Introduction à JMS

Introduction

- JMS = Java Messaging Service
- JMS est une *API Java* proposée par Sun, permettant aux applications réparties de créer, envoyer, recevoir et lire des messages
- première specification publiée en 1998
- un *fournisseur JMS* (*JMS provider*) est un système qui implémente les interfaces JMS (et offre des outils d'administration)

Remarque:

- JMS permet un *couplage faible* entre entités réparties (les différentes entités n'ont pas besoin d'être actives simultanément)
- Par comparaison RMI, CORBA: *couplage fort* entre entités réparties (les différentes entités doivent être actives simultanément)

Références

- R. Monson-Haefel, D.A. Chappell
  Java Message Service
  O'Reilly, 2001

- P. Giotta et al.
  Professional JMS Programming
  Wrox Press, 2000

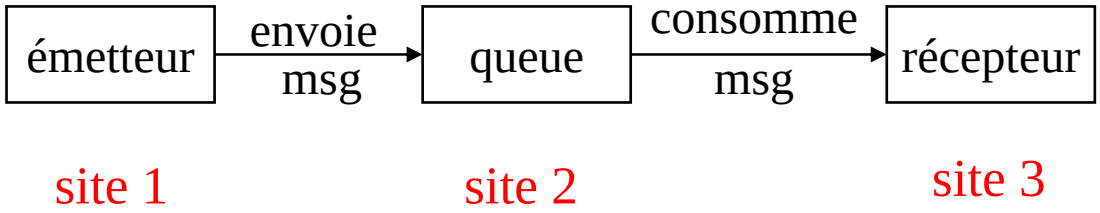
- http://java.sun.com/products/jms/tutorial/

JMS: les deux modèles de communication

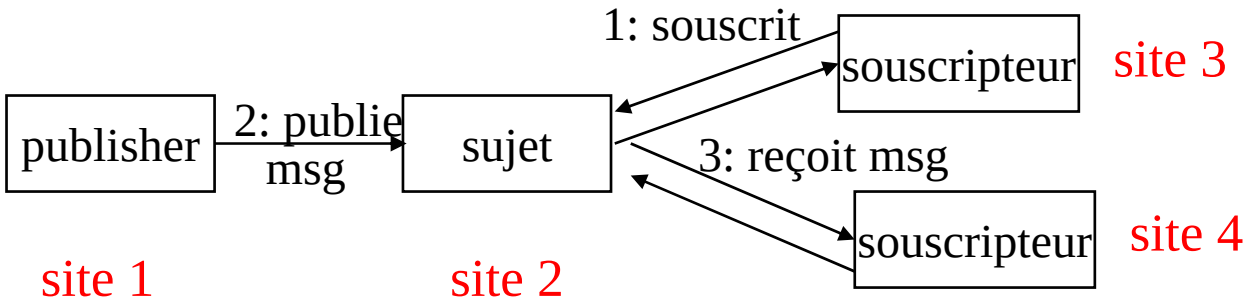
JMS offre deux modèles de communication par échange de messages:

- communication point-à-point basé sur une queue de messages (1 vers 1). Il peut y avoir plusieurs émetteurs et plusieurs récepteurs.
- publish-subscribe (1 vers n). Il peut y avoir plusieurs "publishers".

point-à-point



pub/sub



JMS: les deux modèles de communication (2)

Exemple d'utilisation d'une queue:

- Une queue pour stocker les commandes d'une entreprise
- Les représentants de l'entreprise envoient des commandes dans cette queue
- Les unités de fabrication prélèvent des commandes dans cette queue pour exécution

JMS: les deux modèles de communication (3)

Exemple d'utilisation du modèle publish/subscribe:

- Bourse: les sujets (topics) correspondent aux différentes entreprises cotées
- La bourse publie régulièrement une mise à jour des cours des différentes entreprises
- Un client intéressé au cours des actions IBM, souscrit à ce sujet. Un sujet correspond ici à une entreprise X, et les messages publiés aux différents cours de l'action de l'entreprise X

