Introduction à JMS et son intégration dans Spring

par Gildas Cuisinier (Hikage.be) (Blog)

Date de publication : 05 juin 2008

Dernière mise à jour :

I - Introduction

Qu'est-ce que JMS ?

JMS (pour Java Messaging System) est l'implémentation Java de ce qui est appellé MOM.

MOM, ou Message-Oriented Middleware

Un MOM est une plateforme logicielle fournissant un moyen de communication entre divers applications, sans que celles-ci soient consciente de l'existence de l'autres.
Le principe est qu'une application ne communique pas directement avec l'autre, mais dépose son "message" dans le MOM. L'autre application de son coté, viendra simplement vérifier l'arrivée de message sur le MOM.

Quels sont les avantages d'une communication JMS par rapport à une communication de type TCP ou Corba ?

Faible Couplage

Pour communiquer, la seule chose qui est nécessaire à deux applications, c'est d'avoir un message compréhensible par les deux cotés, et un serveur JMS commun. Pas besoin de connaitre l'adresse de l'autre, pas besoin de stub pour accèder au service de l'autre.

Intérêt : Si l'application réceptrice est modifiée, déplacée ou remplacée, cela reste complètement transparent du point de vue de l'émetteur

Asynchrone

JMS est un protocole asynchrone, ce qui veut dire que l'application émettrice émet son message, et continue son traitement sans attendre que le récepteur confirme l'arrivée du message.
De son coté, le récepteur récupère les messages quand il le souhaite.

Intérêt : Si l'application réceptrice effectue une tâche longue, l'émetteur n'est pas bloqué

Persistant

JMS propose un mode persistant, c'est à dire que les messages sur une Queue/Topic (envoyé par un émetteur et non encore consommé par un client) sont stocké de manière persistance (disque, base de données, ...).
Ce qui assure une garantie de livraison du message, même si le serveur JMS devait être arrêté (maintenance, plantage, ...).

Intérêt : Assurance que le message ne sera pas perdu et arrivera bien au client

Transactionel

Une communication JMS peut être comprise dans une transaction, locale ou globale (via JTA/XA).
Imaginons qu'une demande d'abonnement est réalisée via JMS, le client reçoit le message, commence le traitement mais une exception est levée. Le message JMS bien que lu par le client n'a pas été consommé et il est remis sur la queue et sera relu plus tard pour une nouvel essai.

Intérêt : Pas de perte de messages en cas de problème dans l'application

Présentation technique de JMS

Quelques définitions

Dans une architecture de communication, il y a plusieurs intervenants et composants qui interviennent.

Le provider JMS

JMS est l'API Standard définie par Sun, il faut donc une implémentation concrête de celle-ci pour pouvoir réellement communiquer en JMS.
Un Provider JMS est donc une implémentation l'interface JMS.

Voici une liste (non exaustive) de provider JMS :

Implémentation	Licence	Description
Apache ActiveMQ	OpenSource - Apache 2.0 License	Une des implémentations OpenSource les plus populaire **Site officiel**
OpenJMS	OpenSource	**Site du projet**
JBoss Messaging	OpenSource	Implementation du serveur JBoss, remplacant de JBossMQ **Site officiel**
Joram	OpenSource - LGPL license	**Site officiel**
WebShere MQ - IBM	Commercial	Implémentation JMS de IBM **Site officiel**
Sonic MQ	Commercial	**Site officiel**
Oracle Advanced Queueing	Commercial	
TIBCO Enterprise Message Service	Commercial	**Site officiel**
Sun Java System Message Queue	Commercial	**Site officiel**

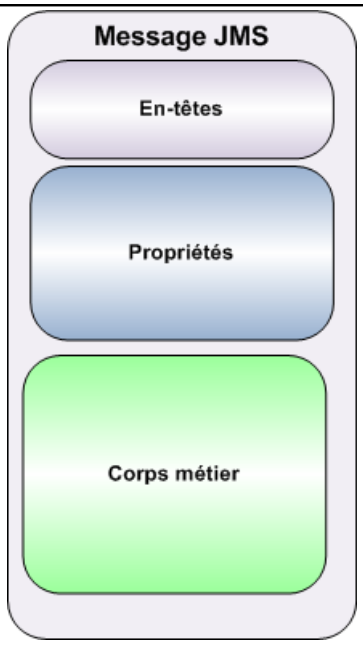
Client JMS

Les clients JMS sont les applications qui utilise JMS pour communiquer, elle sont de deux types :

- Producteur : Client qui crée et envoie des messages
- Consomateur : Client qui reçoit des messages

Message JMS

Un message JMS est la structure qui transite dans une communication JMS. Il est composé de plusieurs parties :

	**En-têtes :** Divers informations d'ordre technique du message : identifiant, identifiant de correlation, priorité, durée de validité **Propriétés : Information de type "clé -> valeurs" qui peuvent être ajouter par l'application pour apporter des précision du le message** **Corps du message : Données métiers du message**

Les différents modèles de communications

(

L'API JMS

Petit exemple

II - JMS avec Spring Framework

L'API Spring pour JMS

Exemple pratique

Etude de cas

Installation de ActiveMQ

Configuration Spring : ConnectionFactory et Queues

Création du producteur

```
public class test{

}
```