JBoss Enterprise Application Platform 5

Guide de d&#xF;marrage rapide pour le d&#xF;veloppement de transactions

Guide de démarrage avec JBoss Transaction Service

Édition 5.1.0

Last Updated: 2017-10-26

JBoss Enterprise Application Platform 5 Guide de d&#65533;marrage rapide pour le d&#65533;veloppement de transactions

Guide de démarrage avec JBoss Transaction Service
Édition 5.1.0

Andrew Dinn
Red Hat
adinn@redhat.com

Mark Little
Red Hat
mlittle@redhat.com

Jonathan Halliday
Red Hat
jhallida@redhat.com

Publié par

Misty Stanley-Jones
Red Hat
misty@redhat.com

Notice légale

Résumé

Ce guide est une introduction rapide destinée aux développeurs Java qui souhaitent écrire des applications avec les API de JBoss Transaction Service.

Table des matières

CHAPITRE 1. DÉMARRER AVEC JTA **3**
1.1. STRUCTURE DU PACKAGE 3
1.2. CONFIGURATION DES PROPRIÉTÉS 3
1.2.1. Spécifier la location du magasin d'objets 3
1.3. DÉMARQUER LES TRANSACTIONS 4
1.3.1. UserTransaction 4
1.3.2. TransactionManager 4
1.3.3. Transaction 4
1.4. IMPLÉMENTATIONS JTA LOCALES VERSUS DISTRIBUÉES 5
1.5. JDBC ET TRANSACTIONS 5
1.6. OPTIONS CONFIGURABLES 6

CHAPITRE 2. DÉMARRER AVEC JTS / OTS **8**
2.1. STRUCTURE DU PACKAGE 8
2.2. CONFIGURATION DES PROPRIÉTÉS 8
2.3. DÉMARRER ET STOPPER LE ORB ET BOA/POA 9
2.4. PRÉCISER LA LOCATION DU MAGASIN D'OBJETS. 10
2.5. PROPAGATION DE TRANSACTIONS IMPLICITES ET INTERPOSITION 10
2.6. CURRENT 12
2.7. RÉSILIATION DE TRANSACTION 12
2.8. USINE DE TRANSACTIONS 12
2.9. RECOVERY MANAGER 13

CHAPITRE 3. COMMENCER AVEC WEB SERVICES TRANSACTIONS ET XTS **14**
3.1. CONFIGURER LE COMPOSANT DE SERVICES WEB 14

ANNEXE A. HISTORIQUE DE RÉVISION **16**

CHAPITRE 1. DÉMARRER AVEC JTA

Ce chapitre récapitule les fonctions principales requises pour construire une application *Java Transactions API (JTA)*. Si vous n'êtes pas familiarisé avec JTA, commencer par lire la première section du *Guide de Dévelopement des Transactions*, fourni avec la suite de documentation d'Enterprise Application Platform.

1.1. STRUCTURE DU PACKAGE

Tout ce dont vous avez besoin pour écrire les application JTA de base est inclus dans Enterprise Application Platform. Les packages clé sont détaillés dans Packages liés au JTA.

Packages liés au JTA

com.arjuna.ats.jts

Inclut l'implémentation de JBoss Transaction Service des *APIs (Application Programming Interfaces)* de JTS et JTA.

com.arjuna.ats.jta

Contient le support d'implémentation JTA distant ou local.

com.arjuna.ats.jdbc

Contient le support JDBC 2.0 transactionnel.

1.2. CONFIGURATION DES PROPRIÉTÉS

Vous pouvez configurer JBossJTA en paramétrant divers attributs de propriété, soit en cours d'exécution sur la ligne de commande, ou bien à travers un fichier de propriétés. Le fichier de propriétés initial se situe à ***$JBOSS_HOME*/server/default/conf/jbossts-properties.xml**.

1.2.1. Spécifier la location du magasin d'objets

JBossJTA utilise un magasin d'objets pour enregistrer les résultats de transactions de manière persistante, pour qu'ils puissent être utilisés en cas d'échecs. Pour personnaliser la location du magasin d'objets, vous aurez besoin de passer la location quand vous exécutez l'application, comme illustré dans Exemple 1.1, « Spécifier le Magasin d'Objets ».

