管理
 - 5.4：过期消息，处理失败的消息如何处理
- 六：ActiveMQ的安全配置
 - 6.1：管理后台的密码设置

1.1: 点对点的消息模式

45

好文要顶

关注我

收藏该文



点对点的模式主要建立在一个队列上面，当连接一个列队的时候，发送端不需要知道接收端是否正在接收，可以直接向ActiveMQ发送消息，发送的消息，将会先进入队列中，如果有接收端在监听，则会发向接收端，如果没有接收端接收，则会保存在activemq服务器，直到接收端接收消息，点对点的消息模式可以有多个发送端，多个接收端，但是一条消息，只会被一个接收端给接收到，哪个接收端先连上ActiveMQ，则会先接收到，而后来的接收端则接收不到那条消息

1.2: 订阅模式

订阅/发布模式，同样可以有着多个发送端与多个接收端，但是接收端与发送端存在时间上的依赖，就是如果发送端发送消息的时候，接收端并没有监听消息，那么ActiveMQ将不会保存消息，将会认为消息已经发送，换一种说法，就是发送端发送消息的时候，接收端不在线，是接收不到消息的，哪怕以后监听消息，同样也是接收不到的。这个模式还有一个特点，那就是，发送端发送的消息，将会被所有的接收端给接收到，不类似点对点，一条消息只会被一个接收端给接收到。

二：点对点的实现代码

这里使用java来实现一下ActiveMQ的点对点模式。

ActiveMQ版本为 5.13.3

项目使用MAVEN来构建

```
<dependencies>
  <dependency>
    <groupId>org.apache.activemq</groupId>
    <artifactId>activemq-core</artifactId>
    <version>5.7.0</version>
  </dependency>
</dependencies>
```

都是当前最新的版本

2.1: 点对点的发送端

按 Ctrl+C 复制代码

45

```
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.DeliveryMode;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.MessageProducer;
import javax.jms.Session;
import javax.jms.TextMessage;

import org.apache.activemq.ActiveMQConnectionFactory;

public class PTPSend {
    //连接账号
    private String userName = "";
    //连接密码
    private String password = "";
    //连接地址
    private String brokerURL = "tcp://192.168.0.130:61616";
    //connection的工厂
    private ConnectionFactory factory;
    //连接对象
    private Connection connection;
    //一个操作会话
    private Session session;
    //目的地，其实就是连接到哪个队列，如果是点对点，那么它的实现是Queue，如果是订阅模式，那么它的实现是Topic
    private Destination destination;
    //生产者，就是产生数据的对象
    private MessageProducer producer;

    public static void main(String[] args) {
        PTPSend send = new PTPSend();
        send.start();
    }
}
```

按 Ctrl+C 复制代码

2.2: 点对点的接收端

```
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.jms.MessageListener;
import javax.jms.Session;
import javax.jms.TextMessage;

import org.apache.activemq.ActiveMQConnectionFactory;

public class PTPReceive {
    //连接账号
    private String userName = "";
    //连接密码
    private String password = "";
    //连接地址
    private String brokerURL = "tcp://192.168.0.130:61616";
    //connection的工厂
    private ConnectionFactory factory;
    //连接对象
    private Connection connection;
    //一个操作会话
    private Session session;
    //目的地，其实就是连接到哪个队列，如果是点对点，那么它的实现是Queue，如果是订阅模式，那么它的实现是Topic
}
```

```

private Destination destination;

//消费者，就是接收数据的对象
private MessageConsumer consumer;

public static void main(String[] args) {

    PTPReceive receive = new PTPReceive();
    receive.start();

}

public void start(){

    try {

        //根据用户名，密码，url创建一个连接工厂
        factory = new ActiveMQConnectionFactory(userName, password, brokerURL);
        //从工厂中获取一个连接
        connection = factory.createConnection();
        //测试过这个步骤不写也是可以的，但是网上的各个文档都写了
        connection.start();
        //创建一个session
        //第一个参数:是否支持事务，如果为true，则会忽略第二个参数，被jms服务器设置为SESSION_TRANSACTED
        //第二个参数为false时，params的值为Session.AUTO_ACKNOWLEDGE, Session.CLIENT_ACKNOWLEDGE, DUPS_OK_ACKNOWLEDGE其中一个。
        //Session.AUTO_ACKNOWLEDGE为自动确认，客户端发送和接收消息不需要做额外的工作。哪怕是接收端发生异常，也会被当作正常发送成功。
        //Session.CLIENT_ACKNOWLEDGE为客户端确认。客户端接收到消息后，必须调用javax.jms.Message的acknowledge方法。jms服务器才会当作发送成功，并删除消息。
        //DUPS_OK_ACKNOWLEDGE允许副本的确认模式。一旦接收方应用程序的方法调用从处理消息处返回，会话对象就会确认消息的接收；而且允许重复确认。
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        //创建一个到达的目的地，其实想一下就知道了，activemq不可能同时只能跑一个队列吧，这里就是连接了一个名为"text-msg"的队列，这个会话将会到这个队列，当然，如果这个队列不存在，将会被创建
        destination = session.createQueue("text-msg");
        //根据session，创建一个接收者对象
        consumer = session.createConsumer(destination);

        //实现一个消息的监听器
        //实现这个监听器后，以后只要有消息，就会通过这个监听器接收到
        consumer.setMessageListener(new MessageListener() {

            @Override
            public void onMessage(Message message) {

                try {

                    //获取到接收的数据
                    String text = ((TextMessage)message).getText();
                    System.out.println(text);
                } catch (JMSEException e) {
                    e.printStackTrace();
                }

            }

        });
        //关闭接收端，也不会终止程序哦
        consumer.close();
    } catch (JMSEException e) {
        e.printStackTrace();
    }

}
}

