

La communication par messages : JMS

Contenu de la section

LA COMMUNICATION PAR MESSAGES : JMS.....1

CONTENU DE LA SECTION.....2

QUELQUES GÉNÉRALITÉS (1).....3

les objectifs de JMS (Java Messaging Service).....3

LE MODÈLE JMS.....4

les concepts principaux.....4

L'API JMS (1).....5

les interfaces QueueConnectionFactory et TopicConnectionFactory.....5

les interface QueueConnection et TopicConnection.....5

l'interface QueueSession et TopicSession.....5

L'API JMS (2).....6

l'interface QueueSender6

l'interface QueueReceiver.....6

LES MESSAGES (1).....7

la structure des messages.....7

LES MESSAGES (2).....8

les types de messages.....8

quelques exemples simples.....8

LES MESSAGES (3).....9

quelques exemples (suite).....9

UN EXEMPLE SIMPLE (1).....10

le client et le serveur d'écho utilisant des queues temporaires10

UN EXEMPLE SIMPLE (2).....11

le code du serveur.....11

LES SÉLECTEURS.....12

les objectifs.....12

QUELQUES ÉLÉMENTS D'ADMINISTRATION DES PLATES-FORMES JORAM (1).....13

les concepts.....13

QUELQUES ÉLÉMENTS D'ADMINISTRATION DES PLATES-FORMES JORAM (2).....14

le lancement des serveurs.....14

les fichiers de configuration.....14

QUELQUES ÉLÉMENTS D'ADMINISTRATION DES PLATES-FORMES JORAM (3).....15

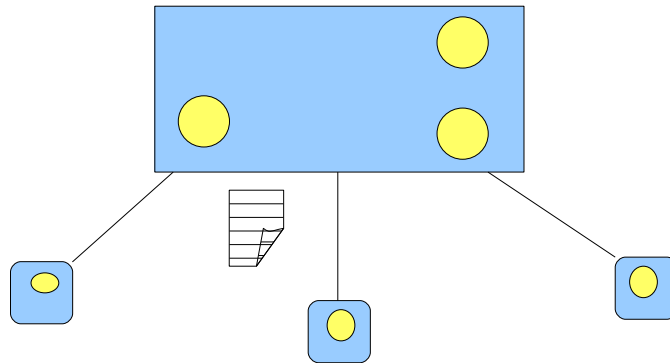
l'administration des serveurs.....15

Quelques généralités (1)

les objectifs de JMS (Java Messaging Service)

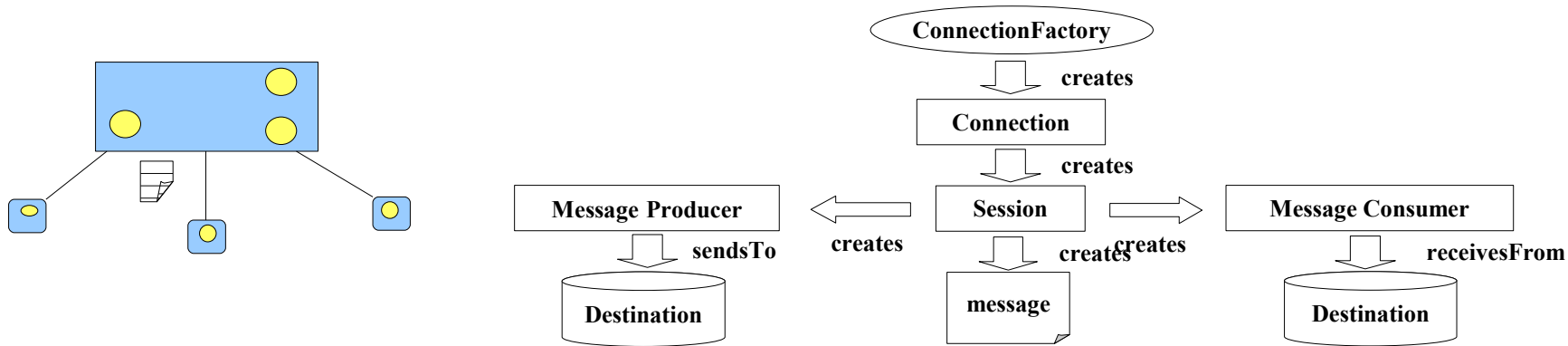
- un modèle standard et une API de communication par messages
- un modèle général + 2 spécialisations :
 - un modèle de communication Point à Point
 - un modèle de communication Publish and Subscribe
- communication synchrone : send, receive
- communication asynchrone
 - association d'un listener déclenché à la réception d'un message sur une file
 - installation d'un selecteur (permettant de filtrer les messages)