Exemple 1.1. Spécifier le Magasin d'Objets

```
   java –
Dcom.arjuna.ats.arjuna.objectstore.objectStoreDir=/location/of/objectsto
re myprogram
```

Par défaut, le magasin de l'objet est situé dans un répertoire qui se trouve sous le répertoire d'exécution en cours.

Par défaut, tous les états d'objets sont stockés dans le sous-répertoire **defaultStore** de la racine du magasin d'objets. Vous pouvez changer le sous-répertoire en configurant la variable de propriété **com.arjuna.ats.arjuna.objectstore.localOSRoot**.

1.3. DÉMARQUER LES TRANSACTIONS

L'API JBossJTA est constitué de trois éléments :

- Une interface de démarcation des transactions d'applications de haut niveau
- Une interface de gestion des transactions de haut niveau destinées au serveur d'applications
- et un mappage Java standard du protocole XA X/Open destiné au gestionnaire de ressources transactionnelles

Toutes les classes et les interfaces JTA se situent dans le package javax.transaction, et les implémentations JBossJTA correspondantes dans le package com.arjuna.ats.jta.

1.3.1. `UserTransaction`

L'interface **`UserTransaction`** permet aux applications de surveiller les limites des transactions.

Vous pouvez obtenir des implémentations de **`UserTransaction`** via JNDI.

Exemple 1.2. Contrôle des transactions

```
// Initialize the context and get UserTransaction
InitialContext ic = new InitialContext();
UserTransaction utx = ic.lookup("java:comp/UserTransaction")
// start transaction work..
utx.begin();
.. do work
utx.commit();
```

1.3.2. `TransactionManager`

L'interface **`TransactionManager`** permet au serveur d'applications de surveiller les limites des transactions pour l'application en cours.

Vous pourrez obtenir des implémentations du **`TransactionManager`** via JNDI.

```
// Initialize the context and get the TransactionManager
InitialContext ic = new InitialContext();
TransactinoManager utm = ic.lookup("java:/TransactionManager")
```

1.3.3. `Transaction`

L'interface **`Transaction`** permet aux transactions d'avoir lieu sur la transaction associée à l'objet cible. Chaque transaction de haut niveau est associée à un objet de **`Transaction`** quand la transaction est créée. L'objet **`Transaction`** a plusieurs utilisations possibles, comme l'explique Utilisations possibles de l'interface **`Transaction`**.

Utilisations possibles de l'interface `Transaction`

- Inscrit les ressources transactionnelles utilisées par l'application.

- Inscrivez-vous pour les rappels de synchronisation de transaction.
- Valider ou annuler la transaction.
- Obtenez le statut de la transaction.

Vous pouvez obtenir un objet **`Transaction`** en invoquant la méthode **`getTransaction`** de l'interface **`TransactionManager`**, comme le montre Exemple 1.3, « Obtenir une transaction ».

Exemple 1.3. Obtenir une transaction

```
Transaction txObj = TransactionManager.getTransaction();
```

1.4. IMPLÉMENTATIONS JTA LOCALES VERSUS DISTRIBUÉES

Vous devrez vous rapporter aux spécifications JTS/OTS pour la propagation de transactions dans les gestionnaires de transactions afin d'assurer l'interopérabilité entre les applications JTA.

Procédure 1.1. Sélection de l'implémentation JTA locale

1. Définir la propriété om.arjuna.ats.jta.jtaTMImplementation à **`com.arjuna.ats.internal.jta.transaction.arjunacore.TransactionManagerImple`**.
2. Définir com.arjuna.ats.jta.jtaUTImplementation à **`com.arjuna.ats.internal.jta.transaction.arjunacore.UserTransactionImple`**.

Procédure 1.2. Sélectionner l'implémentation JTA distribuée

1. Définir la propriété com.arjuna.ats.jta.jtaTMImplementation à **`com.arjuna.ats.internal.jta.transaction.jts.TransactionManagerImple`**.
2. Définir la propriété com.arjuna.ats.jta.jtaUTImplementation à **`com.arjuna.ats.internal.jta.transaction.jts.UserTransactionImple`**.

