Ecole Normale Supérieure de l'Enseignement Technique Mohammedia
Université Hassan II de Casablanca

ENSET

المدرسة العليا لأساتذة التعليم التقني المحمدية
جامعة الحسن الثاني بالدار البيضاء

Département Mathématiques et Informatique

Filière :

« Ingénierie Informatique : Big Data et Cloud Computing »

II-BDCC

Travail Pratique : Application de chat avec JMS ActiveMQ

Réalisé par :

GUEDDI Hamza

Année universitaire : 2020-2021

Table des matières

Enoncé 3
Introduction 3
Architecture de JMS 5
Code 5
Captures d'écran 12
Conclusion 14

Enoncé

On souhaite créer une application de chat basée sur JMS qui permet aux différents clients de :

- Se connecter au Provider JMS en utilisant un code.
- Envoyer et recevoir des messages de type Texte.
- Envoyer et recevoir des messages contenant des images.

L'interface graphique de l'application est basée sur JavaFX.

Introduction

JMS a été intégré à la plate-forme J2EE à partir de la version 1.3. Il n'existe pas d'implémentation officielle de cette API avant cette version. JMS est utilisable avec les versions antérieures mais elle oblige à utiliser un outil externe qui implémente l'API.

Chaque fournisseur (provider) doit fournir une implémentation de ses spécifications. Il existe un certain nombre d'outils qui implémentent JMS dont la majorité sont des produits commerciaux.

Dans la version 1.3 du J2EE, JMS peut être utilisé dans un composant web ou un EJB, un type d'EJB particulier a été ajouté pour traiter les messages et des échanges JMS peuvent être intégrés dans une transaction gérée avec JTA (Java Transaction API).

JMS définit plusieurs entités :

Un provider JMS : outil qui implémente l'API JMS pour échanger les messages : ce sont les brokers de messages

Un client JMS : composant écrit en java qui utilise JMS pour émettre et/ou recevoir des messages.

Un message : données échangées entre les composants

Différents objets utilisés avec JMS sont généralement stockés dans l'annuaire JNDI du serveur d'applications ou du provider du MOM :

La fabrique de connexions (ConnectionFactory)

Les destinations à utiliser (Queue et Topic)

JMS définit deux modes pour la diffusion des messages :

Point à point (Point to point) : dans ce mode un message est envoyé par un producteur et est reçu par un unique consommateur. Le support utilisé pour la mise en oeuvre de ce mode est la file (queue). Le message émis est stocké dans la file jusqu'à ce que le consommateur le lise et envoie une notification de réception du message. A ce moment là le message est supprimé de la file. Le message a généralement une date d'expiration.

Publication / souscription (publish/subscribe) : dans ce mode un message est envoyé par un producteur et est reçu par un ou plusieurs consommateurs. Le support utilisé pour la mise en oeuvre de ce mode est le sujet (topic). Chaque consommateur doit s'abonner à un sujet (souscription). Seuls les messages émis à partir de cet abonnement sont accessibles par le consommateur.

Dans la version 1.0 de JMS, ces modes utilisent des interfaces distinctes.

Dans la version 1.1 de JMS, ces interfaces sont toujours utilisables mais il est aussi possible d'utiliser des interfaces communes à ces modes ce qui les rend interchangeables.