```

三：订阅/发布模式的实现代码

3.1: 订阅模式的发送端

45

好文要顶

关注我

收藏该文



```

import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.DeliveryMode;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.MessageProducer;
import javax.jms.Session;
import javax.jms.TextMessage;

import org.apache.activemq.ActiveMQConnectionFactory;

public class TOPSend {
    //连接账号
    private String userName = "";
    //连接密码
    private String password = "";
    //连接地址
    private String brokerURL = "tcp://192.168.0.130:61616";
    //connection的工厂
    private ConnectionFactory factory;
    //连接对象
    private Connection connection;
    //一个操作会话
    private Session session;
    //目的地, 其实就是连接到哪个队列, 如果是点对点, 那么它的实现是Queue, 如果是订阅模式, 那它的实现是Topic
    private Destination destination;
    //生产者, 就是产生数据的对象
    private MessageProducer producer;

    public static void main(String[] args) {
        TOPSend send = new TOPSend();
        send.start();
    }

    public void start(){
        try {
            //根据用户名, 密码, url创建一个连接工厂
            factory = new ActiveMQConnectionFactory(userName, password, brokerURL);
            //从工厂中获取一个连接
            connection = factory.createConnection();
            //测试过这个步骤不写也是可以的, 但是网上的各个文档都写了
            connection.start();
            //创建一个session
            //第一个参数:是否支持事务, 如果为true, 则会忽略第二个参数, 被jms服务器设置为SESSION_TRANSACT
            //第二个参数为false时, paramB的值可为Session.AUTO_ACKNOWLEDGE, Session.CLIENT_ACKNOWLEDGE
            //Session.AUTO_ACKNOWLEDGE为自动确认, 客户端发送和接收消息不需要做额外的工作。哪怕是接收端发
            //Session.CLIENT_ACKNOWLEDGE为客户端确认。客户端接收到消息后, 必须调用javax.jms.Message
            送成功, 并删除消息。
            //DUPS_OK_ACKNOWLEDGE允许副本的确认模式。一旦接收方应用程序的方法调用从处理消息处返回, 会话对
            认。

            session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
            //创建一个到达的目的地, 其实想一下就知道了, activemq不可能同时只能跑一个队列吧, 这里就是连接了
            到这个队列, 当然, 如果这个队列不存在, 将会被创建

            //=====
            //点对点与订阅模式唯一不同的地方, 就是这一行代码, 点对点创建的是Queue, 而订阅模式创建的是Topic
            destination = session.createTopic("topic-text");
            //=====

```

```

//从session中,获取一个消息生产者
producer = session.createProducer(destination);
//设置生产者的模式,有两种可选
//DeliveryMode.PERSISTENT 当activemq关闭的时候,队列数据将会被保存
//DeliveryMode.NON_PERSISTENT 当activemq关闭的时候,队列里面的数据将会被清空
producer.setDeliveryMode(DeliveryMode.PERSISTENT);

//创建一条消息,当然,消息的类型有很多,如文字,字节,对象等,可以通过session.create..方法来创建出来
TextMessage textMsg = session.createTextMessage("哈哈");
long s = System.currentTimeMillis();
for(int i = 0 ; i < 100 ; i++){
    //发送一条消息
    producer.send(textMsg);
}
long e = System.currentTimeMillis();
System.out.println("发送消息成功");
System.out.println(e - s);
//即便生产者的对象关闭了,程序还在运行哦
producer.close();

} catch (JMSException e) {
    e.printStackTrace();
}
}
}