Communication point-à-point (PTP)

- l'émetteur envoie son message dans une queue
- le destinataire prélève un message dans une queue (chaque message n'est reçu que par un seul destinataire). Les messages sont prélevés dans l'ordre FIFO.
- les messages peuvent être *persistants* ou *non persistants* (seul un message persistant survit à la défaillance du serveur qui gère la queue)

Pub/sub

- le "publisher" adresse son message à un sujet
- un souscripteur souscrit à un sujet
- un message peut être reçu par de multiples destinataires (les différents souscripteurs)
- un souscripteur qui souscrit à l'instant t ne reçoit que les messages publiés après t. Par ailleurs, le souscripteur doit rester actif pour continuer à recevoir les messages publiés.
- souscription *durable*: permet au souscripteur de recevoir les messages publiés pendant les périodes ou le souscripteur n'est pas actif
- les messages peuvent être *persistants* ou *non persistants* (seul un message persistant survit à la défaillance du serveur qui gère le sujet)

NB La durabilité est une propriété liée au souscripteur, alors que la persistance est une propriété liée au serveur qui gère le sujet

Réception de messages

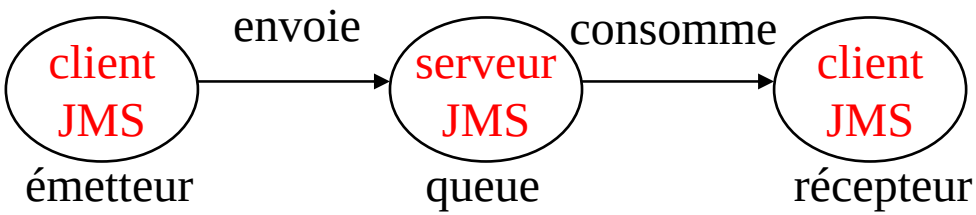
La réception de messages peut être *synchrone* ou *asynchrone*:

- **réception synchrone**: le destinataire (modèle PTP) ou le souscripteur (modèle pub/sub) appelle la méthode `receive`. L'appel est bloquant jusqu'à ce qu'un message soit obtenu. Il est possible de spécifier un timeout.

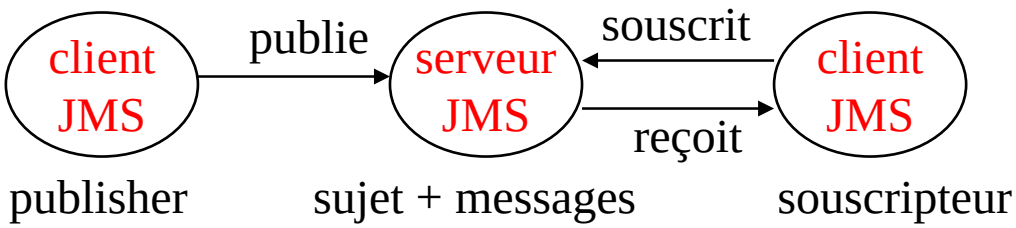
- **réception asynchrone**: le destinataire (PTP) ou le souscripteur peut enregistrer un `message listener`, qui est un objet contenant une méthode `onMessage`. Cette méthode est appelée lors de la réception d'un message.

Client JMS / serveur JMS

- *client JMS*: programme ou composant écrit en Java qui envoie ou reçoit des messages
- *serveur JMS*: composant qui permet à des clients JMS de communiquer entre eux (*serveur JMS* est fournit par un *JMS provider*)