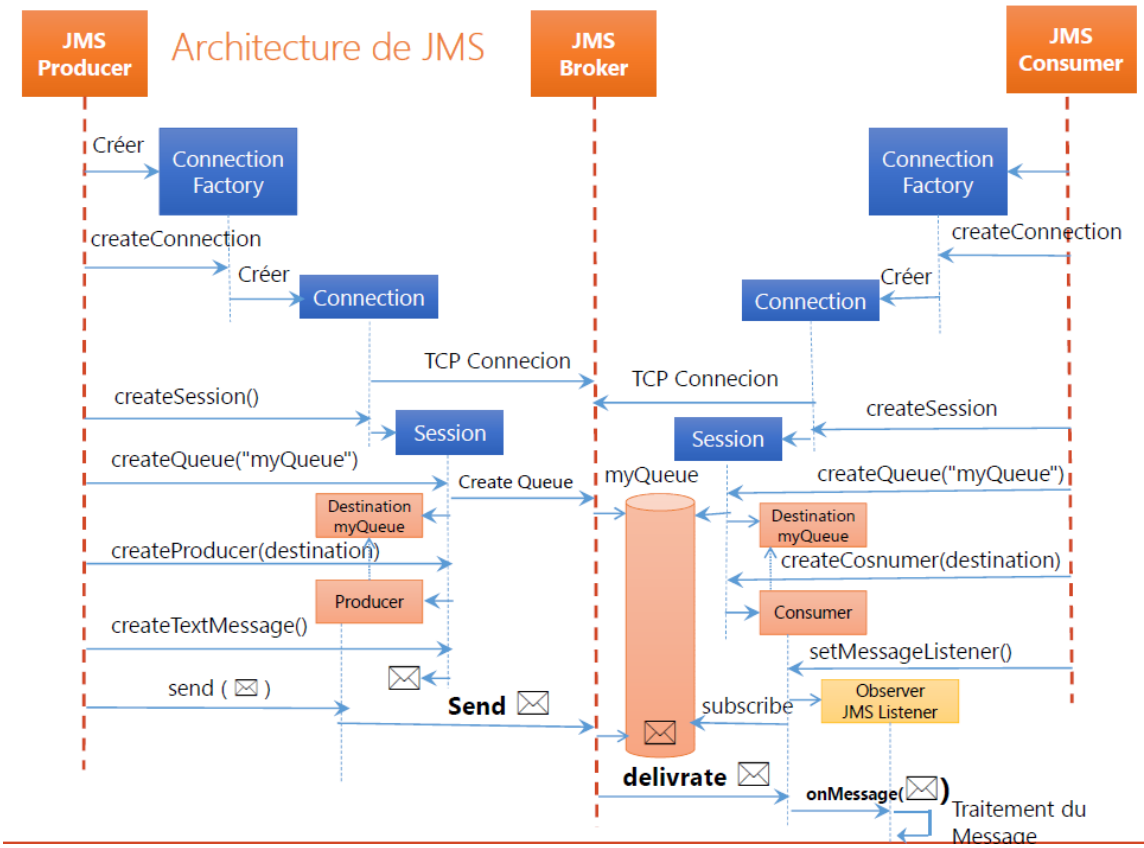
Les messages sont asynchrones mais JMS définit deux modes pour consommer un message :

Mode synchrone : ce mode nécessite l'appel de la méthode receive() ou d'une de ses surcharges. Dans ce cas, l'application est arrêtée jusqu'à l'arrivée du message. Une version surchargée de cette méthode permet de rendre la main après un certain timeout.

Mode asynchrone : il faut définir un listener qui va lancer un thread attendant les messages et exécutant une méthode à leur arrivée.

JMS propose un support pour différents types de messages : texte brut, flux d'octets, objets Java sérialisés...