Mais ne spécifie aucune fonction de :
load balancing
notification des erreurs
administration des ressources
sécurité
annuaire
protocole de communication



Le modèle JMS

les concepts principaux



concepts	Point à Point	Publish and Subscribe	
Connection	QueueConnection	TopicConnection	donne accès aux services de la plate-forme JMS
Destination	Queue	Topic	file de messages cible des messages produits source des messages consommés
Session	QueueSession	TopicSession	ressources et contexte monothreadé pour produire et consommer des message
Producer	QueueSender	TopicPublisher	file de messages en émission
Consumer	QueueReceiver	TopicSubscriber	file de messages en réception
Message			unité d'échange dans un acte de communication
MessageListener			objet invoqué par une destination à la réception d'un message (onMessage())
MessageSelector			expression (sous la forme d'une String) servant à filtrer les messages


L'API JMS (2)

l'interface QueueSender

- void send(Message msg), void send(Message msg, int deMode, int prio, int TTL)
- void close()
- getPriority(), getDeliveryMode(), getTimeToLive(),

l'interface QueueReceiver

- Message receive(), Message receive(long timeout)
- MessageListener getMessageListener ()
- String getMessageSelector()
- void close()



```
interface MessageListener {  
    void onMessage(Message msg);  
}
```

Les messages (1)

la structure des messages

- un message est formé de 3 parties : l'entête (Header), les propriétés (Properties), le corps (Body)
- l'entête contient des champs prédéfinis
 - JMSDestination, JMSDeliveryMode [PERSISTANT, NON_PERSISTENT]
 - JMSMessageID, JMSTimestamp, JMSCorrelationID, JMSReplyTo, JMSPriority, ...
- les propriétés sont un ensemble d'informations additionnelles (utilisées par les sélecteurs)
 - propriété = (name, value)
 - propriétés « JMS predefined » identifiées par le préfixe JMSX :
JMSXUserID, JMSXDeliveryCount, JMSXGroupID, JMSXGroupSeq,
ConnectionMetaData.getJMSXPropertyNames()
 - propriétés "Provider-Specific" identifiées par le préfixe JMS_<nom_vendeur>
 - propriétés "User-specific"
- l'API permet d'attribuer/récupérer les propriétés prédéfinies des messages
 - String getJMSCorrelationID(), int getJMSDeliveryMode(), Destination getJMSDestination(), long getJMSTypeStamp(), int getJMSExpiration(), ...
 - void setJMSCorrelationID(String id), void getJMSDeliveryMode(int mode), void getJMSDestination(Destination dest), ...
- l'API permet d'attribuer/récupérer des propriétés additionnelles des messages
 - boolean getBooleanProperty(String nm), int getIntProperty(String nm), Object getObjectProperty(String nm),
getStringProperty(String nm),
 - void setBooleanProperty(String nm, boolean val), void setIntProperty(String nm, int val),

Les messages (2)

les types de messages

- `BytesMessage` : message manipulé comme un `OutputStream` ou `InputStream`
- `StreamMessage` : message manipulé comme un `DataOutputStream` ou un `DataInputStream`
- `ObjectMessage` : message manipulé comme un `ObjectOutputStream` ou un `ObjectInputStream`
- `MapMessage` : message manipulé comme un ensemble de couple (clé, valeur)
- `TextMessage` : message manipulé comme une `String`

quelques exemples simples

.....

//un `BytesMessage`

```
BytesMessage bytesMessage = queueSession.createBytesMessage();
```

```
bytesMessage.writeBytes(byteData);
```

```
bytesMessage.reset();
```

```
length = bytesMessage.readBytes(inByteData);
```

.....

//un `TextMessage`

```
TextMessage textMessage = queueSession.createTextMessage();
```

```
textMessage.setText(msgText);
```

```
System.out.println(" " + textMessage.getText());
```

les messages sont créés par les sessions

//byteData est un `byte[]`

Les messages (3)

quelques exemples (suite)

//un MapMessage

```
MapMessage mapMessage = queueSession.createMapMessage();
mapMessage.setString("Message type", "Map");
mapMessage.setInt("An Integer", 3456);
mapMessage.setDouble("A Double", 1.23456789);
System.out.println(" Type: " + mapMessage.getString("Message type"));
System.out.println(" Double: " + mapMessage.getDouble("A Double"));
System.out.println(" Integer: " + mapMessage.getInt("An Integer"));
```

