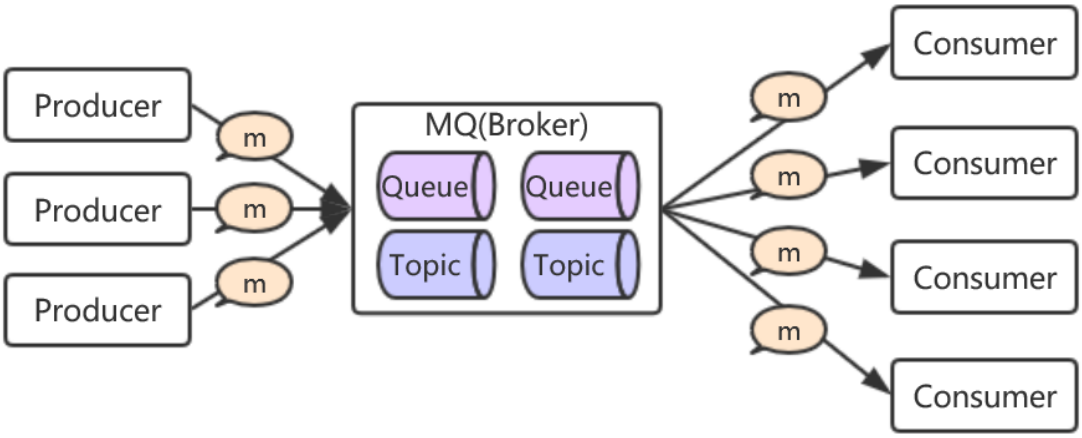
ActiveMQ-特性详解



1 连接

了解BrokerUrl上能带哪些参数（参数的作用）

http://activemq.apache.org/connection-configuration-uri.html

2 Producer Send Message To MQ

2.1 Destination

https://activemq.apache.org/destination-features

2.1.1 Destination创建

我们的消息将发往哪里?

Queue 和 Topic 需要我们提前创建好吗?

https://activemq.apache.org/how-do-i-create-new-destinations

不需要，可以提前创建

https://activemq.apache.org/configure-startup-destinations

2.1.2 复合目标Composite Destinations

我们可能会有哪些发往目标需要?

发给一个Queue；发给一个Topic

会不会有需要：一个消息需发给多个queue或topic?

客户端复合目标:

https://activemq.apache.org/composite-destinations

```
// send to 3 queues as one logical operation
Queue queue = new ActiveMQQueue("FOO.A,FOO.B,FOO.C");
producer.send(queue, someMessage);
```

can use a prefix of `queue://` or `topic://` to differentiate the type of destination

```
// send to queues and topic one logical operation
Queue queue = new ActiveMQQueue("FOO.A,topic://NOTIFY.FOO.A");
producer.send(queue, someMessage);
```

Broker 端复合目标 【虚拟目标】

https://activemq.apache.org/virtual-destinations

```
<!-- 在 activemq.xml 的broker节点下添加 -->
<destinationInterceptors>
 <virtualDestinationInterceptor>
   <virtualDestinations>
     <compositeQueue name="MY.QUEUE">
       <forwardTo>
         <queue physicalName="FOO" />
         <topic physicalName="BAR" />
       </forwardTo>
     </compositeQueue>
   </virtualDestinations>
 </virtualDestinationInterceptor>
</destinationInterceptors>
```

Filter physical Destinations

虚拟目标+Selector

```
<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>
```

2.2 发送方式

同步/异步

3 Consumer receive Message From MQ

3.1 Destination

https://activemq.apache.org/destination-features

3.1.1 从多个目标消费消息【复合目标】

从多个队列消费

订阅多个主题

```
// 在消费时，并不可以复合对列与主题
new ConsumerThread("tcp://mq.study.com:61616", "queue2,topic://topic2").start();
// 在消费时，可以复合消费多个队列的消息
//new ConsumerThread("tcp://mq.study.com:61616", "queue2,queue3").start();
// 可以订阅多个主题
//new ConsumerThread("tcp://mq.study.com:61616", "topic1,topic2").start();
// 注意：如果同一条消息发给了多个队列或主题，消费者只会收到一次。
```