```

3.2: 订阅模式的接收端

```

import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.DeliveryMode;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.MessageProducer;
import javax.jms.Session;
import javax.jms.TextMessage;

import org.apache.activemq.ActiveMQConnectionFactory;

public class TOPSend {
    //连接账号
    private String userName = "";
    //连接密码
    private String password = "";
    //连接地址
    private String brokerURL = "tcp://192.168.0.130:61616";
    //connection的工厂
    private ConnectionFactory factory;
    //连接对象
    private Connection connection;
    //一个操作会话
    private Session session;
    //目的地,其实就是连接到哪个队列,如果是点对点,那么它的实现是Queue,如果是订阅模式,那它的实现是Topic
}

```

```
private Destination destination;
//生产者，就是产生数据的对象
private MessageProducer producer;
```

```
public static void main(String[] args) {
    TOPSend send = new TOPSend();
    send.start();
}
```

```
public void start(){
    try {
        //根据用户名，密码，url创建一个连接工厂
        factory = new ActiveMQConnectionFactory(userName, password, brokerURL);
        //从工厂中获取一个连接
        connection = factory.createConnection();
        //测试过这个步骤不写也是可以的，但是网上的各个文档都写了
        connection.start();
        //创建一个session
        //第一个参数:是否支持事务，如果为true，则会忽略第二个参数，被jms服务器设置为SESSION_TRANSACTED
        //第二个参数为false时，paramB的值可为Session.AUTO_ACKNOWLEDGE, Session.CLIENT_ACKNOWLEDGE, DUPS_OK_ACKNOWLEDGE其中一个。
        //Session.AUTO_ACKNOWLEDGE为自动确认，客户端发送和接收消息不需要做额外的工作。哪怕是接收端发生异常，也会被当作正常发送成功。
        //Session.CLIENT_ACKNOWLEDGE为客户端确认。客户端接收到消息后，必须调用javax.jms.Message的acknowledge方法。jms服务器才会当作发送成功，并删除消息。
        //DUPS_OK_ACKNOWLEDGE允许副本的确认模式。一旦接收方应用程序的方法调用从处理消息处返回，会话对象就会确认消息的接收；而且允许重复确认。

        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        //创建一个到达的目的地，其实想一下就知道了，activemq不可能同时只能跑一个队列吧，这里就是连接了一个名为"text-msg"的队列，这个会话将会到这个队列，当然，如果这个队列不存在，将会被创建

        //=====
        //点对点与订阅模式唯一不同的地方，就是这一行代码，点对点创建的是Queue，而订阅模式创建的是Topic
        destination = session.createTopic("topic-text");
        //=====

        //从session中，获取一个消息生产者
        producer = session.createProducer(destination);
        //设置生产者的模式，有两种可选
        //DeliveryMode.PERSISTENT 当activemq关闭的时候，队列数据将会被保存
        //DeliveryMode.NON_PERSISTENT 当activemq关闭的时候，队列里面的数据将会被清空
        producer.setDeliveryMode(DeliveryMode.PERSISTENT);

        //创建一条消息，当然，消息的类型有很多，如文字，字节，对象等，可以通过session.create..方法来创建出来
        TextMessage textMsg = session.createTextMessage("哈哈");
        long s = System.currentTimeMillis();
        for(int i = 0 ; i < 100 ; i++){
            //发送一条消息
            textMsg.setText("哈哈" + i);
            producer.send(textMsg);
        }
        long e = System.currentTimeMillis();
        System.out.println("发送消息成功");
        System.out.println(e - s);
        //即便生产者的对象关闭了，程序还在运行哦
        producer.close();

    } catch (JMSException e) {
        e.printStackTrace();
    }
}
```

