

SPRING JMS ActiveMQ

L'objectif de cet article est de vous permettre d'appréhender les bases de la mise en place de JMS avec SPRING et activeMQ.

Le projet de test que vous allez réaliser est téléchargeable à la fin de cet article, ce projet un projet maven donc pour le build : mvn package .Maven s'occupera de télécharger les dépendances pour vous .

Spring est le meta-framework du moment, il intègre les concepts comme **AOP IOC MVC** et fourni un nombre considérable d'helpers pour l'intégrations de technologies, nous l'utiliserons afin de bénéficier des templates jms qu'il fournis.

Java Message Service (JMS) permet d'envoyer et de recevoir des messages de manière asynchrone entre applications ou composants Java. Un client peut également recevoir des messages de façon synchrone dans le mode de communication point à point.

ActiveMQ d'Apache est le plus populaire et puissant des Message Broker open source, il permet une intégration aisé et multi langages.

1. Installation activeMQ

tout d'abord vous devez télécharger activeMQ [ici](#) , une fois cela fait vous le décompacter

tar zxvf apache-activemq-5.0.0.tar.gz

puis vous vous positionner dans le répertoire correspondant

cd apache-activemq-5.0.0/bin

et enfin vous lancer le script d'execution

activemq > activemq-log.txt

si vous vérifier les logs vous devriez voir le résultat suivant

Loading message broker from: xbean:activemq.xml

INFO BrokerService - Using Persistence Adapter: AMQPersistenceAdapter(/home/ubuntu/apache-activemq-5.0.0/data/localhost)

INFO BrokerService - ActiveMQ 5.0.0 JMS Message Broker (localhost) is starting

INFO BrokerService - For help or more information please see: <http://activemq.apache.org/>

INFO AMQPersistenceAdapter - AMQStore starting using directory: /home/ubuntu/apache-activemq-5.0.0/data/localhost

INFO KahaStore - Kaha Store using data directory /home/ubuntu/apache-activemq-5.0.0/data/localhost/kr-store/state

INFO ManagementContext - JMX consoles can connect to service:jmx:rmi:///jndi/rmi://localhost:1099/jmxrmi

INFO AMQPersistenceAdapter - Active data files: []

Tutoriels & co

```
INFO KahaStore          - Kaha Store using data directory /home/ubuntu/apache-activemq-5.0.0/data/localhost/kr-
store/data
INFO TransportServerThreadSupport - Listening for connections at: tcp://ubuntu-desktop:61616
INFO TransportConnector      - Connector openwire Started
INFO TransportServerThreadSupport - Listening for connections at: ssl://ubuntu-desktop:61617
INFO TransportConnector      - Connector ssl Started
INFO TransportServerThreadSupport - Listening for connections at: stomp://ubuntu-desktop:61613
INFO TransportConnector      - Connector stomp Started
INFO TransportServerThreadSupport - Listening for connections at: xmpp://ubuntu-desktop:61222
INFO TransportConnector      - Connector xmpp Started
INFO NetworkConnector        - Network Connector
org.apache.activemq.transport.discovery.multicast.MulticastDiscoveryAgent@1c21ece Started
INFO BrokerService          - ActiveMQ JMS Message Broker (localhost, ID:ubuntu-
desktop-48019-1200088019292-0:0) started
INFO TransportConnector      - Connector vm://localhost Started
INFO log                    - Logging to org.slf4j.impl.JCLLoggerAdapter(org.mortbay.log) via
org.mortbay.log.Slf4jLog
INFO log                    - jetty-6.1.4
INFO WebConsoleStarter      - ActiveMQ WebConsole initialized.
INFO /admin                  - Initializing Spring FrameworkServlet 'dispatcher'
INFO log                    - ActiveMQ Console at http://0.0.0.0:8161/admin
INFO log                    - ActiveMQ Web Demos at http://0.0.0.0:8161/demo
```

a ce stade activeMQ est prêt à être utilisé. Vous pouvez vous connecter à la console pour vérifier à l'adresse <http://0.0.0.0:8161/admin/>

activeMQ est désormais accessible sur le port 61616 pour vos applications java.