3.1.2 镜像队列

ActiveMQ每一个queue中消息只能被一个消费者消费，然而，有时候，你希望能够监视生产者和消费者之间的消息流。即你也想获得queue中的消息（但queue不能像topic一样发给多个消费者）。为解决这种需要，ActiveMQ提供了镜像队列功能：将队列中的消息copy一份到一个主题中，我们想监控某队列的消息就订阅这个主题。

https://activemq.apache.org/mirrored-queues

配置：

```
<broker ........, useMirroredQueues="true">
    ....
    <destinationInterceptors>
        <!--prefix指定主题的前缀，默认是 VirtualTopic.Mirror.
            postfix 指定主题的后缀 默认无
        -->
        <mirroredQueue copyMessage="true" postfix=".qmirror" prefix=""/>
    </destinationInterceptors>

    ....
</broker>
```

如你想获得队列 **Foo.Bar** 的消息，你可以订阅主题 **VirtualTopic.Mirror.Foo.Bar** 。

3.1.3 主题-集群负载均衡

Virtual Topic

https://activemq.apache.org/virtual-destinations

3.1.4 Destination Options【了解】

ActiveMQ 是采用推的模式来实时传递消息。

请考虑消费者端可能会有什么情况?

https://activemq.apache.org/destination-options

3.1.5 Per Destination Policies【了解】

https://activemq.apache.org/per-destination-policies

3.1.6 Prefetch

问题：ActiveMQ是如何将消息推送给消费者的?

推送一个，收到确认后，再推送下一个?

这样效率怎样?

怎样来提高效率?

提前送达一批消息给消费者处理。

ActiveMQ默认的Prefetch策略:

- 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`)

请思考：如果某消息的消费者是慢消费者，它处理一个消息要花费很长时间，因此我们往往会采用集群负载均衡的方式来快速处理消息，此时默认的Prefetch会对此有影响吗?

会造成负载不均，消息得不到及时处理。

怎么修改prefetch策略

http://activemq.apache.org/what-is-the-prefetch-limit-for

The prefetch limit can also be configured on the connection URI used to establish a connection the broker. To change the prefetch limit for all consumer types configure a connection URI as follows:

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

To change the prefetch limit for queue consumers only configure the connection URI as follows:

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

It can also be configured on a per consumer basis using Destination Options:

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

3.2 Message Dispatch 派发 【了解】

队列被多个消费者共享时，消息如何派发给消息者?

https://activemq.apache.org/dispatch-policies

3.3 Dispatch async

http://activemq.apache.org/consumer-dispatch-async

3.4 Consumer

3.4.1 消费者优先级【了解】

有时我们希望那些硬件资源充裕，网络环境好的消费者优先来接收处理消息，可以通过设置消费者的优先级达成。

http://activemq.apache.org/consumer-priority

The priority for a consumer is set using Destination Options as follows:

```
queue = new ActiveMQQueue("TEST.QUEUE?consumer.priority=10");
consumer = session.createConsumer(queue);
```

The range of priority values is: **0** to **127**. The highest priority is **127**. The default priority is **0**.

The broker orders a queue's consumers according to their priorities, dispatching messages to the highest priority consumers first. Once a particular consumer's prefetch buffer is full the broker will start dispatching messages to the consumer with the next lowest priority whose prefetch buffer is not full.

3.4.2 严格顺序处理消息

场景：队列中的消息必须按队列中的顺序一个接一个串行处理。

这怎么实现?

队列一个消费者。

如何保证高可用，万一这个消费者挂了怎么办?

ActiveMQ 中提供了独家消费者 和 消息组 来达成。

Exclusive Consumer 独家消费者

http://activemq.apache.org/exclusive-consumer

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

多个消费者都以独家方式共享同一队列，只有一个会消费消息，当它挂了时，会选择下一个来发送。

message group 消息分组

消息分组是对独家消费者的增强，能负载均衡

http://activemq.apache.org/message-groups

同组的消息发给同一消费者

4 Broker delivery message 递送消息

递送模式

NON_PERSISTENT

非持久化消息将存放在哪里?

如果消费者迟迟未消费消息，Broker中会无限堆积消息吗?

你觉得应该怎么控制?

