Institut Informatique
Claude Chappe

Master Informatique

Programmation distribuée
M1 / 178UD02

# C3 – JMS

Thierry Lemeunier
thierry.lemeunier@univ-lemans.fr

# Plan du cours

- Communication par message *MOM* :
  - Principes
  - Les messages
  - Fonctionnalités d'un *MOM*
  - JMS (Principes ; API ; Exemple)

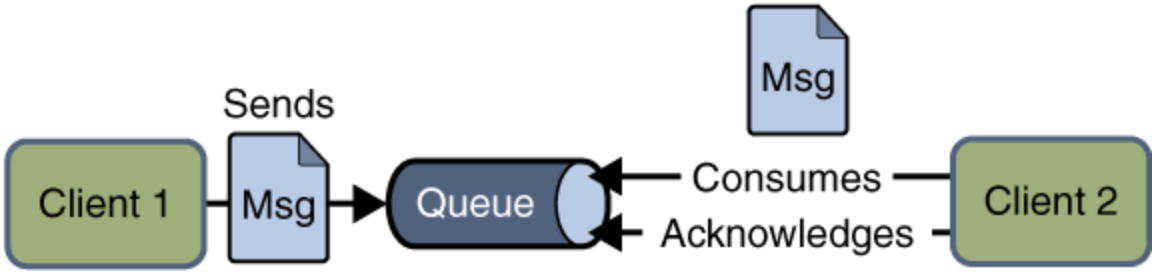
# Communication par message – Principes (1/3)

- Principe général :
  - Communication asynchrone pour nœuds faiblement couplés
    - Communication non bloquante pour la **production** d'un message
    - Communication indirecte du **producteur** et du **consommateur** du message via une boîte aux lettres (persistance des messages non acheminés)
  - ➔ *Message Oriented Middleware* (*MOM*)
- Philosophie :
  - Un *MOM* n'est pas orienté vers l'exécution de tâches (comme *RMI*) mais vers l'acheminement d'information
  - ➔ On sort de la « philosophie » mise à disposition de services distants
- Utilisations :
  - Diffusion massive d'informations
    - ➔ Systèmes d'interconnections bancaires (bourse électronique…)
  - Intégration d'applications hétérogènes
    - ➔ Systèmes indépendants et/ou évolutifs communiquant
  - Éloignement géographique important et/ou changeant
    - ➔ Systèmes ubiquitaires (omniprésents) et mobiles
  - Interruption normale de la communication
    - ➔ Surveillance et contrôle d'équipements

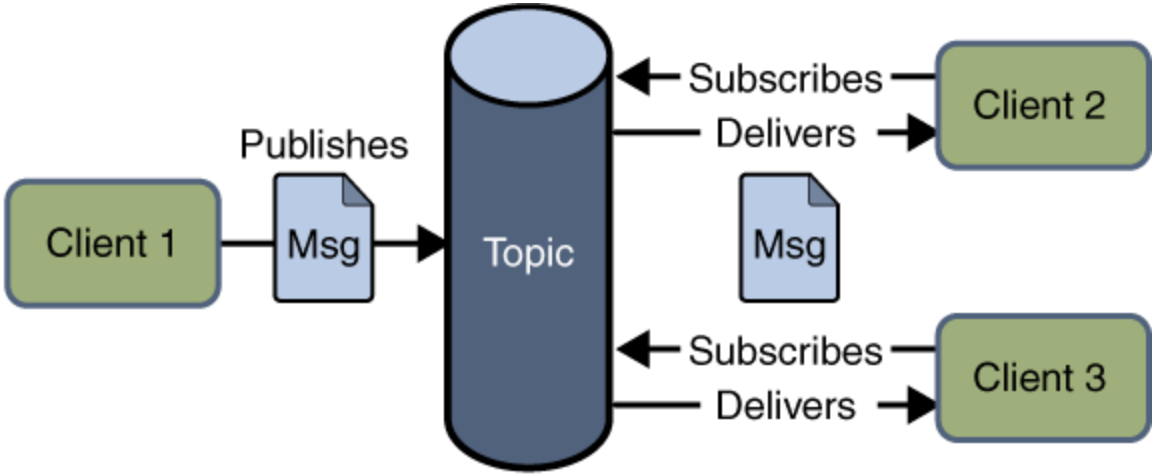
# Communication par message – Principes (2/3)