Architecture de JMS



Code

JMSChat.java (Classe principale)

```
import javafx.application.Application;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.*;
import javafx.scene.paint.Color;
```

```
import javafx.stage.Stage;
import org.apache.activemq.ActiveMQConnectionFactory;

import javax.jms.*;
import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.FileInputStream;

public class JMSChat extends Application {

    private MessageProducer messageProducer;
    private Session session;
    private String codeUser;
    //private Session session2;

    public static void main(String[] args) {
        Application.launch(JMSChat.class);
    }
    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("JMS Chat App");
        BorderPane borderPane = new BorderPane();

        HBox hBox = new HBox();
        hBox.setPadding(new Insets(10));
        hBox.setSpacing(10);
        hBox.setBackground(new Background(new
BackgroundFill(Color.CADETBLUE , CornerRadii.EMPTY
,Insets.EMPTY)));

        Label labelCode = new Label("Code:");
        labelCode.setPadding(new Insets(5));
        TextField textFieldCode = new TextField("C1");
        textFieldCode.setPromptText("Code");

        Label labelHost = new Label("Host:");
        labelHost.setPadding(new Insets(5));
        TextField textFieldHost = new TextField("localhost");
        textFieldHost.setPromptText("Host");

        Label labelPort = new Label("Port:");
        labelPort.setPadding(new Insets(5));
        TextField textFieldPort = new TextField("61616");
        textFieldPort.setPromptText("Port");

        Button connectButton = new Button("Connect");

        hBox.getChildren().add(labelCode);
        hBox.getChildren().add(textFieldCode);
```

```
        hBox.getChildren().add(labelHost);
        hBox.getChildren().add(textFieldHost);
        hBox.getChildren().add(labelPort);
        hBox.getChildren().add(textFieldPort);
        hBox.getChildren().add(connectButton);

        borderPane.setTop(hBox);

        VBox vBox = new VBox();
        GridPane gridPane = new GridPane();
        HBox hBox2 = new HBox();
        vBox.getChildren().add(gridPane);
        vBox.getChildren().add(hBox2);
        borderPane.setCenter(vBox);

        Label labelTo = new Label("To:");
        TextField textFieldTo = new TextField("C1");
        textFieldTo.setPrefWidth(250);
        Label labelMessage = new Label("Message:");
        TextArea textAreaMessage = new TextArea();
        textAreaMessage.setPrefWidth(250);
        Button sendButton = new Button("Send");
        Label labelImage = new Label("Image");

        File images = new File("images");
        ObservableList<String> observableListImages =

FXCollections.observableArrayList(images.list());


        ComboBox<String> comboBoxImages = new
ComboBox<String>(observableListImages);
        comboBoxImages.getSelectionModel().select(0);
        Button sendImageButton = new Button("Send Image");

        gridPane.setPadding(new Insets(10));
        gridPane.setVgap(10);
        gridPane.setHgap(10);
        textAreaMessage.setPrefRowCount(1);

        gridPane.add(labelTo , 0 , 0);
        gridPane.add(textFieldTo,1,0);
        gridPane.add(labelMessage,0,1);
        gridPane.add(textAreaMessage,1,1);
        gridPane.add(sendButton,2,1);
        gridPane.add(labelImage,0,2);
        gridPane.add(comboBoxImages,1,2);
        gridPane.add(sendImageButton,2,2);
```

```
        ObservableList<String> observableListMessages =
                FXCollections.observableArrayList();
        ListView<String> listViewMessages = new
ListView<>(observableListMessages);

        File images2 = new
File("images/"+comboBoxImages.getSelectionModel().getSelectedI
tem());
        Image image = new Image(images2.toURI().toString());
        ImageView imageView = new ImageView(image);
        imageView.setFitWidth(480);
        imageView.setFitHeight(360);

        hBox2.getChildren().add(listViewMessages);
        hBox2.getChildren().add(imageView);
        hBox2.setPadding(new Insets(10));
        hBox2.setSpacing(10);