Producer Flow Control 生产者流量控制 【了解】

http://activemq.apache.org/producer-flow-control.html

对于 NON_PERSISTENT 消息，ActiveMQ默认采用 限量内存 + 限量临时文件 来限流。

- **不进行流量控制**

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

- **不使用临时文件存储**

```
<policyEntry queue=">" producerFlowControl="true" memoryLimit="1mb">
  <pendingQueuePolicy>
    <vmQueueCursor/>
  </pendingQueuePolicy>
</policyEntry>
```

- **空间不够时，不阻塞而是发送异常给生产者**

```
<systemUsage>
 <systemUsage sendFailIfNoSpace="true">
   <memoryUsage>
     <memoryUsage limit="20 mb"/>
   </memoryUsage>
 </systemUsage>
</systemUsage>
```

- **空间不够时，阻塞等待一定时间后还是没有空间再发送异常给生产者**

```
<systemUsage>
 <systemUsage sendFailIfNoSpaceAfterTimeout="3000">
   <memoryUsage>
     <memoryUsage limit="20 mb"/>
   </memoryUsage>
 </systemUsage>
</systemUsage>
```

- **通过系统用量来限流量**

```
<systemUsage>
  <systemUsage>
    <memoryUsage>
      <memoryUsage limit="64 mb" />
      <!-- 也可以使用百分比 -->
      <!-- <memoryUsage percentOfJvmHeap="70" />  -->
    </memoryUsage>
    <storeUsage>
      <storeUsage limit="100 gb" />
    </storeUsage>
    <tempUsage>
      <tempUsage limit="10 gb" />
    </tempUsage>
  </systemUsage>
</systemUsage>
```

You can set limits of memory for `NON_PERSISTENT` messages, disk storage for `PERSITENT` messages and total usage for temporary messages, the broker will use before it slowdown producers. *Using the default settings shown above, the broker will block the send() call until some messages are consumed and space becomes available on the broker.* The default values are shown above, you will probably need to increase these values for your environment.

PERSISTENT

5 持久化

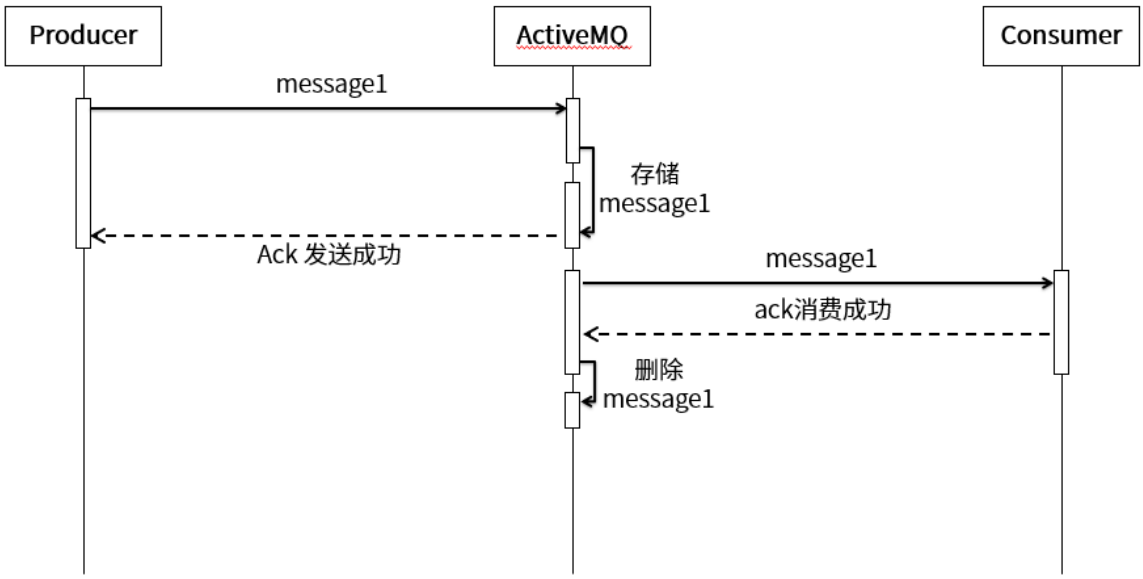
1、了解消息在MQ中的存储、移除流程。

2、了解持久化消息的持久化存储时机。

3、了解activemq支持哪些持久化存储方式

4、掌握持久化方式如何配置

5.1 消息传递流程



5.2 ActiveMQ支持的持久化方式

官网介绍链接：http://activemq.apache.org/persistence

- AMQ
- JDBC ActiveMQ V4 加入
- KahaDB ActiveMQ V5.3 加入，5.4开始为默认持久化方式
- LevelDB ActiveMQ V5.8 加入 官方已废弃并不在支持
- Replicated LevelDB Store ActiveMQ V5.8 加入 官方已废弃并不在支持