1.5. JDBC ET TRANSACTIONS

JBossJTA supporte la construction d'applications transactionnelles distribuées et locales à la fois, qui accèdent à la base de données par les API JDBC 2.0. JDBC 2.0 supporte *two-phase commit* (validation des transactions en deux temps), et ressemble au standard XA X/Open. Le support JDBC 2.0 se trouve dans le package com.arjuna.ats.jdbc.

JBossJTA intègre des connexions JDBC dans les transactions en fournissant des pilotes JDBC transactionnels par lesquels toutes les interactions ont lieu. Ces pilotes interceptent toutes les invocations et s'assurer qu'elles sont enregistrées avec et conduites par les opérations qui conviennent. Il y a un seul type de pilote transactionnel par l'intermédiaire duquel un conducteur JDBC conducteur peut être piloté. Ce pilote est com.arjuna.ats.jdbc.TransactionalDriver, et il implémente l'interface **`java.sql.Driver`**.

Vous pouvez établir la connexion par la méthode **`java.sql.DriverManager.getConnection`**. Après avoir établie la connexion, JBossJTA surveille toutes les opérations. Vous pouvez utiliser ces connexions de la même façon que n'importe quelle autre connexion de pilote JDBC.

Les connexions de JBossJTA peuvent être utilisées dans plusieurs transactions à la fois. Des threads différents, avec différentes notions de la transaction en cours, peuvent utiliser la même connexion JDBC. JBossJTA effectue la connexion en commun pour chaque transaction au sein de la connexion JDBC. Bien que plusieurs threads peuvent utiliser la même instance de la connexion JDBC, en interne, une instance de connexion différente peut être utilisée pour chaque transaction. À l'exception de la méthode **`close`**, toutes les opérations sur la connexion au niveau application sont effectuées uniquement sur cette connexion de transaction spécifique.

JBossJTA enregistre automatiquement la connexion du pilote JDBC avec la transaction via une ressource appropriée. À la fin de la transaction, cette ressource s'engage ou annule toutes les modifications apportées à la base de données sous-jacente, via des appels appropriés sur le pilote JDBC.

1.6. OPTIONS CONFIGURABLES

Options configurables importantes montre les fonctionnalités de configuration les plus importantes, avec les valeurs par défaut ou autres valeurs possibles. Pour plus d'informations à ce sujet, veuillez consulter *Transactions Development Guide*.

Options configurables importantes

com.arjuna.ats.jta.supportSubtransactions

Valeurs possibles

1. Oui (par défaut)
2. Non

com.arjuna.ats.jta.jtaTMImplementation

Valeurs possibles

1. com.arjuna.ats.internal.jta.transaction.arjunacore.TransactionManagerImple
2. com.arjuna.ats.internal.jta.transaction.jts.TransactionManagerImple

com.arjuna.ats.jta.jtaUTImplementation

Valeurs possibles

1. com.arjuna.ats.internal.jta.transaction.arjunacore.UserTransactionImple
2. com.arjuna.ats.internal.jta.transaction.jts.UserTransactionImple

com.arjuna.ats.jta.xaBackoffPeriod

Valeurs possibles

1. Heures en secondes

com.arjuna.ats.jdbc.isolationLevel

Valeurs possibles

1. N'importe quel niveau d'isolation JDBC supporté

CHAPITRE 2. DÉMARRER AVEC JTS / OTS

Ce chapitre explique les fonctionnalités clé nécessaires pour construire une application *OTS (Object Transaction Service)* de base par les interfaces OTS brutes, définies par la spécificaiton *Object Management Group (OMG)*. Ce travail est axé sur les détails d'implémentation. Voir le guide *Transactions Development Guide* pour un aperçu conceptuel.