四：发送消息的数据类型

上面的代码演示，全部都是发送字符串，但是ActiveMQ支持哪些数据呢？

大家可以看一下 `javax.jms.Message` 这个接口，只要是这个接口的数据，都可以被发送。

或者这样看起来有点麻烦，那么看到上面的代码，创建消息，是通过session这个对象来创建的，那我们来看一下这里有哪些可以被创建的呢？

```
// 纯字符串的数据
session.createTextMessage();

// 序列化的对象
session.createObjectMessage();

// 流，可以用来传递文件等
session.createStreamMessage();

// 用来传递字节
session.createBytesMessage();

// 这个方法创建出来的就是一个map，可以把它当作map来用，当你看了它的一些方法，你就懂了
session.createMapMessage();

// 这个方法，拿到的是javax.jms.Message，是所有message的接口
session.createMessage();
```

4.1: 传递javabean对象

传递一个java对象，可能是最多的使用方式了，而且这种数据接收与使用都方便，那么，下面的代码就来演示下如何发送一个java对象
当然了，这个对象必须序列化，也就是实现Serializable接口

```
// 通过这个方法，可以把一个对象发送出去，当然，这个对象需要序列化，因为一切在网络传输的，都是字节
ObjectMessage obj = session.createObjectMessage();

for(int i = 0 ; i < 100 ; i++){
    Person p = new Person(i, "名字");
    obj.setObject(p);
    producer.send(obj);
}
```

那么在接收端要怎么接收这个对象呢？

//实现一个消息的监听器

//实现这个监听器后,以后只要有消息,就会通过这个监听器接收到

```
consumer.setMessageListener(new MessageListener() {  
    @Override  
    public void onMessage(Message message) {  
        try {  
            //同样的,强转为ObjectMessage,然后拿到对象,强转为Person  
            Person p = (Person) ((ObjectMessage)message).getObject();  
            System.out.println(p);  
        } catch (JMSEException e) {  
            e.printStackTrace();  
        }  
    }  
});
```



好文要顶

关注我

收藏该文



4.2: 发送文件

发送文件, 这里用BytesMessage

```
BytesMessage bb = session.createBytesMessage();  
bb.writeBytes(new byte[] {2});
```

至于这里的新 Byte[] {2}, 肯定不是这样写的, 从文件里面拿流出来即可

接收的话



```
consumer.setMessageListener(new MessageListener() {  
    @Override  
    public void onMessage(Message message) {  
  
        BytesMessage bm = (BytesMessage)message;  
        FileOutputStream out = null;  
        try {  
            out = new FileOutputStream("d:/1.ext");  
        } catch (FileNotFoundException e2) {  
            e2.printStackTrace();  
        }  
        byte[] by = new byte[1024];  
        int len = 0 ;  
        try {  
            while((len = bm.readBytes(by)) != -1){  
                out.write(by, 0, len);  
            }  
        } catch (JMSEException | IOException e1) {  
            e1.printStackTrace();  
        }  
    }  
});
```

```
}  
});
```

五：ActiveMQ的应用

5.1:保证消息的成功处理

消息发送成功后，接收端接收到了消息。然后进行处理，但是可能由于某种原因，高并发也好，IO阻塞也好，反正这条消息在接收端处理失败了。而点对点的特性是一条消息，只会被一个接收端给接收，只要接收端A接收成功了，接收端B，就不可能接收到这条消息，如果是一些普通的消息还好，但是如果是一些很重要的消息，比如说用户的支付订单，用户的退款，这些与金钱相关的，是必须保证成功的，那么这个时候要怎么处理呢？

我们可以使用 `CLIENT_ACKNOWLEDGE` 模式

之前其实就有提到当创建一个session的时候，需要指定其事务，及消息的处理模式，当时使用的是

```
session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
```

AUTO_ACKNOWLEDGE

这一个代码的是，当消息发送给接收端之后，就自动确认成功了，而不管接收端有没有处理成功，而一旦确认成功后，就会把队列里面的消息给清除掉，避免下一个接收端接收到同样的消息。

那么，它还有另外一个模式，那就是 `CLIENT_ACKNOWLEDGE`

这行要写在接收端里面，不是写在发送端的

```
session = connection.createSession(false, Session.CLIENT_ACKNOWLEDGE);
```

这行代码以后，如果接收端不确认消息，那么activemq将会把这条消息一直保留，直到有一个接收端那么要怎么确认消息呢？

在接收端接收到消息的时候，调用 `javax.jms.Message` 的 `acknowledge` 方法

```
@Override  
  
public void onMessage(Message message) {  
    try {  
        //获取到接收的数据  
        String text = ((TextMessage)message).getText();  
        System.out.println(text);  
        //确认接收，并成功处理了消息  
        message.acknowledge();  
    } catch (JMSException e) {  
        e.printStackTrace();  
    }  
}
```