point-à-point



pub/sub

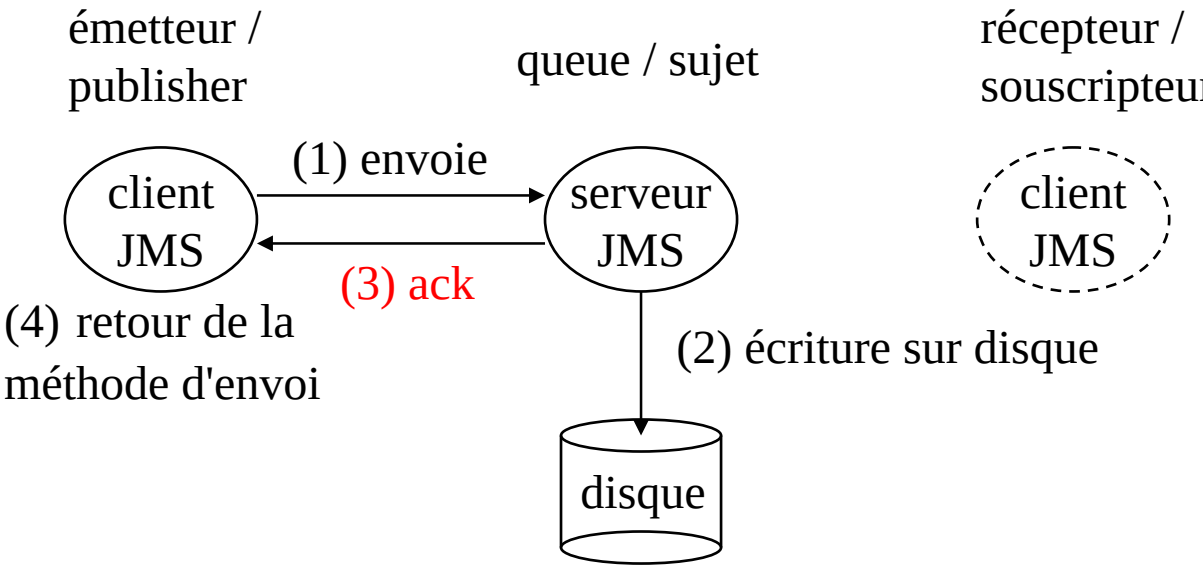
Messages persistants

- Les messages persistants (PTP ou pub/sub) sont conservés par le serveur JMS sur disque. Les messages survivent ainsi à une défaillance du serveur JMS.

- Dans le cas de messages persistants, le protocole entre *client JMS* et *serveur JMS* inclus un message d'acquittement (cf slides suivants)

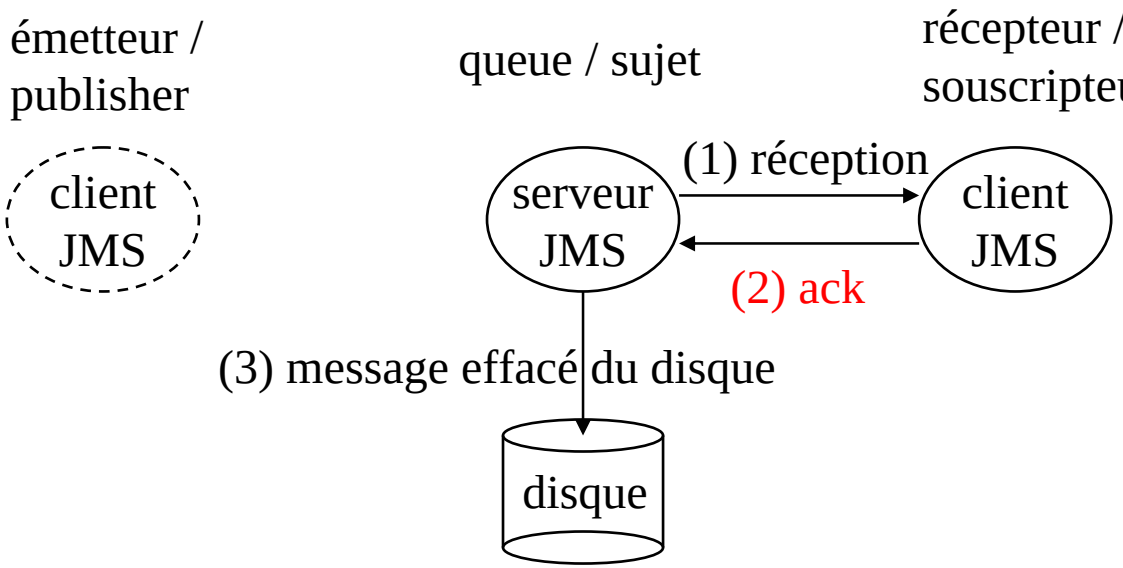
Messages persistants (2)

Envoi de message persistant: la méthode appelée par le client ne se termine que lorsque le message est stocké sur disque



Messages persistants (3)

Réception de message persistant: le message n'est effacé du disque que lorsque le serveur JMS a reçu un ACK du client JMS.



