Java EE 6
JMS

CREATION : 2010/06/27
MISE A JOUR : 2010/06/27
ENVIRONNEMENT : Glassfish 3.0.1 et Netbeans 3.9

Le but de ce TP est de mettre en œuvre l'API JMS (Java Messaging Service). Les clients peuvent communiquer grâce à JNDI. Dans ce TP, nous allons plutôt nous servir de Glassfish et de l'injection de dépendance dans l'ACC (le conteneur d'application client).

L'application se décompose en deux volets : le producteur (ou éditeur) et le consommateur (les abonnés).

L'éditeur

- Créer un projet « Entreprise Application Client » que l'on nommera Journal (ou Twitter ou FluxRSS ;-))
- Doter la classe des paramètres de la connexion

```
@Resource(mappedName = "journalFactory")
private static ConnectionFactory connectionFactory;
@Resource(mappedName = "journalTopic")
private static Topic topic;
```

- Ecrire la méthode main()

```
Connection connection = connectionFactory.createConnection();
Session session = connection.createSession(false,
                  Session.AUTO_ACKNOWLEDGE);
MessageProducer producer = session.createProducer(topic);

TextMessage message = session.createTextMessage();
message.setText("Première publication");
producer.send(message);
connection.close();
```

- Il manque la gestion des exceptions.
- Netbeans peut résoudre tous les imports automatiquement, attention toutefois que les packages proposés soient les bons.
- Nous reviendrons sur ce code une fois qu'il aura été testé.

L'abonné

On peut écrire la classe abonné de la même manière que la classe précédente si une communication synchrone nous intéresse. Nous allons plutôt explorer le modèle asynchrone avec l'utilisation d'un Listener.

- Créer nouveau projet de type « Enterprise Application Client ». Le nommer « Abonne »
- Doter la classe des attributs *connectionFactory* et *topic* non statiques.
- Ecrire la méthode Main()

```
Connection connection = connectionFactory.createConnection();
Session session = connection.createSession(false,
                  Session.AUTO_ACKNOWLEDGE);
MessageConsumer consumer = session.createConsumer(topic);

consumer.setMessageListener(new Main());
connection.start();
```

- Ecrire le Listener. Dans le cas présent, la classe est son propre listener. Il faut ajouter l'interface MessageListener à la définition de la classe et implémenter la méthode suivante :

```
@Override
public void onMessage(Message m) {
      System.out.println("Message recu : " + ((TextMessage) m).getText());
}
```

- Résoudre les imports correctement et ajouter la gestion des exceptions.

Configurer les ressources

Pour pouvoir utiliser JMS, il faut configurer correctement toutes les ressources JMS. Avec la console d'administration, il faut développer « Ressources », puis « Ressources JMS »

- Créer la destination JMS. Le nom JNDI doit être unique, je vous propose **journalTopic**. La destination physique est créée automatiquement. Nous laissons le même nom. Bien entendu, le type de ressources est javax.jms.Topic.
- Créer l'« usine » (fabrique) de connexion. Je vous propose de la nommer **journalFactory** de type ConnectionFactory.
- Vérifiez bien que toutes les ressources sont actives.

Lancer l'application

Pour mettre en pratique JMS, il suffit de :

- Définir les ressources JMS au sein de Glassfish soit par la console graphique, soit par asadmin.
- Lancer le ou les programmes « Abonné »

- Lancer le programme « Editeur »

Aller plus loin

Une fois que vous aurez pu vérifier le comportement des deux programmes, je vous propose de modifier les programmes pour en faire quelque chose de plus sympa.

- Modifier le l'application Editeur pour que les textes à envoyer soient saisis sur la console par l'utilisateur. La fin de la session se fera par la saisie de la chaîne « . » et enverra aux clients le message « Fin de connexion ».
- Faire réagir (terminer) les abonnés à la réception du message « Fin de connexion ».

Un abonné ne reçoit les messages que s'il est connecté. Je vous propose de modifier le programme abonné pour que les messages soient gardés, même après déconnexion. Pour cela, il faudra créer un client durable avec un identifiant unique.

- Prendre en compte les paramètres de la ligne de commande : s'il n'y en a pas le client est normal, s'il y a un paramètre, c'est l'identifiant du client durable
- Modifier la connexion pour prendre en compte l'identifiant avec `connection.setClientID()` avant même de créer la session.
- `session.createDurableSubscriber()` permet de créer un abonné durable ….
- Vérifier que tout se passe comme prévu.

Bon test et bonne communication !!!