2. configuration du projet maven

Création de votre pom.xml avec les dépendances nécessaires :

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
  <name>TutorielActiveMQSPRING</name>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.karlverger.tutoriel</groupId>
  <artifactId>activemq</artifactId>
  <packaging>war</packaging>
  <version>1.0.0-SNAPSHOT</version>

  <repositories>
    <repository>
      <id>people.apache.org</id>
      <name>Apache ActiveMQ SNAPSHOT</name>
      <url>http://people.apache.org/repo/m2-snapshot-repository/</url>
      <layout>default</layout>
    </repository>
```

Tutoriels & co

```
</repositories>
```

```
<build>
```

```
  <plugins>
```

```
    <plugin>
```

```
      <artifactId>maven-compiler-plugin</artifactId>
```

```
      <configuration>
```

```
        <source>1.5</source>
```

```
        <target>1.5</target>
```

```
      </configuration>
```

```
    </plugin>
```

```
  </plugins>
```

```
</build>
```

```
<dependencies>
```

```
  <dependency>
```

```
    <groupId>commons-dbcp</groupId>
```

```
    <artifactId>commons-dbcp</artifactId>
```

```
    <version>1.2.1</version>
```

```
    <scope>compile</scope>
```

```
  </dependency>
```

```
  <dependency>
```

```
    <groupId>javax.transaction</groupId>
```

```
    <artifactId>jta</artifactId>
```

```
    <version>1.0.1B</version>
```

```
  </dependency>
```

```
  <dependency>
```

```
    <groupId>org.apache.activemq</groupId>
```

```
    <artifactId>activemq-core</artifactId>
```

```
    <version>5.0.0</version>
```

```
  </dependency>
```

```
  <dependency>
```

```
    <groupId>org.springframework</groupId>
```

```
    <artifactId>spring</artifactId>
```

```
    <version>2.0.5</version>
```

```
    <scope>compile</scope>
```

```
  </dependency>
```

```
  <dependency>
```

```
    <groupId>org.springframework</groupId>
```

```
    <artifactId>spring-jdbc</artifactId>
```

```
    <version>2.5</version>
```

```
    <scope>compile</scope>
```

```
  </dependency>
```

Tutoriels & co

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-mock</artifactId>
  <version>2.0.5</version>
  <scope>compile</scope>
</dependency>
```

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-remoting</artifactId>
  <version>2.0.5</version>
  <scope>compile</scope>
</dependency>
```

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-support</artifactId>
  <version>2.0.5</version>
  <scope>compile</scope>
</dependency>
```

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.1</version>
</dependency>
```

```
</dependencies>
```

```
</project>
```

3. Le fichier de configuration SPRING applicationContext-services.xml

maintenant nous allons déclarer les beans nécessaires à notre projet, le fichier applicationContext-services.xml est auto documenté

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
  xmlns:lang="http://www.springframework.org/schema/lang"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-
beans-2.0.xsd
http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.0.xsd
http://www.springframework.org/schema/lang http://www.springframework.org/schema/lang/spring-lang-2.0.xsd">
<!-- pool de connection JMS -->
```

Tutoriels & co

```
<bean id="jmsFactory" class="org.apache.activemq.pool.PooledConnectionFactory" destroy-method="stop">
  <property name="connectionFactory">
    <bean class="org.apache.activemq.ActiveMQConnectionFactory">
      <property name="brokerURL">
        <value>tcp://localhost:61616</value>
      </property>
    </bean>
  </property>
</bean>
<!--
template JMS Spring simplifiant la gestion des message JMS
-->
<bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate">
  <property name="connectionFactory">
    <ref local="jmsFactory"/>
  </property>
</bean>
<!--
Définition d'une queue permettant d'échanger les message JMS
-->
<bean id="destination" class="org.apache.activemq.command.ActiveMQQueue">
  <constructor-arg index="0">
    <value>input.queue</value>
  </constructor-arg>
</bean>
<!--
mise en place d'un listener qui consommera les message
de manière asynchrones sur la destination déclarer au-dessus
-->
<bean id="myAsynchJmsListener" class="com.karlverger.tutoriel.activemq.MyAsynchJmsListener"/>
<bean id="listenerContainer" class="org.springframework.jms.listener.DefaultMessageListenerContainer">
  <property name="concurrentConsumers" value="5"/>
  <property name="connectionFactory" ref="jmsFactory" />
  <property name="destination" ref="destination" />
  <property name="messageListener" ref="myAsynchJmsListener" />
</bean>
<!--
création d'un bean spring qui utilise le template JMS et la destination
pour envoyer les message
-->
<bean id="springProducer" class="com.karlverger.tutoriel.activemq.SpringProducer">
  <property name="template" ref="jmsTemplate"/>
  <property name="destination" ref="destination"/>
</bean>
</beans>
```