这样，当消息处理成功之后，确认消息，如果不确定，activemq将会发给下一个接收端处理

注意：只在点对点中有效，订阅模式，即使不确认，也不会保存消息

5.2:避免消息队列的并发

JMQ设计出来的原因，就是用来避免并发的，和沟通两个系统之间的交互。

5.2.1:主动接收队列消息

先看一下之前的代码：

```
//实现一个消息的监听器
//实现这个监听器后，以后只要有消息，就会通过这个监听器接收到
consumer.setMessageListener(new MessageListener() {
    @Override
    public void onMessage(Message message) {
        try {
            //获取到接收的数据
            String text = ((TextMessage)message).getText();
            System.out.println(text);
            //确认接收，并成功处理了消息
            message.acknowledge();
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }
});
```

之前的代码里面，实现了一个监听器，监听消息的传递，这样只要每有一个消息，都会即时的传递到程序中。

但是，这样的处理，在高并发的时候，因为它是被动接收，并没有考虑到程序的处理能力，可能会压跨系统，那要怎么办呢？

答案就是把被动变为主动，当程序有着处理消息的能力时，主动去接收一条消息进行处理

实现的代码如下：

```
if (当程序有能力处理) { //当程序有能力处理时接收
    Message receive = consumer.receive();
    //这个可以设置超时时间，超过则不等待消息
    receive.receive(10000);
    //其实receive是一个阻塞式方法，一定会拿到值的
    if (null != receive) {
        String text = ((TextMessage)receive).getText();
        receive.acknowledge();
        System.out.println(text);
    } else {
        //没有值嘛
        //
    }
}
```

通过上面的代码，就可以让程序自己判断，自己是否有能力接收这条消息，如果不能接收，那就给别的接收端接收，或者等自己有能力处理的时候接收

5.2.2:使用多个接收端

ActiveMQ是支持多个接收端的，如果当程序无法处理这么多数据的时候，可以考虑多个线程，或者增加服务器来处理。

5.3:消息有效期的管理

这样的场景也是有的，一条消息的有效时间，当发送一条消息的时候，可能希望这条消息在指定的时间被处理，如果超过了指定的时间，那么这条消息就失效了，就不需要进行处理了，那么我们可以使用ActiveMQ的设置有效期来实现

代码如下：

```

    TextMessage msg = session.createTextMessage("哈哈");
    for(int i = 0 ; i < 100 ; i ++){
        //设置该消息的超时时间
        producer.setTimeToLive(i * 1000);
        producer.send(msg);
    }

```

这里每一条消息的有效期都是不同的，打开ip:8161/admin/就可以查看到，里面的消息越来越少了。

过期的消息是不会被接收到的。

过期的消息会从队列中清除，并存储到ActiveMQ.DLQ这个队列里面，这个稍后会解释。

5.4:过期消息，处理失败的消息如何处理

过期的、处理失败的消息，将会被ActiveMQ置入“ActiveMQ.DLQ”这个队列中。

这个队列是ActiveMQ自动创建的。

如果需要查看这些未被处理的消息，可以进入这个队列中查看

```

//指定一个目的地，也就是一个队列的位置
destination = session.createQueue("ActiveMQ.DLQ");

```

这样就可以进入队列中，然后实现接口，或者通过receive()方法，就可以拿到未被处理的消息，从而

正确的处理

六: ActiveMQ的安全配置

6.1:管理后台的密码设置

我们都知道，打开ip:8161/admin/ 就是activemq的管理控制台，它的默认账号和密码都是admin, 在生产环境肯定需要更改密码的，这要怎么做呢？

在activemq/conf/jetty.xml中找到

```
<pre name="code" class="html"> <bean id="securityConstraint" class="org.eclipse.jetty.util.security.Constraint">
    <property name="name" value="BASIC" />
    <property name="roles" value="admin" />
    <!-- 把这个改为true,当然，高版本的已经改为了true -->
    <property name="authenticate" value="true" />
</bean>
```