- Normes ?
  - Il n'y a pas de normalisation mais des solutions propriétaires…
  - *Java Messaging Service* (1998 puis 2002) est la spécification pour Java
- Les deux modèles de communication :
  - Communication point-à-point (*Point-To-Point*) :
    - Le message produit est consommé par un destinataire unique désigné par l'émetteur (1à1)
    - Exemple type : le système des *email*
  - Communication multipoints (*Publish and Subscribe*) :
    - Un message produit est consommé par une communauté de destinataires désignés (1àN) ou anonymes (diffusion NàN) qui doivent s'abonner
    - Exemple type : le système des *news*
- Désignation du destinataire
  - Le destinataire est connu par un nom symbolique
  - Le *MOM* doit gérer une base (éventuellement répartie) d'association entre le nom symbolique du destinataire et son adresse réseau

# Communication par message – Principes (3/3)




Point-To-Point Mode




Publish/Subscribe Mode

[ Source : The Java™ EE 5 Tutorial Third Edition ]

# Plan du cours

- Communication par message *MOM* :
  - ✓ Principes
  - Les messages
  - Fonctionnalités d'un *MOM*
  - JMS (Principes ; API ; Exemple)

# Communication par message – Les messages

- Un message est constitué de trois parties :
  - Une entête (des info. techniques tels que son identifiant, date de dépôt...)
  - Des propriétés fonctionnelles (différentes pour chaque application émettrice)
  - Des données applicatives

- Persistance des messages :
  - Les messages sont stockés dans des files ou des topics dans le cas du modèle multipoints ; ces files et topics étant gérés par le *MOM*
  - Le producteur, le consommateur et le stockage sont généralement sur des machines distinctes

- Acheminement des messages :
  - « Directement » entre le producteur et le consommateur via le *MOM*
  - Indirectement via plusieurs nœuds intermédiaires (*routers*) du *MOM*
  - Indirectement via des noeuds transformateurs de messages si l'émetteur et le destinataire utilisent des formats différents

- 2 modes de réception des messages (en *PTP* et en *Pub/Sub)* :
  - Le destinataire doit lire ses messages de lui-même
  - Le destinataire peut être informé automatiquement de la présence de messages en attente

# Plan du cours

- Communication par message *MOM* :
  - ✓ Principes
  - ✓ Les messages
  - Fonctionnalités d'un *MOM*
  - JMS (Principes ; API ; Exemple)

# Communication par message – Fonctionnalités

- Fonctionnalités d'un *MOM* :
  - Acheminement, stockage, recherche/filtrage des messages…
  - Gestion de message à priorité
  - Hiérarchie de topic
  - Compression du contenu applicatif des messages
  - Faire expirer un message à une date donnée
  - Ne rendre un message disponible qu'à partir d'une certaine date
  - Des services de routage des messages d'un noeud à l'autre (un peu à la manière des serveurs de mails)
  - Des fonctionnalités de *triggering* : lancement d'applications lorsque des messages sont disponibles pour elles
  - Des possibilités d'alertes suivant la présence de messages dans une file donnée ou suivant un nombre de messages donné
  - Etc.

- Remarque : *email* et *MOM*
  - Un *MOM* est un système de communication inter-applications informatiques
  - Le courrier électronique est un système pour la communication humaine

# Plan du cours

- Communication par message *MOM* :
  - ✓ Principes
  - ✓ Les messages
  - ✓ Fonctionnalités d'un *MOM*
  - JMS (Principes ; API ; Exemple)