5.2.1 配置方式

http://activemq.apache.org/schema/core/activemq-core.xsd

```
<xs:element name="persistenceAdapter" maxOccurs="1" minOccurs="0">
    <xs:annotation>
        <xs:documentation>
        <![CDATA[

                    Sets the persistence adaptor implementation to use for
this broker
```

```
        ]]>
        </xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:choice maxOccurs="1" minOccurs="0">
            <xs:element ref="tns:jdbcPersistenceAdapter"/>
            <xs:element ref="tns:journalPersistenceAdapter"/>
            <xs:element ref="tns:kahaDB"/>
            <xs:element ref="tns:levelDB"/>
            <xs:element ref="tns:mKahaDB"/>
            <xs:element ref="tns:memoryPersistenceAdapter"/>
            <xs:element ref="tns:replicatedLevelDB"/>
            <xs:any namespace="##other"/>
        </xs:choice>
    </xs:complexType>
</xs:element>
```

5.2.2 AMQ

基于文件存储。它具有写入速度快和容易恢复的特点，但是由于其重建索引时间过长，而且索引文件占用磁盘空间过大，所以已经不推荐使用。

官网详细介绍：https://activemq.apache.org/amq-message-store

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

5.2.3 JDBC

官网详细说明：https://activemq.apache.org/jdbc-support

对于长期的持久性，建议使用JDBC和高性能日志。如果您愿意，可以只使用JDBC，但是它非常慢。

JDBC + 高性能日志用： journalPersistenceAdapter 参考：http://activemq.apache.org/persistence

MySQL 示例

1. 引入mysql的驱动jar，放到 activemq的 lib目录下
2. 配置 activemq.xml

```
    <broker ...>
        ...
        <persistenceAdapter>
            <!-- <kahaDB directory="${activemq.data}/kahadb"/> -->
            <jdbcPersistenceAdapter dataSource="#mysql-ds" />
        </persistenceAdapter>
        ...
    </broker>
```

```
    <!-- MySql DataSource Sample Setup -->
    <bean id="mysql-ds" class="org.apache.commons.dbcp2.BasicDataSource"
destroy-method="close">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <!-- 【注意】一定要带参数 relaxAutoCommit=true -->
        <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>
```

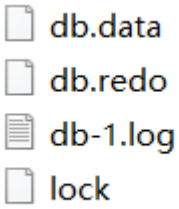
5.2.4 KahaDB 【掌握】

KahaDB是一个基于文件的持久性数据库，它位于使用它的消息代理的本地。它已经为快速持久性进行了优化。它是ActiveMQ 5.4以来的默认存储机制。KahaDB使用更少的文件描述符，并且比它的前身AMQ消息存储提供更快的恢复。**【官方推荐使用的持久化存储方式】**

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

官方详细说明：http://activemq.apache.org/kahadb

请了解KahaDB存储结构




```
db log files：以db-递增数字.log命名。
archive directory: 当配置支持archiving(默认不支持)并且存在，该文件夹才会创建。用于存储不再需要的data logs。
db.data：存储btree索引
db.redo：用于hard-stop broker后，btree索引的重建
```

所有目标的消息都存储在一起。

5.2.5 Multi(m) kahaDB Persistence Adapter

ActiveMQ 5.6:可以跨多个kahdb持久性适配器分发目标存储。你什么时候会这么做?如果您有一个快速的生产者/消费者目的地和另一个定期的生产者目的地，它们具有不规则的批处理消费，那么随着未消费的消息分布在多个日志文件中，磁盘使用可能会失控。每个一个单独的日志，可以确保最小限度地使用日志。此外，有些目的地可能很重要，需要磁盘同步，而有些则不是。在这些情况下，您可以使用mKahaDB持久性适配器，并使用通配符过滤目的地，就像使用目的地策略一样。

Configuration

Each instance of `kahaDB` can be configured independently. If no destination is supplied to a `filteredKahaDB`, the implicit default value will match any destination, queue or topic. This is a handy catch all. If no matching persistence adapter can be found, destination creation will fail with an exception. The `filteredKahaDB` shares its wildcard matching rules with Per Destination Policies.