        Scene scene = new Scene(borderPane , 1024 , 768);
        primaryStage.setScene(scene);
        primaryStage.show();


comboBoxImages.getSelectionModel().selectedItemProperty().addL
istener(new ChangeListener<String>() {
            @Override
            public void changed(ObservableValue<? extends
String> observable, String oldValue, String newValue) {
                File images3 = new File("images/"+newValue);
                Image image = new
Image(images3.toURI().toString());
                imageView.setImage(image);

            }
        });

        sendButton.setOnAction(event -> {
            try {
                TextMessage textMessage =
session.createTextMessage();

textMessage.setText(textAreaMessage.getText());
                textMessage.setStringProperty("code"
,textFieldTo.getText());
                messageProducer.send(textMessage);
            } catch (JMSException e) {
                e.printStackTrace();
            }
```

```
        });

        sendImageButton.setOnAction(event -> {
            try {
                StreamMessage streamMessage =
session.createStreamMessage();

streamMessage.setStringProperty("code",textFieldTo.getText());
                File images4 = new
File("images/"+comboBoxImages.getSelectionModel().getSelectedI
tem());
                FileInputStream fileInputStream = new
FileInputStream(images4);
                byte[] data = new byte[(int)
images4.length()];
                fileInputStream.read(data);

streamMessage.writeString(comboBoxImages.getSelectionModel().g
etSelectedItem());
                streamMessage.writeInt(data.length);
                streamMessage.writeBytes(data);
                messageProducer.send(streamMessage);


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


        });


        connectButton.setOnAction(new
EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                try {
                    codeUser = textFieldCode.getText();
                    String host = textFieldHost.getText();
                    int port =
Integer.parseInt(textFieldPort.getText());
                    ConnectionFactory connectionFactory =
                            new
ActiveMQConnectionFactory("tcp://"+host+":"+port);
                    Connection connection =
connectionFactory.createConnection();
                    connection.start();
                    session = connection.createSession(false ,
Session.AUTO_ACKNOWLEDGE);
```

```
                    Destination destination =
session.createTopic("bdcc.chat");
                    MessageConsumer messageConsumer =
session.createConsumer(destination , "code='"+codeUser+"'");

                    messageProducer =
session.createProducer(destination);

messageProducer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);


                    //start of function
                    messageConsumer.setMessageListener(new
MessageListener() {
                        @Override
                        public void onMessage(Message message)
{
                            try {
                                if (message instanceof
TextMessage) {
                                    TextMessage textMessage =
(TextMessage) message;

observableListMessages.add(textMessage.getText());
                                } else if (message instanceof
StreamMessage) {
                                    StreamMessage
streamMessage = (StreamMessage) message;
                                    String pictureName =
streamMessage.readString();

observableListMessages.add("Picture received : "+pictureName);
                                    int size =
streamMessage.readInt();
                                    byte[] data = new
byte[size];

streamMessage.readBytes(data);
                                    ByteArrayInputStream
byteArrayInputStream = new ByteArrayInputStream(data);
                                    Image image = new
Image(byteArrayInputStream);
                                    imageView.setImage(image);

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