2.1. STRUCTURE DU PACKAGE

Tableau 2.1. Packages importants pour créer les applications OTS

Package	Description
com.arjuna.orbportability	ce package contient les classes qui constituent la bibliothèque de portabilité ORB et les autres classes d'utilitaires utiles.
org.omg.CosTransactions	ce package contient les classes qui constitueront le module CosTransactions.idl.
com.arjuna.ats.jts	ce package contient les implémentations de JBoss Transaction Service du JTS ou JTA.
com.arjuna.ats.arjuna	ce package contient des classes supplémentaires utiles à l'implémentation JBoss Transaction Service du JTS.
com.arjuna.ats.jta	ce package contient le support d'implémentation JTA distant ou local.
com.arjuna.ats.jdbc	ce package contient le support JDBC 2.0 transactionnel.

2.2. CONFIGURATION DES PROPRIÉTÉS

Vous pouvez configurer JBoss Transaction Services en cours d'exécution, en paramétrant divers attributs de propriétés, qui sont décrits dans les sections suivantes. Vous pouvez fournir ces attributs en cours d'exécution sur la ligne de commande. Cependant, il est souvent plus pratique de les spécifier à travers un fichier de propriétés **`jbossts-properties.xml`**, qui peut se trouver dans n'importe quelle location spécifiée, dans Locations possibles du fichier **`jbossts-properties.xml`**, dans l'ordre de recherche.

Locations possibles du fichier `jbossts-properties.xml`

1. Le répertoire de travail en cours.
2. Le répertoire d'accueil d'un utilisateur en cours.
3. Le **`CLASSPATH`**, par la méthode **`getResource`**.

Quand le fichier de propriétés a été identifié, toutes les entrées qui s'y trouvent sont ajoutées aux propriétés du système, et remplacent les valeurs par défaut. Vous pouvez spécifier d'autres propriétés qui ne sont pas spécifiques au Service de transactions.

2.3. DÉMARRER ET STOPPER LE ORB ET BOA/POA

BOA se réfère au *Basic Object Adapter*, et POA se réfère au *Portable Object Adapter*.

JBoss Transaction Service doit être correctement initialisé avant la création d'un objet d'application. Pour garantir cela, vous devez utiliser la méthode **`initORB`**, et une des deux méthodes, soit **`initBOA`** soit **`initPOA`** appartenant à la classe **`ORBInterface`**, décrite dans le manuel de portabilité ORB. N'utilisez pas les méthodes **`ORB_init`**, **`BOA_init`**, ou **`create_POA`** fournies par l'ORB sous-jacent, car elles pourraient mener à des applications qui n'opèrent pas correctement.

Exemple 2.1. Initialisation ORB

```
public static void main (String[] args)
{
    ORBInterface.initORB(args, null);
    ORBInterface.initOA();
    . . .
};
```

`ORBInterface` Methods

`orb`

Renvoie des références à l'ORB

`boa`

Renvoie des références au BOA

`poa`

Renvoie des références au POA

`rootPoa`

Renvoie des références à la racine de POA

`shutdownOA`

Fermer le BOA. Exécuter ceci avant le **`shutdownORB`**, et avant de terminer l'application.

`shutdownORB`

Fermer l'ORB. Utiliser ceci après le **`shutdownOA`**. Exécuter ceci avant de terminer l'application.

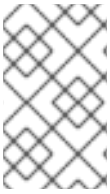
Utiliser les méthodes **`shutdownOA`** et **`shutdownORB`**, séquentiellement, avant de fermer l'application. Cela permet à JBoss Transaction Service de procéder aux routines de nettoyage habituelles. La routine **`shutdownOA`** ferme le BOA ou le POA, suivant l'ORB utilisé.

Exemple 2.2. Fermeture de l'ORB

```
public static void main (String[] args)
{
    . . .
```

```
    ORBInterface.shutdownOA();
    ORBInterface.shutdownORB();
};
```

N'utilisez pas davantage d'objets CORBA après le **`shutdown`**. Vous aurez besoin de réinitialiser le BOA/POA avant d'utiliser davantage d'objets CORBA.



NOTE

L'expression *Object Adapter* sera utilisée tout au long de ce guide pour faire référence au BOA ou au POA, l'un ou l'autre. Si possible, ce guide utilise les classes de Portabilité d'ORB pour masquer les différences entre POA et BOA.