From ActiveMQ 5.15, `filteredKahaDB` support a StoreUsage attribute named `usage`. This allows individual disk limits to be imposed on matching queues.

```
<broker brokerName="broker">

 <persistenceAdapter>
  <mKahaDB directory="${activemq.base}/data/kahadb">
    <filteredPersistenceAdapters>
      <!-- match all queues -->
      <filteredKahaDB queue=">">
        <usage>
         <storeUsage limit="1g" />
        </usage>
        <persistenceAdapter>
          <kahaDB journalMaxFileLength="32mb"/>
        </persistenceAdapter>
      </filteredKahaDB>

      <!-- match all destinations -->
      <filteredKahaDB>
        <persistenceAdapter>
          <kahaDB enableJournalDiskSyncs="false"/>
        </persistenceAdapter>
      </filteredKahaDB>
    </filteredPersistenceAdapters>
  </mKahaDB>
 </persistenceAdapter>

</broker>
```

Automatic Per Destination Persistence Adapter

Set `perDestination="true"` on the catch all, i.e., when no explicit destination is set, `filteredKahaDB` entry. Each matching destination will be assigned its own `kahaDB` instance.

```
<broker brokerName="broker">

 <persistenceAdapter>
  <mKahaDB directory="${activemq.base}/data/kahadb">
    <filteredPersistenceAdapters>
      <!-- kahaDB per destinations -->
      <filteredKahaDB perDestination="true">
        <persistenceAdapter>
          <kahaDB journalMaxFileLength="32mb"/>
        </persistenceAdapter>
      </filteredKahaDB>
    </filteredPersistenceAdapters>
```

```
    </mKahaDB>
  </persistenceAdapter>

</broker>
```

6 协议

学习目标：

- 了解ActiveMQ支持的协议与传输方式，及适用场景。
- 知道如何配置。
- 了解各协议的说明文档链接

官方说明文档：

- 客户端连接可用uri协议介绍：https://activemq.apache.org/uri-protocols
- ActiveMQ Broker支持的协议列表：https://activemq.apache.org/protocols
- transport传输器配置说明列表：http://activemq.apache.org/configuring-transports.html

6.1 ActiveMQ支持多种协议

- AMQP
- AUTO
- MQTT
- OpenWire
- REST
- RSS and Atom
- Stomp
- WSIF
- WS Notification
- XMPP

【注意】 当选择使用某种协议时，一定要从ActiveMQ官网了解清楚它实现的该协议的那个版本。

ActiveMQ支持多种传输方式:

TCP、NIO、SSL、HTTP(S)、UDP、VM

官网介绍链接：http://activemq.apache.org/activemq-connection-uris

6.2 协议的配置方式

http://activemq.apache.org/configuring-transports.html

```
        <!--
            The transport connectors expose ActiveMQ over a given protocol to
            clients and other brokers. For more information, see:

            http://activemq.apache.org/configuring-transports.html
        -->
```

```
        <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;wireFormat.maxFrameSize=104857600"/>
            <transportConnector name="amqp" uri="amqp://0.0.0.0:5672?
maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>
            <transportConnector name="stomp" uri="stomp://0.0.0.0:61613?
maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>
            <transportConnector name="mqtt" uri="mqtt://0.0.0.0:1883?
maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>
            <transportConnector name="ws" uri="ws://0.0.0.0:61614?
maximumConnections=1000&amp;wireFormat.maxFrameSize=104857600"/>
        </transportConnectors>
```

6.3 OpenWire

OpenWire 是Apache的一种跨语言的协议，允许从不同的语言和平台访问ActiveMQ，是ActiveMQ 4.x以后的版本默认的传输协议。

openwire简介：http://activemq.apache.org/openwire

openwire协议说明：http://activemq.apache.org/openwire-version-2-specification.html

6.4 AMQP

服务端配置说明：http://activemq.apache.org/amqp

客户端使用：

1. 引入AQMP实现客户端

```
        <dependency>
            <groupId>org.apache.qpid</groupId>
            <artifactId>qpid-jms-client</artifactId>
            <version>0.37.0</version>
        </dependency>
```

2. 连接工厂换成如下的 JmsConnectionFactory

```
// 1、创建连接工厂
                JmsConnectionFactory connectionFactory = new
JmsConnectionFactory(null, null, brokerUrl);
```