```
                    }
                });

                //end of function
                hBox.setDisable(true);

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

Classe ActiveMQBroker (Classe optionnelle pour démarrer ActiveMQ)

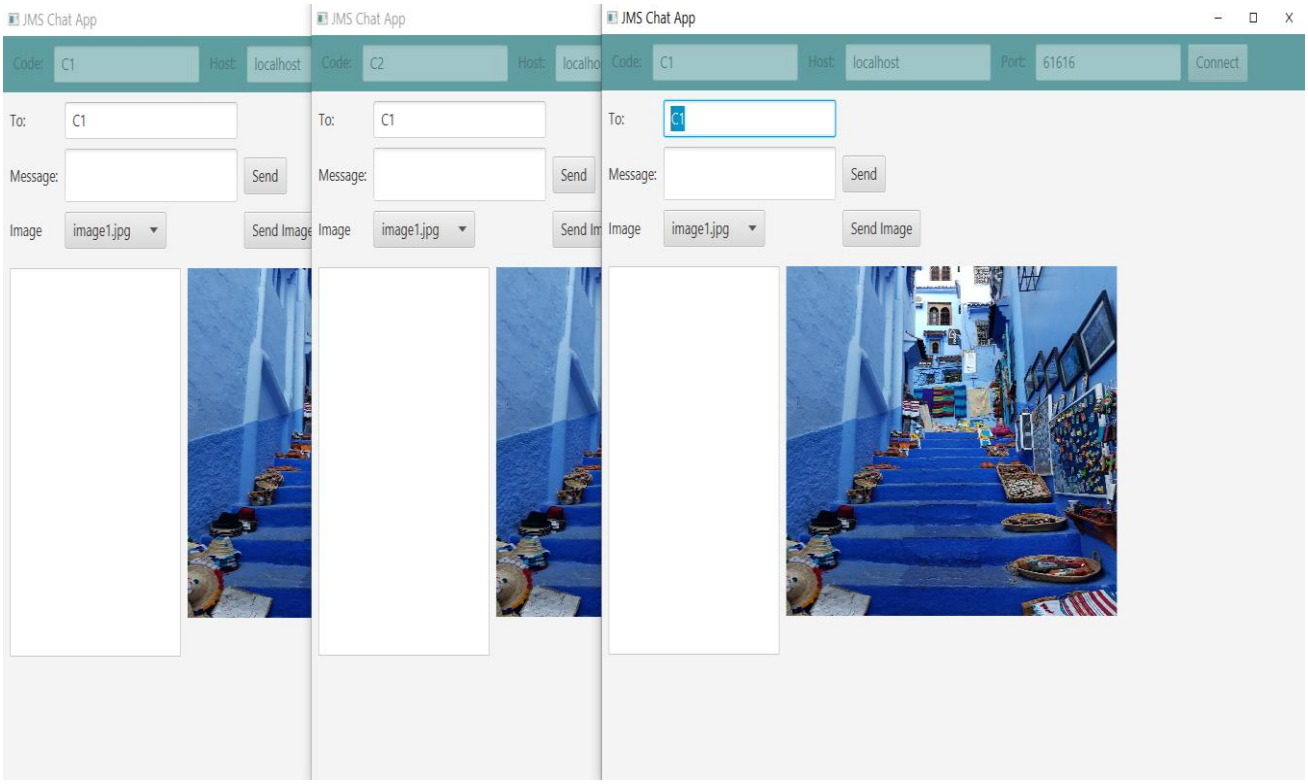
```
import org.apache.activemq.broker.BrokerService;

public class ActiveMQBroker {

    public static void main(String[] args) {
        try {
            BrokerService brokerService = new BrokerService();
            brokerService.setPersistent(false);
            brokerService.addConnector("tcp://0.0.0.0:61616");
            brokerService.start();
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
}
```