//un StreamMessage

```
StreamMessage streamMessage = queueSession.createStreamMessage();
streamMessage.writeString("Stream message");
streamMessage.writeDouble(123.456789e222);
streamMessage.writeInt(223344);
streamMessage.reset();
System.out.println(" String: " + streamMessage.readString());
System.out.println(" Double: " + streamMessage.readDouble());
System.out.println(" Integer: " + streamMessage.readInt());
```

Un exemple simple (1)

le client et le serveur d'écho utilisant des queues temporaires ...

```
.....
try {
    InitialContext ictx = new InitialContext();
    QueueConnectionFactory qCnF = (QueueConnectionFactory) ictx.lookup("cf0");
    QueueConnection qConnection = qCnF.createQueueConnection("root","root");
    QueueSession qSession = qConnection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
    Queue srvQueue = (Queue) ictx.lookup("echo-queue");

    cltQueue = qSession.createTemporaryQueue();
    cltQueue.setFreeWriting(true);
    cltQueue.setFreeReading(true);

    QueueSender sender = qSession.createSender(srvQueue);
    QueueReceiver receiver = qSession.createReceiver(cltQueue);
    qConnection.start();

    for (int i = 0; i < 50; i++) {
        StreamMessage msg = qSession.createStreamMessage();
        msg.setJMSReplyTo(cltQueue);
        msg.writeDouble(Math.random());
        sender.send(msg);
        StreamMessage msg1 = (StreamMessage) receiver.receive();
        System.out.println("valeur " + msg1.readDouble());
    }
} catch (JMSEException e) { .....
} finally { try { qConnection.close(); } catch (JMSEException e) { .... }
}
.....
```

utilisation d'un serveur JNDI (création d'un contexte initial suivie d'une recherche dans un espace de nommage arborescent). La ConnectionFactory a été enregistré préalablement dans le contexte initial sous le nom cf0. La Queue du serveur sous le nom echo-queue

création d'une Queue temporaire comme destination de réponse

Un exemple simple (2)

le code du serveur

utilisation d'un serveur JNDI (création d'un contexte initial suivie d'une recherche dans un espace de nommage arborescent). La ConnectionFactory a été enregistré préalablement dans le contexte initial sous le nom cf0. La Queue du serveur sous le nom echo-queue

```
.....
try {
    InitialContext ictx = new InitialContext();
    QueueConnectionFactory qCnF = (QueueConnectionFactory) ictx.lookup("cf0");
    QueueConnection qConnection = qCnF.createQueueConnection("root","root");
    QueueSession qSession = qConnection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
    Queue srvQueue = (Queue) ictx.lookup("echo-queue");

    QueueReceiver receiver = qSession.createReceiver(srvQueue);
    while (true) {
        StreamMessage msg = (StreamMessage ) receiver.receive();
        QueueSender sender = qSession.createSender(msg.getJMSReplyTo());
        msg.setReplyTo(null);
        sender.send(msg);
    }
} catch (JMSEException e) { .....
} finally { try { qConnection.close(); } catch (JMSEException e) { .... } }
```