3. 其他步骤不变。

```
package com.study.activemq.le3_protocol.amqp;
```

```
import javax.jms.Connection;
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.qpid.jms.JmsConnectionFactory;

/**
 * 简单生产者
 */
public class Producer {
    public static void main(String[] args) {
        new ProducerThread("amqp://mq.study.com:5672", "queue1").start();
    }

    static class ProducerThread extends Thread {
        String brokerUrl;
        String destinationUrl;

        public ProducerThread(String brokerUrl, String destinationUrl) {
            this.brokerUrl = brokerUrl;
            this.destinationUrl = destinationUrl;
        }

        @Override
        public void run() {
            JmsConnectionFactory connectionFactory;
            Connection conn;
            Session session;

            try {
                // 1、创建连接工厂
                connectionFactory = new JmsConnectionFactory(null, null,
brokerUrl);
                // 2、创建连接
                conn = connectionFactory.createConnection();
                conn.start(); // 一定要start

                // 3、创建会话（可以创建一个或者多个session）
                session = conn.createSession(false,
Session.AUTO_ACKNOWLEDGE);

                // 4、创建消息发送目标 (Topic or Queue)
                Destination destination =
session.createQueue(destinationUrl);

                // 5、用目的地创建消息生产者
                MessageProducer producer =
session.createProducer(destination);
                // 设置递送模式(持久化 / 不持久化)
                producer.setDeliveryMode(DeliveryMode.PERSISTENT);
                producer.setPriority(7);

                // 6、创建一条文本消息
```

```
                String text = "Hello world! From: " +
Thread.currentThread().getName() + " : "
                        + System.currentTimeMillis();
                TextMessage message = session.createTextMessage(text);

                // 7、通过producer 发送消息
                System.out.println("Sent message: " + text);
                producer.send(message);

                // 8、 清理、关闭连接
                session.close();
                conn.close();
            } catch (JMSException e) {
                e.printStackTrace();
            }
        }
    }
}
```

6.5 MQTT

MQTT（Message Queuing Telemetry Transport）消息队列遥测传输是IBM开发的一个即时通讯协议，已成为物联网通信的标准。

服务端配置说明：http://activemq.apache.org/mqtt

客户端使用：

1. 引入mqtt客户端jar

```
        <dependency>
            <groupId>org.fusesource.mqtt-client</groupId>
            <artifactId>mqtt-client</artifactId>
            <version>1.15</version>
        </dependency>
```

2. 编写客户端代码

   publisher:

```
/**
 * 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
 * <p>
 * http://www.apache.org/licenses/LICENSE-2.0
 * <p>
 * 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.
 */
package com.study.activemq.le3_protocol.mqtt;

import org.fusesource.hawtbuf.AsciiBuffer;
import org.fusesource.hawtbuf.Buffer;
import org.fusesource.hawtbuf.UTF8Buffer;
import org.fusesource.mqtt.client.Future;
import org.fusesource.mqtt.client.FutureConnection;
import org.fusesource.mqtt.client.MQTT;
import org.fusesource.mqtt.client.QoS;

/**
 * Uses a Future based API to MQTT.
 */
class Publisher {

    public static void main(String[] args) throws Exception {

        MQTT mqtt = new MQTT();
        mqtt.setHost("localhost", 1883);
        // mqtt.setUserName(user);
        // mqtt.setPassword(password);

        FutureConnection connection = mqtt.futureConnection();
        connection.connect().await();

        UTF8Buffer topic = new UTF8Buffer("foo/blah/bar");
        Buffer msg = new AsciiBuffer("mqtt message");

        Future<?> f = connection.publish(topic, msg, QoS.AT_LEAST_ONCE,
false);
        f.await();

        connection.disconnect().await();

        System.exit(0);
    }

}
```