Captures d’écran

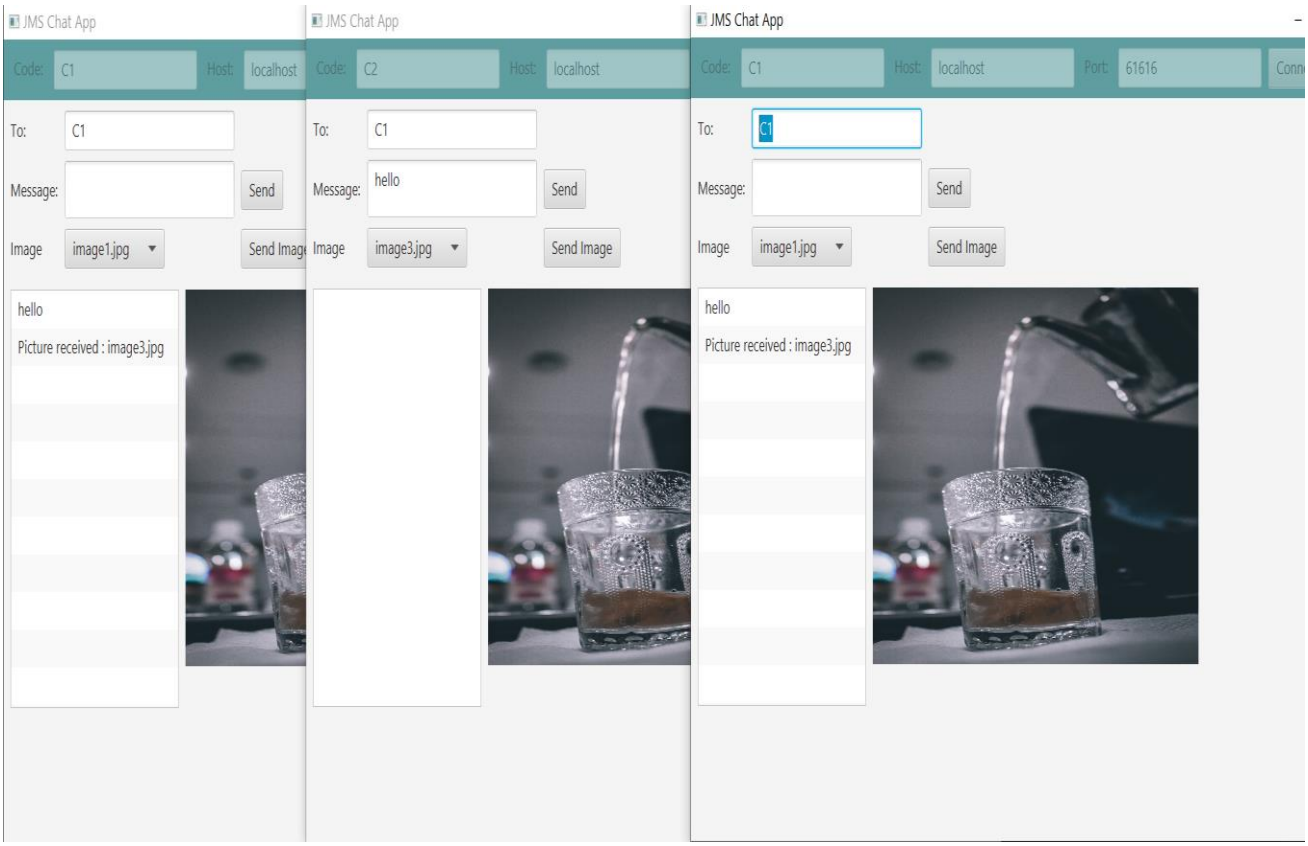
Dans ces captures d’écran , on ne va pas utiliser un embedded ActiveMQ , mais on va le démarrer à partir du ligne de commandes.

```
C:\apache-activemq-5.15.9\bin>activemq start
```

On démarre maintenant l’application et on lance 3 instances pour créer 3 consumers et on les connecte.



Maintenant on va envoyer un message texte « hello ! » et l’image 3 de la part de C2 vers C1



On voit donc que le message et l'image ont été envoyés avec succès.
On accède maintenant au GUI d'administration du serveur.

Topics

Name ↑	Number Of Consumers	Messages Enqueued	Messages Dequeued	Operations
ActiveMQ.Advisory.Connection	0	45	0	Send To Active Subscribers Active Producers Delete
ActiveMQ.Advisory.Consumer.Topic.bdcc.chat	0	45	0	Send To Active Subscribers Active Producers Delete
ActiveMQ.Advisory.MasterBroker	0	1	0	Send To Active Subscribers Active Producers Delete
ActiveMQ.Advisory.Producer.Topic.bdcc.chat	0	45	0	Send To Active Subscribers Active Producers Delete
ActiveMQ.Advisory.Queue	0	1	0	Send To Active Subscribers Active Producers Delete
ActiveMQ.Advisory.Topic	0	2	0	Send To Active Subscribers Active Producers Delete
bdcc.chat	3	20	18	Send To Active Subscribers Active Producers Delete
enset.topic	0	0	0	Send To Active Subscribers Active Producers Delete

Donc d'après la liste des topics, on voit qu'on a un topic « bdcc.chat » et il existe 3 consumers connecté , et le nombre des messages crées par le producer , et ceux reçus par les consumers.

Conclusion

Ce TP a été très utile car j'ai pu apprendre le modèle de systèmes distribués asynchrones, les brokers qui existent, et comment créer une application basée sur JMS pour l'envoi des messages que ce soit du texte ou des images.