# Communication par message – JMS – Principes

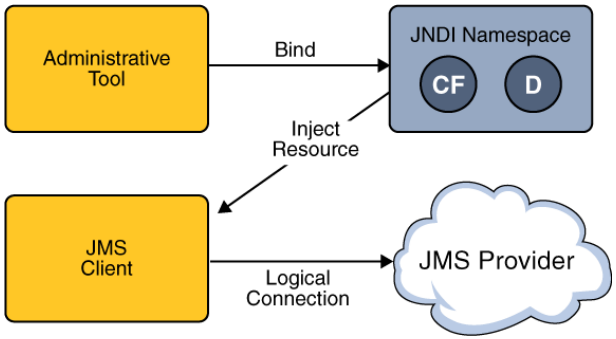
- ***API et provider***
  - *JMS* est une *API* Java de spécification d'un *MOM*
  - Package *javax.jms* (v1.1 en 2002 ; v2.0 en 2013)
  - Un *provider* est un éditeur qui implémente l'*API JMS*
    - ➔ IBM WebSphere MQ ; Apache ActiveMQ ; OpenJMS…
  - Le provider JMS fournit des outils d'administration (gestion des droits, création des files et des topics, gestion de la persistance, etc.)
- ***JMS* définit les 2 modèles de communication via une interface unique**
  - Mode *PTP* :
    - 1 producteur ➔ [File] du Provider *JMS* ➔ 1 consommateur
    - Le message disparaît de la file lorsque le consommateur acquitte la réception
  - Mode *Pub/Sub* :
    - 1 producteur ➔ [Topic] du Provider *JMS* ➔ N consommateurs/souscripteurs
    - Le message disparaît du topic lorsque tous les souscripteurs l'ont acquitté
    - Le souscripteur ne voit que les messages publiés postérieurement à sa date d'abonnement
  - Un message avec une date d'expiration disparaît avant d'être lu s'il a expiré !
- **2 types de réceptions des messages**
  - Synchrone : lecture sans ou avec timer bloquant (fini ou infini)
  - Asynchrone : notification (le consommateur implémente *MessageListener)*

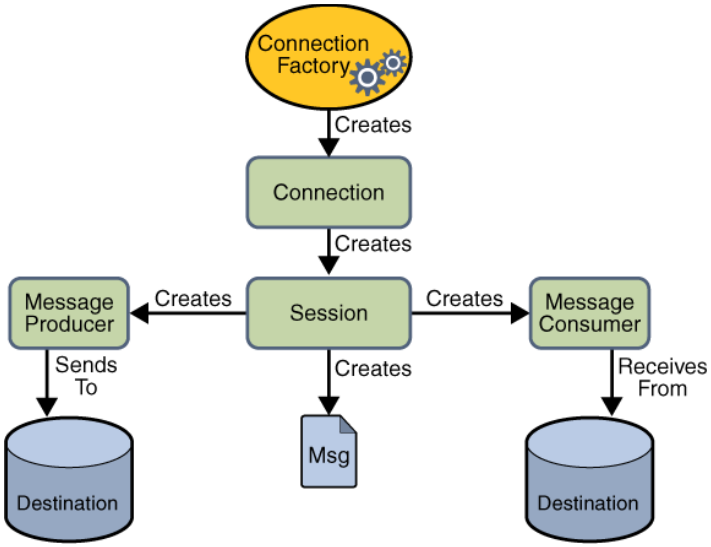
# Plan du cours

- Communication par message *MOM* :
  - ✓ Principes
  - ✓ Les messages
  - ✓ Fonctionnalités d'un *MOM*
  - JMS (Principes ; API ; Exemple)

# Communication par message – JMS – API (1/3)

- Les objets administrés de types *ConnectionFactory* et *Destination*
  - ***ConnectionFactory*** : la fabrique pour obtenir une connection au provider et tous les autres objets
  - ***Destination*** : les files et les topics du provider pour stocker les messages
  - Les objets administrés sont gérés par l'administrateur du provider
  - Le programmeur les récupère via le sercice *JNDI* (*Java Naming and Directory Interface*)
  - L'accès au provider via *JNDI* peut être programmé ou placé dans un fichier jndi.properties




```
java.naming.provider.url=tcp://localhost:3035
java.naming.factory.initial=org.exolab.jms.jndi.InitialContextFactory
java.naming.security.principal=admin
java.naming.security.credentials=openjms
```

[ Source : The Java™ EE 5 Tutorial Third Edition ]

# Communication par message – JMS – API (2/3)

- Interface ***Connection*** :
  - La connection au provider retourné par la *ConnectionFactory*
  - Login et mot de passe sont lus dans jndi.properties ou donnés en argument
- Interface ***Session*** :
  - Session de travail avec le provider. Il existe deux types de session :
    - Les sessions transactionnelles utilisées pour groupé l'envoi/réception de messages. La transaction n'est validée qu'après appel explicite à l'ordre *commit*
    - Les sessions non-transactionnelles (*commit* est optionnel dans ce cas)
  - L'objet *Session* est obtenu auprès de l'objet *Connection*
- Interface ***Message*** :
  - 6 types de message possibles
    - *Message* : message sans données (utilisé pour la fin d'une communication…)
    - *TextMessage* : message contenant un objet *String*
    - *ByteMessage* : message contenant un flux d'octets
    - *StreamMessage* : message contenant un flux de types primitifs du langage Java
    - *MapMessage* : message contenant une table de hachage contenant des types primitifs
    - *ObjectMessage* : message contenant un objet Java sérialisable
  - Les champs de l'entête du message sont accessibles par des getter/setter (cf. doc !)
  - Les propriétés sont des couples [*attribut*, *valeur_de_type_primitif*] (cf. doc !)