Listener

```
/**
 * 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.
 */
package com.study.activemq.le3_protocol.mqtt;

import org.fusesource.hawtbuf.Buffer;
import org.fusesource.hawtbuf.UTF8Buffer;
import org.fusesource.mqtt.client.Callback;
import org.fusesource.mqtt.client.CallbackConnection;
import org.fusesource.mqtt.client.MQTT;
import org.fusesource.mqtt.client.QoS;
import org.fusesource.mqtt.client.Topic;

/**
 * Uses an callback based interface to MQTT. Callback based interfaces are
 * harder to use but are slightly more efficient.
 */
class Listener {

    public static void main(String[] args) throws Exception {

        MQTT mqtt = new MQTT();
        mqtt.setHost("localhost", 1883);
        // mqtt.setUserName(user);
        // mqtt.setPassword(password);

        final CallbackConnection connection = mqtt.callbackConnection();
        connection.listener(new org.fusesource.mqtt.client.Listener() {
            long count = 0;
            long start = System.currentTimeMillis();

            public void onConnected() {
            }

            public void onDisconnected() {
            }

            public void onFailure(Throwable value) {
                value.printStackTrace();
                System.exit(-2);
            }

            public void onPublish(UTF8Buffer topic, Buffer msg, Runnable
ack) {
                String body = msg.utf8().toString();
                System.out.println("收到消息: " + body);
                ack.run();
            }
        });
        connection.connect(new Callback<Void>() {
            @Override
            public void onSuccess(Void value) {
                Topic[] topics = { new Topic("foo/blah/bar",
QoS.AT_LEAST_ONCE) };
                connection.subscribe(topics, new Callback<byte[]>() {
                    public void onSuccess(byte[] qoses) {
                    }
```

```
                    public void onFailure(Throwable value) {
                        value.printStackTrace();
                        System.exit(-2);
                    }
                });
            }

            @Override
            public void onFailure(Throwable value) {
                value.printStackTrace();
                System.exit(-2);
            }
        });

        // Wait forever..
        synchronized (Listener.class) {
            while (true)
                Listener.class.wait();
        }
    }

}
```

6.6 AUTO

Starting with version 5.13.0, ActiveMQ supports wire format protocol detection. OpenWire, STOMP, AMQP, and MQTT can be automatically detected. This allows one transport to be shared for all 4 types of clients.

从 ActiveMQ 5.13.0开始，ActiveMQ 开始支持协议格式检测，可以自动检测OpenWire、STOMP、AMQP和MQTT。允许这4种类型的客户端共享一个传输。

官网说明：http://activemq.apache.org/auto

7 安全

ActiveMQ默认并未开启安全访问控制。支持对Queue、Topic的认证、鉴权。

https://activemq.apache.org/security

认证

简单认证方式

在activemq.xml的broker中配置简单认证的插件及用户。

```
        <plugins>
            <!-- Configure authentication; Username, passwords and groups -->
            <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>
```

```
ConnectionFactory connectionFactory = new
ActiveMQConnectionFactory(username,password,url);
```

也可以支持匿名访问

```
<simpleAuthenticationPlugin anonymousAccessAllowed="true">.....
```

JAAS方式

1、在activemq.xml的broker中配置JAAS插件。

```
        <plugins>
            <!--use JAAS to authenticate using the login.config file on the
classpath to configure JAAS -->
            <!--
            <jaasAuthenticationPlugin configuration="activemq" />
        </plugins>
```

configuration="activemq" 指定使用 login.config中的"activemq"配置。

conf/login.config

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

2、在conf/users.properties中配置用户

```
#用户名=密码
admin=admin
user=password
guest=guest
```

3、在conf/groups.properties中配置用户组（角色）

```
#组名=用户1,用户2      组名自定义
admins=admin
users=admin,user
guests=guest
```

权限控制

控制用户组对Queue/Topic的操作权限。

权限有：

Operation	Description
read	You can browse and consume from the destination
write	You can send messages to the destination
admin	You can lazily create the destination if it does not yet exist. This allows you fine grained control over which new destinations can be dynamically created in what part of the queue/topic hierarchy

```
<broker>
  ..
    <plugins>
      ..
      <authorizationPlugin>
        <map>
          <authorizationMap>
              <!-- 对Queue/Topic进行用户组授权 -->
            <authorizationEntries>
              <authorizationEntry queue="TEST.Q" read="users" write="users"
admin="users" />
              <authorizationEntry topic="ActiveMQ.Advisory.>" read="all"
write="all" admin="all"/>
            </authorizationEntries>
              <!-- 对临时目标进行访问控制，这个控制是对所有临时目标 -->
            <tempDestinationAuthorizationEntry>
              <tempDestinationAuthorizationEntry read="admin" write="admin"
admin="admin"/>
            </tempDestinationAuthorizationEntry>
          </authorizationMap>
        </map>
      </authorizationPlugin>
```

```
      ..
    </plugins>
  ..
</broker>
```

临时目标说明

生命周期与创建它的连接的生命周期关联目标：

- 通过Session的createTemporaryQueue，CreateTemporaryTopic创建临时目标；
- 临时目标的生命周期是创建它的Connection关联，如果关闭了创建此临时目标的Connection，那么临时目标被关闭，内容也将消失
- 只有创建它的Connection所创建的session才能从临时目标中接收消息；不过任何的生产者都可以向临时目标中发送消息；