高版本的已经默认成为了true。所以我们直接进行下一步即可

在activemq/conf/jetty-realm.properties文件中配置，打开如下

```
## -----
## Licensed to the Apache Software Foundation (ASF) under one or more
## contributor license agreements. See the NOTICE file distributed with
## this work for additional information regarding copyright ownership.
## The ASF licenses this file to You under the Apache License, Version 2.0
## (the "License"); you may not use this file except in compliance with
## the License. You may obtain a copy of the License at
##
## http://www.apache.org/licenses/LICENSE-2.0
##
## Unless required by applicable law or agreed to in writing, software
## distributed under the License is distributed on an "AS IS" BASIS,
## WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
## See the License for the specific language governing permissions and
## limitations under the License.
## -----

# Defines users that can access the web (console, demo, etc.)
# username: password [,rolename ...]
#用户名，密码，角色
admin: admin, admin
user: user, user
```

注意：大家重点看倒数第二行，那里三个分别是用户名，密码，角色，其中admin角色是固定的

6.2:生产消费者的连接密码

注意:activemq默认是不需要密码，生产消费者就可以连接的

我们需要经过配置，才能设置密码，这一步在生产环境中一定要配置

45

找到activemq/conf/activemq.xml, 并打开
在<broker>节点中，在<systemUsage>节点上面，增加如下的一个插件

```
<plugins>

    <simpleAuthenticationPlugin>

        <users>

            <authenticationUser username="{activemq.username}" password="{activemq.password}" groups="users,admins"/>

        </users>

    </simpleAuthenticationPlugin>

</plugins>
```

这样就开启了密码认证
然后账号密码的配置在activemq/conf/credentials.properties文件中
打开这个文件如下

```
## -----
## Licensed to the Apache Software Foundation (ASF) under one or more
## contributor license agreements.  See the NOTICE file distributed with
## this work for additional information regarding copyright ownership.
## The ASF licenses this file to You under the Apache License, Version 2.0
## (the "License"); you may not use this file except in compliance with
## the License.  You may obtain a copy of the License at
##
## http://www.apache.org/licenses/LICENSE-2.0
##
## Unless required by applicable law or agreed to in writing, software
## distributed under the License is distributed on an "AS IS" BASIS,
## WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
## See the License for the specific language governing permissions and
## limitations under the License.
## -----

# Defines credentials that will be used by components (like web console) to access the broker


#账号
activemq.username=admin

#密码
activemq.password=123456
guest.password=password
```

这样就配置完毕了。

感谢您的阅读，如果您觉得阅读本文对您有帮助，请点一下“推荐”按钮。本文欢迎各位转载，但是
出作者和原文连接。

分类: 消息队列

 **朱小杰**
关注 - 5
粉丝 - 187

+加关注

« 上一篇 : [mysql转换类型](#)
» 下一篇 : [java多线程等待协调工作:CountDownLatch类的高级应用](#)

posted @ 2016-06-06 18

评论列表

45

#1楼 2016-06-06 18:39 Sam Xiao

这玩意儿，确实不得了。

好文要顶 关注我 收藏该文 微博 微信

#2楼 2016-06-08 11:09 梦醒心晴

楼主你好，发布订阅模式一定没有办法保证到达么？一个订单号出来以后会触发多条业务线运行，如果订阅模式没法保证一定到达的话，是不是我要选择点对点模式，发多条消息？

支持(0) 反对(0)

#3楼[楼主] 2016-06-08 11:10 朱小杰

@ 梦醒心晴
当接收端开启的时候，是可以保证接收到的

支持(0) 反对(0)

#4楼 2016-06-08 11:13 梦醒心晴

问题是客户端包括手机端，pc应用程序，国内的网络环境，你懂的，时不时就会掉一次线

支持(0) 反对(0)

#5楼[楼主] 2016-06-08 11:15 朱小杰

@ 梦醒心晴
抛开服务器性能问题不说，你也不能用点对点，因为点对点，一条记录，只会被一个接收端接收到，它保存一条记录，只被一次处理

支持(0) 反对(0)

#6楼 2016-06-08 11:16 梦醒心晴

可是发布订阅没法保证到达，纠结

支持(0) 反对(0)

#7楼[楼主] 2016-06-08 11:19 朱小杰

@ 梦醒心晴
你这种情况不应该由消息队列来处理，写数据库表不是挺好的吗

支持(0) 反对(0)

#8楼 2016-06-08 11:20 梦醒心晴

就跟你上面说的 订单之类的业务是必须保证到达的，不然客户端可能漏单，能加qq聊么 516999605

支持(0) 反对(0)