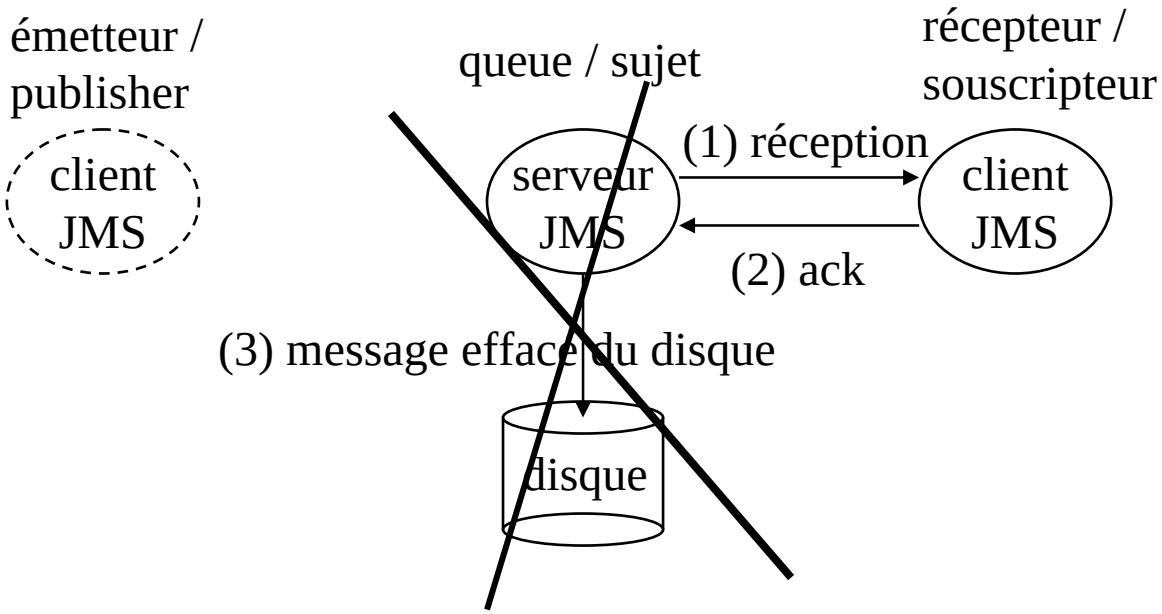
Envoi des ACKs

- En mode `AUTO_ACKNOWLEDGE`, les ACKs sont envoyés automatiquement, soit à la fin de la méthode `receive()` (réception synchrone), soit à la fin de la méthode `onMessage()` (réception asynchrone)

- Le récepteur peut également contrôler manuellement l'envoi du ACK. Pour cela il doit utiliser le mode `CLIENT_ACKNOWLEDGE`. Le ACK est envoyé en appelant la méthode `message.acknowledge()`.

Problème des défaillances

- Si le serveur JMS tombe en panne, le message reçu par le client JMS n'est pas nécessairement effacé du disque. Dans ce cas, lorsque le serveur JMS redémarre, il cherchera à renvoyer le même message. Il peut être souhaitable d'éviter qu'un même message ne soit reçu plus d'une fois par l'application.