# Communication par message – JMS – API (3/3)

- Interface ***MessageProducer*** :
  - Créé par la *Session*, il envoie un message à une *Destination*
  - On peut indiquer un temps de vie du message, une priorité (0 à 9) et un mode de stockage en cas d'arrêt du provider (persistant par défaut)
- Interface ***MessageConsumer*** :
  - Créé par la *Session*, il permet de lire de façon synchrone les messages depuis une file ou un topic auquel il souscrit
  - Pour lire les messages publiés antérieurement sur le topic souscrit, il faut être inscrit comme souscripteur durable auprès de ce topic
- Interface ***MessageListener*** :
  - Associé à un *MessageConsumer*, il permet de faire une lecture asynchrone des messages d'une file ou d'un topic
- Sélecteurs SQL :
  - Permet de filtrer les messages d'une file ou d'un topic lus par un *MessageConsumer* sur certains champs de l'entête et les propriétés
  - Le sélecteur est indiqué à la création du *MessageConsumer*

# Plan du cours

- Communication par message *MOM* :
  - ✓ Principes
  - ✓ Les messages
  - ✓ Fonctionnalités d'un *MOM*
  - JMS (Principes ; API ; Exemple)

# Communication par message – JMS – Exemple (1/3)

**Extrait du code d'un producteur d'un message texte**

```
…
try {
   context = new InitialContext();    // create the JNDI initial context using properties file
   factory = (ConnectionFactory) context.lookup("ConnectionFactory");    // look up the ConnectionFactory
   dest = (Destination) context.lookup("Queue1");     // look up the Destination

   connection = factory.createConnection();     // create the connection
   session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);    // create the session
   sender = session.createProducer(dest);    // create the sender
   connection.start();    // start the connection to enable message sends

   TextMessage message = session.createTextMessage("Hello World");
   sender.send(message);

} catch (JMSException exception) { exception.printStackTrace(); }
  catch (NamingException exception) { exception.printStackTrace(); }
  finally {
   if (context != null)  // close the context
      try { context.close(); } catch (NamingException exception) { exception.printStackTrace(); }
   if (connection != null)  // close the connection
      try { connection.close(); } catch (JMSException exception) { exception.printStackTrace(); }
}
…
```

[ Source : The Java™ EE 5 Tutorial Third Edition ]

# Communication par message – JMS – Exemple (2/3)

**Extrait du code d'un consommateur asynchrone**

```
…
try {
   context = new InitialContext(); // create the JNDI initial context
   factory = (ConnectionFactory) context.lookup(factoryName); // look up the ConnectionFactory
   dest = (Destination) context.lookup(destName); // look up the Desination
   connection = factory.createConnection(); // create the connection
   session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); // create the session
   receiver = session.createConsumer(dest); // create the receiver

   receiver.setMessageListener(new SampleListener()); // register a listener
   connection.start(); // start the connection, to enable message receipt

   System.out.println("Press [return] to quit");
   new BufferedReader(new InputStreamReader(System.in)).readline();

} catch (IOException exception) { exception.printStackTrace(); }
  catch (JMSException exception) { exception.printStackTrace(); }
  catch (NamingException exception) { exception.printStackTrace(); }
  finally {
    if (context != null) // close the context
       try { context.close(); } catch (NamingException exception) { exception.printStackTrace(); }
    if (connection != null) // close the connection
       try { connection.close(); } catch (JMSException exception) { exception.printStackTrace(); }
}
…
```

[ Source : The Java™ EE 5 Tutorial Third Edition ]

# Communication par message – JMS – Exemple (3/3)

**Le *MessageListener* de réception des messages**

```
public class SampleListener implements MessageListener {

    public void onMessage(Message message) {
        if (message instanceof TextMessage) {
            TextMessage text = (TextMessage) message;
            try {
                System.out.println("Received: " + text.getText());
            } catch (JMSException exception) {
                System.err.println("Failed to get message text: " + exception);
            }
        }
    }

}
```

[ Source : The Java™ EE 5 Tutorial Third Edition ]