Les sélecteurs

les objectifs

- un filtre associé à un Message Consumer (QueueReceiver ou TopicSubscriber)
 - soit à la création du Message Consumer
 - soit à la réception d'un message (argument du receive)
- une expression de filtrage construite à partir des champs de l'en-tête et des propriétés
 - String dont la forme correspond à un sous ensemble des expressions conditionnelles de SQL92
 - Opérateurs logiques : NOT, AND, OR
 - Opérateurs de comparaison : =, > , >=, <, <= , <>
 - BETWEEN, LIKE,

```
import javax.jms.*;
.....

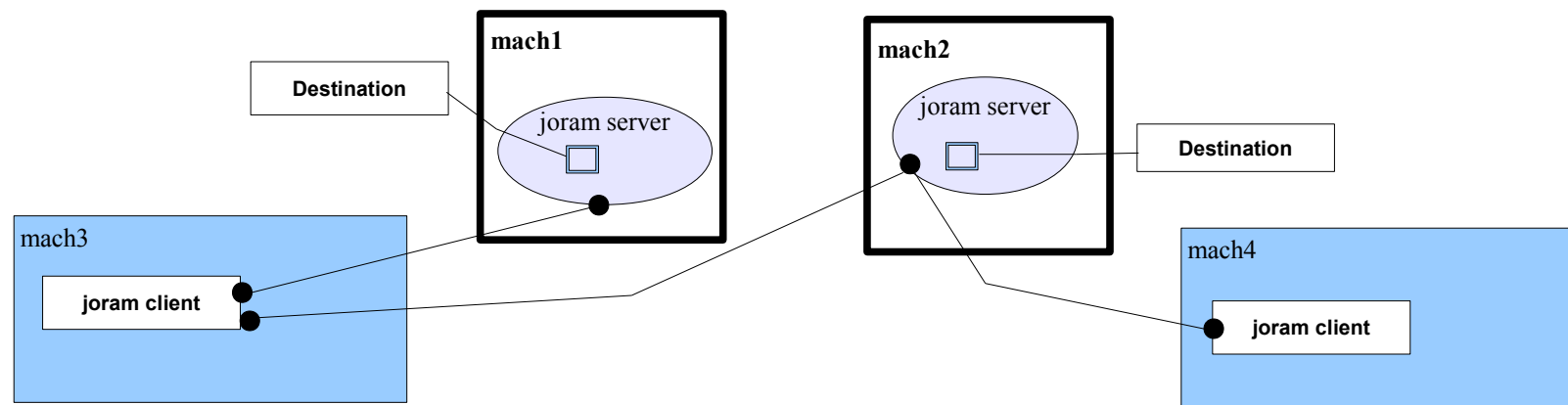
try {
    .....
    String selector = "JMSCorrelationID = 50 OR orderValue > 2500 ";
    receiver = session.createQueueReceiver(replyQ, selector);
    TextMessage replyMsg = receiver.receive();

} catch (JMSException e) {
    .....
} finally {
    .....
    try { queueConnection.close(); } catch (JMSException e) { .... }
    .....
}
```

Quelques éléments d'administration des plates-formes Joram (1)

les concepts

- un serveur Joram : une entité java offrant des fonctions de messagerie et hébergeant des Destinations
- un client JMS Joram : une entité java utilisant les fonctions de messagerie.
doit se connecter à un/des serveur(s) Joram



- plate-forme joram = ensemble des serveurs Joram
- administration de la plate-forme = lancement et configuration des serveurs
- un serveur Joram
 - administration de la plate-forme (gestion des connexions)
 - communication avec les clients (proxy TCP)
 - un service JNDI

Quelques éléments d'administration des plates-formes Joram (2)

le lancement des serveurs

- java fr.dyade.aaa.agent.AgentServer
- le fichier de configuration d'un agent : a3servers.xml

doit être dans le **REPERTOIRE DE LANCEMENT** de la commande :
java fr.dyade.aaa.agent.AgentServer 0 ./s0 !!!!!

les fichiers de configuration

- a3servers.xml contient les informations nécessaires au lancement et à l'accès des services offerts par les serveurs

```
<?xml version= "1.0"?>
<config>
  <property name= "Transaction" value="fr.dyade.aaa.util.NullTransaction"/>
  <server id="0" name="s0" hostname="localhost">
    <service class="org.objectweb.joram.mom.proxies.ConnectionManager" args="root root"/>
    <service class="org.objectweb.joram.momproxies.tcp.TcpProxyService" args="16010"/>
    <service class="fr.dyade.aaa.jndi2.server.JndiServer" args="16400"/>
  </server>
  <server id="1" name="s1" hostname="host1">
    .....
  </server>
</config>
```

doit être dans le CLASSPATH des applications

- jndi.properties contient les informations nécessaires aux clients pour accéder à l'annuaire :

```
java.naming.factory.initial fr.dyade.aaa.jndi2.client.NamingcontextFactory
java.naming.factory.host localhost
java.naming.factory.port 16400
```

Quelques éléments d'administration des plates-formes Joram (3)

l'administration des serveurs

.....

//Connexion à l'administrateur

AdminModule.connect("root", "root", 60);

//User user0 = User.create("fho", "fhofho", 0);

//Creation de l'utilisateur fho sur le serveur 0

//creation de destinations sur le serveur 0:

Queue echoQ = (Queue) Queue.create(0);

//droits de lecture/ecriture pour tous

philoQ.setFreeReading();

philoQ.setFreeWriting();

synchroQ.setFreeReading();

synchroQ.setFreeWriting();

//creation d'une ConnectionFactory pour le serveur local

javax.jms.ConnectionFactory cfact = TcpConnectionFactory.create("localhost", 16010);

//enregistrement dans l'annuaire JNDI:

javax.naming.Context jndiCtx = new javax.naming.InitialContext();

jndiCtx.rebind("echo-queue", echoQ);

jndiCtx.rebind("cf0", cfact);

jndiCtx.close();

enregistrement de la ConnectionFactory et
de la Queue directement dans le contexte initial

AdminModule.disconnect();

.....