2.4. PRÉCISER LA LOCATION DU MAGASIN D'OBJETS.

JBoss Transaction Services utilise un magasin d'objets pour enregistrer de façon persistante tous les résultats des transactions, pour les recouvrements suite à des défaillances. Vous pouvez spécifier la location du magasin d'objets par la propriété **`objectStoreDir`**.

Exemple 2.3. Spécifier le Magasin d'objets dans l'Exécution de l'application.

```
 java
Dcom.arjuna.ats.arjuna.objectstore.objectStoreDir=/var/tmp/ObjectStore
myprogram
```

Par défaut, le magasin d'objets de situe dans un répertoire qui se situe sous le répertoire d'exécution en cours. Dans la configuration par défaut, tous les états d'objets sont stockés dans le **`defaultStore`**. Cependant, ce sous-répertoire peut être modifié en définissant la variable de propriété com.arjuna.ats.arjuna.objectstore.localOSRoot.

2.5. PROPAGATION DE TRANSACTIONS IMPLICITES ET INTERPOSITION

Vous pouvez créer des transaction dans un domaine et les utiliser dans un autre. De ce fait, l'information sur une transaction, appelée *transaction context*, a besoin d'être propagée entre ces domaines.

Propagation du contexte de transaction

Propagation explicite

Une application passe des objets de contexte sous forme de paramètres explicites. Ces objets sont soit des instances de l'interface de **`Control`**, ou la structure de **`PropagationContext`**, et ils sont définis par le Service de propagation. Il est plus efficace d'utiliser la structure **`PropagationContext`** à la place de **`Control`**

Propagation implicite

Les demandes sur les objets sont implicitement associées à la transaction du client et partagent le contexte de la transaction du client. Le contexte est implicitement transmis aux objets, sans intervention directe du client.

Les objets OTS qui supportent l'interface de Control sont des objets standard CORBA. Quand l'interface est passée comme paramètre dans un appel d'opération vers un serveur distant, seule la référence à l'objet est passée. Tout opération effectuée par l'objet distant sur l'interface aura lieu sur l'objet réel.

Ce comportement peut imposer des sanctions importantes sur une application qui utilise fréquemment ces interfaces en raison des problèmes de surcharge des invocations distantes. Pour éviter cette surcharge, JBoss Transaction Service prend en charge l' *interposition*. En interposition, le serveur crée un objet local qui agit comme un proxy pour la transaction distante, et remplit toutes les demandes qui seraient normalement transmises à l'appelant. Cet objet local s'enregistre avec le coordonnateur de la transaction originale, afin qu'il puisse participer correctement à la fin de la transaction. Les coordonnateurs interposés forment efficacement une structure arborescente avec les coordonnateurs parents, comme le montre Figure 2.1, « Interposition ».