Problème des défaillances (2)

- En mode `AUTO_ACKNOWLEDGE` une table des ID de messages reçus est gérée automatiquement sur le client JMS. Lors de la réception d'un message, si l'ID figure dans la table (message déjà reçu), le message est ignoré.
- Si la réception multiple d'un message ne pose pas de problème au client JMS, celui-ci peut se mettre dans le mode `DUPS_OK_ACKNOWLEDGE`. Dans ce cas le client JMS ne gère pas de table avec les ID des messages reçus (comme dans le mode `AUTO_ACKNOWLEDGE`).
- En mode `CLIENT_ACKNOWLEDGE`, aucune table d'ID de messages n'est gérée. Toutefois une exception est levée chez le client JMS en cas de défaillance du serveur JMS (qui pourrait conduire le serveur JMS à redélivrer le même message). Lorsque l'exception est levée, le client JMS doit se préparer à recevoir le message une 2ème fois (et par exemple l'ignorer).

Transactions JMS

Une *transaction JMS* permet de grouper l'envoi de plusieurs messages ou la réception de plusieurs messages.

Exemple d'envoi transactionnel:

- un client JMS envoie *m1* et *m2*, mais ne souhaite pas qu'en cas de défaillance (du client) seul *m1* soit reçu par un serveur JMS.

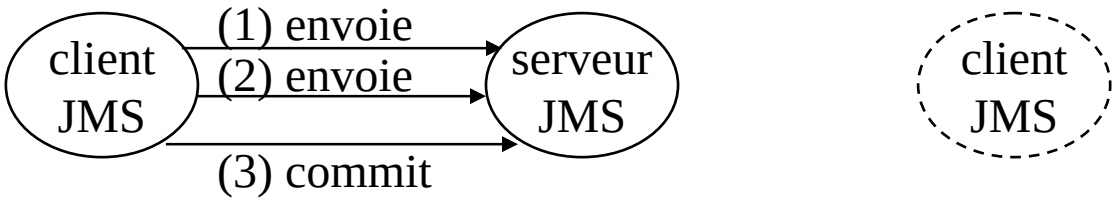
Exemple de réception transactionnelle:

- Un client JMS souhaite recevoir deux messages *m1 et m2*, et ne souhaite pas qu'en cas de défaillance (du client) seul *m1* ait été reçu et traité.

Transactions JMS (2)

- Une transaction JMS est indiquée par un paramètre au moment de la création de la connexion entre le client JMS et le serveur JMS
- La méthode `commit()` est appelée pour indiquer la fin de la transaction

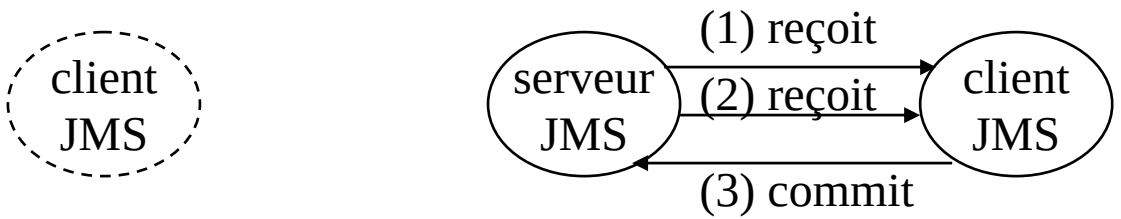
Exemple 1: transaction de l'émetteur



Les messages sont gardés par le serveur JMS; ils ne sont pas délivrés à un client JMS tant que commit() n'a pas été invoqué. Si le client JMS invoque rollback() (au lieu de commit()), les messages sont détruits.

Transactions JMS (3)

Exemple 2: transaction du récepteur



Les messages envoyés au client JMS sont gardés par le serveur JMS; ils sont détruits uniquement lorsque le client invoque `commit()`. Si le client invoque `rollback()` (au lieu de `commit()`), les messages ne sont pas détruits par le serveur JMS.

Transactions JMS (4)

- Il est également possible de grouper dans une transaction une réception et un envoi de messages (par ex. réception d'une queue q1 et envoi dans une queue q2)

- Par contre, ne jamais grouper dans une transaction (1) un envoi de messages, suivi (2) d'une réception de messages. *Cela provoque un interblocage!*

Exemple 3