#9楼[楼主] 2016-06-08 11:23 朱小杰

@ 梦醒心晴
用http长连接，或者用socket建立对等连接，由服务器主动通知客户端，

支持(0) 反对(0)

#10楼 2016-09-28 17:22 图_图

博主，在吗。我有一些amq点对点传输上的可靠性想要请教你

支持(0) 反对(0)

#11楼 2016-12-06 15:09 zala.eric

楼主在么，你的订阅接收端 跟发送端一样的，还有订阅接收端是可以多个，具体在应用中是怎么设计的 订阅接收端都是一样的形式 部署多套？？

支持(3) 反对(0)

#12楼 2017-06-21 18:14 奔跑的小河

亲，传递javabean对象会不成功!可能得设置connectionFactory.setTrustAllPackages();

支持(0) 反对(0)

#13楼 2017-08-24 20:08 向马湾

@ 朱小杰
那个小晴问你：楼主你好，发布订阅模式一定没有办法保证到达么？一个订单号出来以后会触发多条业务线运行，如果订阅模式点对点模式，发多条消息？
博主回答：@ 梦醒心晴
抛开服务器性能问题不说，你也不能用点对点，因为点对点，一条记录，只会被一个接收端接收到，它保存一条记录，只被一次处理

但是，我的理解是：心晴的意思是，她有N个需要通知到的订单号的业务线，就依次对每一条业务线发送点对点的消息，所以后者都采取CLIENT_ACKNOWLEDGE 模式，目的是保证每个消息都必达，这样岂不是达到了通知所有业务线的目的，同时避免重复消息？

支持(0) 反对(0)

#14楼 2017-11-09 22:59 达兔哥

写的很详细

支持(0) 反对(0)

#15楼 2017-12-12 13:32 会飞的鱼188

很用心 写的很好 学习了

支持(0) 反对(0)

45

学习了

好文要顶

关注我

收藏该文

#16楼 2017-12-14 10:07 snakejia

你好，请问一下，如何异步连接？有时候服务器消息队列挂了，客户端无法启动。

支持(0) 反对(0)

#17楼 2018-01-11 19:33 穆建情

文章总结的不错，写的也很详细，但是3.1和 3.2真的是一样的，博主赶紧改掉吧

支持(0) 反对(0)

#18楼 2018-05-04 16:24 飞龙在天001

感谢楼主分享

支持(0) 反对(0)

#19楼 2018-06-13 14:33 javahepeng

git上有没有完整的代码？

支持(0) 反对(0)

#20楼 2018-06-27 17:03 tuwosh

服务端和接收端设置的session模式不一致按哪个执行呢？

支持(0) 反对(0)

#21楼 2018-06-27 17:51 tuwosh

订阅模式接收端代码的文本接收部分的代码是不是有问题，看起来是直接拷贝的发送端的代码。

```
1  TextMessage textMsg = session.createTextMessage("哈哈");
2  long s = System.currentTimeMillis();
3  for(int i = 0 ; i < 100 ; i++){
4      //发送一条消息
5      textMsg.setText("哈哈" + i);
6      producer.send(textMsg);
7  }
8  long e = System.currentTimeMillis();
9  System.out.println("发送消息成功");
10 System.out.println(e - s);
```

支持(1) 反对(0)

#22楼 2018-06-28 08:59 luckyFireHao

很全面，完全明白了，谢谢

支持(0) 反对(0)

#23楼 2018-07-28 14:47 Bactryki

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。

【推荐】超50万VC++源码：大型组态工控、电力仿真CAD与GIS源码库！

【推荐】华为云11.11普惠季 血拼风暴 一促即发

【拼团】腾讯云服务器拼团活动又双叒来了！

【推荐】腾讯云新注册用户域名抢购1元起

最新IT新闻：

- 谷歌首次在非洲推出摩托车导航 首站肯尼亚
- 阿里云IoT联合意法半导体推出“未来工程师”计划
- 二十年内机器人将代替人类上战场
- Uber的秘密武器是一队经济学家
- 中移动与CBA签约：真4K将首次应用国内球赛

» 更多新闻...

最新知识库文章:

- 为什么说 Java 程序员必须掌握 Spring Boot ?
- 在学习中，有一个比掌握知识更重要的能力
- 如何招到一个靠谱的程序员
- 一个故事看懂“区块链”
- 被踢出去的用户
- » 更多知识库文章...

好文要顶

关注我

收藏该文