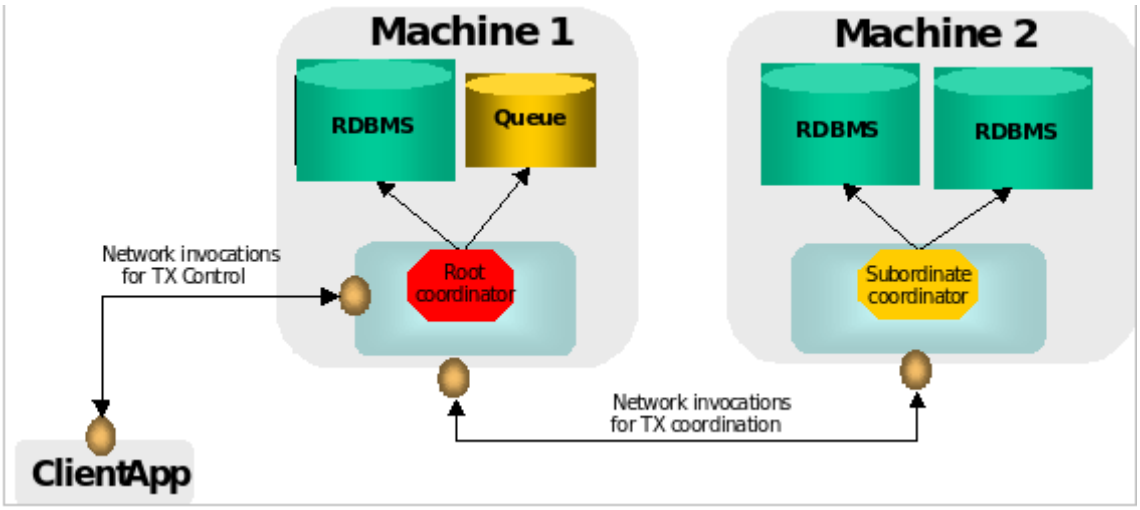


Figure 2.1. Interposition

NOTE

La propagation de transaction implicite n'implique pas qu'une interposition est également utilisée dans le serveur. Plutôt, l'interposition exige normalement une propagation implicite.

Si vous avez besoin d'une propagation de contexte implicite et une interposition, veillez à ce que JBoss Transaction Service soit correctement initialisé avant de créer un objet. Le client et le serveur ont besoin de se mettre d'accord pour savoir si la propagation ou l'interposition, ou aucune d'entre elles, est utilisée. La propagation de contexte implicite n'est possible que sur les ORB qui supportent les filtres et les intercepteurs, ou qui supportent l'interface **`CosTSPortability`**. JacORB et JDK miniORB procurent tous deux le support nécessaire.

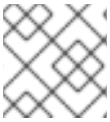
Activer la propagation

Propagation de contexte implicite

Définir la variable de propriété com.arjuna.ats.jts.contextPropMode à **`CONTEXT`**.

Interposition

Définir la variable de propriété com.arjuna.ats.jts.contextPropMode à **`INTERPOSITION`**.



NOTE

Pour utiliser l'API avancé de JBoss Transaction Service, vous devez utiliser l'interposition.

2.6. CURRENT

Vous pourrez obtenir le pseudo-objet **Current** de la classe **com.arjuna.ats.jts.OTSManager** par sa méthode **get_current**.

2.7. RÉSILIATION DE TRANSACTION

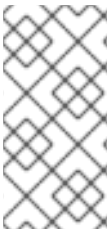
La durée pendant laquelle un **Control** peut accéder à une transaction résiliée est spécifique à l'implémentation. Dans JBoss Transaction Service, si vous utilisez le pseudo-objet **Current**, toutes les informations sur une transaction sont détruite à la fin. Pour cette raison, vous ne devez pas utiliser de référence de **Control** de la transaction suite à la validation ou la restauration de la transaction.

Toutefois, si vous résiliez explicitement la transaction, en utilisant l'interface **Terminator** les informations sur la transaction seront supprimées que lorsque toutes les références restantes à son sujet auront été détruites. Vous pourrez signal que les informations sur la transaction ne sont plus nécessaires, à l'aide de la méthode **destroyControl** de la classe de l'OTS, qui se trouve dans le package com.arjuna.CosTransactions. Une fois que le programme indique que les informations de transaction ne sont plus nécessaires, vous ne devrez plus utiliser de références de **Control** de la transaction.

2.8. USINE DE TRANSACTIONS

Par défaut, JBoss Transaction Service n'utilise pas de gestionnaire de transactions séparé quand il crée des transactions à travers l'interface **Current**. Chaque client de transaction possède son propre gestionnaire de transactions **TransactionFactory**, qui est co-chargé. Afin de pouvoir remplacer de comportement en cours d'exécution, définir la variable de propriété com.arjuna.ats.jts.transactionManager à **YES**. Pour exécuter l'usine de transactions, exécuter le script **start-transaction-service**, qui se trouve dans le répertoire ***ATS_ROOT*/bin**.

Current réussit normalement à localiser l'usine **CosServices.cfg** qui se trouve dans le répertoire ***$JBOSS_HOME*/etc**. Ce fichier ressemble au fichier **resolve_initial_references**, et est automatiquement créé ou mis à jour au démarrage de l'usine de transactions sur une machine précise. Ce fichier doit être copié localement sur chaque machine ayant besoin de partager la même usine de transactions.