4. developpement des beans et réalisation des test

Nous allons créer le bean SpringProducer, cette classe s'apuié sur le JmsTemplate de spring pour créer et expédier les message JMS, les propriétés sont injecté par spring.

```
package com.karlverger.tutoriel.activemq;

import javax.jms.Destination;
import javax.jms.JMSEException;
import javax.jms.Message;
import javax.jms.Session;
import javax.jms.TextMessage;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.jms.core.MessageCreator;

/**
 * ce bean montre un exemple simple d'utilisation
 * @author karl verger
 */
public class SpringProducer {

    private JmsTemplate template;
    private Integer messageCount;
    private Destination destination;

    public void sendMessage(){
        MessageCreator messageCreator = new MessageCreator() {
            public Message createMessage(Session session) throws JMSEException {
                TextMessage textMessage = session.createTextMessage();
                textMessage.setText( "Mon message");
                textMessage.setJMSType("Mon type");
                return textMessage;
            }
        };
        getTemplate().send(getDestination(), messageCreator);
        System.err.println("Le message à été envoyé");
    }

    public JmsTemplate getTemplate() {
        return template;
    }

    public void setTemplate(JmsTemplate template) {
        this.template = template;
    }
}
```

Tutoriels & co

```
public Destination getDestination() {
    return destination;
}

public void setDestination(Destination destination) {
    this.destination = destination;
}

public Integer getMessageCount() {
    return messageCount;
}

public void setMessageCount(Integer messageCount) {
    this.messageCount = messageCount;
}
}
```

Maintenant nous allons créer le bean `MyAsynchJmsListener`, cette classe s'occupe de consommer les messages de manière asynchrone.

```
package com.karlverger.tutoriel.activemq;

import java.util.logging.Level;
import java.util.logging.Logger;
import javax.jms.JMSEException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;

/**
 * ce bean montre un exemple simple de consommation asynchrone
 * de message
 * @author karl verger
 */
public class MyAsynchJmsListener implements MessageListener {

    /**
     * cette méthode est appelé lors de la réception d'un message
     * sur la queue défini dans applicationContext-services.xml
     * @param message
     */
    public void onMessage(Message message) {
        try {
            System.err.println("Instance du message : "+message.toString());
            System.err.println("Contenu du message : "+ ((TextMessage)message).getText() );
            System.err.println("Type de message : " + message.getJMSType());
        }
    }
}
```

Tutoriels & co

```
    } catch (JMSEException ex) {
        Logger.getLogger(MyAsynchJmsListener.class.getName()).
            log(Level.SEVERE, null, ex);
    }
}
}
```

Maintenant nous pouvons réaliser notre classe de test celle ci ne se contente que d'exécuter la méthode `sendMessage` de notre bean `SpringProducer`, mais vous devriez voir l'expédition et la consommation du message car le listener JMS aura été instancié par spring et se déclenchera à l'arrivée du message envoyé par notre `SpringProducer`.

```
package com.karlverger.tutoriel.activemq;

import org.springframework.test.AbstractTransactionalDataSourceSpringContextTests;
import static org.junit.Assert.*;

/**
 *
 * @author karl verger
 */
public class SpringProducerTest extends AbstractTransactionalDataSourceSpringContextTests {

    protected SpringProducer springProducer;

    public SpringProducerTest() {
        setPopulateProtectedVariables(true);
    }

    @Override
    protected String[] getConfigLocations() {
        return new String[] {"applicationContext-services.xml"};
    }

    public void testSendMessage(){
        getSpringProducer().sendMessage();
    }

    public SpringProducer getSpringProducer() {
        return springProducer;
    }

    public void setSpringProducer(SpringProducer springProducer) {
        this.springProducer = springProducer;
    }
}
```

Tutoriels & co

}