NOTE

L'information sur **CosServices.cfg** fait réfèrence au nom et à la location par défaut du fichier de configuration. Pour changer le nom du fichier, utiliser la variable com.arjuna.orbportability.initialReferencesFile. Pour en changer la location, définir la variable com.arjuna.orbportability.initialReferencesRoot.

Exemple 2.4. Personnalisation du fichier de références initiales.

```
 java –Dcom.arjuna.orbportability.initialReferencesFile=ref –
Dcom.arjuna.orbportability.initialReferencesRoot=c:\\temp prog
```

Vous pouvez remplacer le mécanisme de location par défaut en définissant la variable de propriété com.arjuna.orbportability.resolveService par n'importe quel paramètre listé dans Paramètres ResolveService.

Paramètres ResolveService

CONFIGURATION_FILE

Le système utilise le fichier **`CosServices.cfg`**. Il s'agit du comportement par défaut.

NAME_SERVICE

JBoss Transaction Services tente d'utiliser un service de noms pour localiser l'usine de transactions. Si non supporté, une exception est lancée.

BIND_CONNECT

JBoss Transaction Services utilise les mécanismes de liaison spécifiques-ORB. Si non suporté, une exception sera lancée.

Si com.arjuna.orbportability.resolveService est spécifié quand l'usine de transactions est en cours d'exécution, l'usine va s'enregistrer elle-même par le mécanisme de résolution spécifié.

2.9. RECOVERY MANAGER

Vous avez besoin du sous-système de gestionnaire de recouvrement pour veiller à ce que toutes les transactions soient recouvrables en cas de défaillance. Pour démarrer le gestionnaire de recouvrement, exécuter le script **`start-recovery-manager`** dans ***`$ATS_ROOT`*`/bin`**.

CHAPITRE 3. COMMENCER AVEC WEB SERVICES TRANSACTIONS ET XTS

3.1. CONFIGURER LE COMPOSANT DE SERVICES WEB

pour davantage d'informations sur JBoss Transactions XTS, voir la section XTS du guide *Guide de Développement de Transactions*, qui est fourni avec la suite de documentation d'Enterprise Application Platform.

Tableau 3.1. Configuration des services Web

Propriété	Valeurs possibles
com.arjuna.orbportability.initialReferencesFile	CosServices.cfg
com.arjuna.orbportability.initialReferencesRoot	Le répertoire contient le fichier arjuna.properties.
ArjunaJTS_LicenceKey	Licence spécifique au système.
com.arjuna.orbportability.resolveService	CONFIGURATION_FILE NAME_SERVICE BIND_CONNECT
com.arjuna.ats.arjuna.objectstore.objectStoreDir	N'importe quelle location dans laquelle l'application peut écrire.
com.arjuna.ats.arjuna.objectstore.localOSRoot	defaultStore
PROPERTIES_FILE	arjuna.properties
com.arjuna.ats.arjuna.coordinator.asyncPrepare	YES/NO
com.arjuna.ats.arjuna.coordinator.asyncCommit	YES/NO
com.arjuna.ats.arjuna.coordinator.commitOnePhase	YES/NO
com.arjuna.ats.arjuna.coordinator.transactionSync	ON/OFF
com.arjuna.ats.arjuna.coordinator.enableStatistics	ON/OFF
com.arjuna.ats.jts.alwaysPropagateContext	YES/NO
com.arjuna.ats.jts.defaultTimeout	No timeout
com.arjuna.ats.jts.supportRollbackSync	YES/NO
com.arjuna.ats.jts.supportInterposedSynchronization	YES/NO

Propriété	Valeurs possibles
com.arjuna.ats.jts.supportSubtransactions	YES/NO
com.arjuna.ats.jts.checkedTransactions	YES/NO
com.arjuna.ats.jts.transactionManager	YES/NO
com.arjuna.ats.jts.needTranContext	YES/NO
com.arjuna.ats.arjuna.coordinator.txReaperTimeout	120000000 microseconds
com.arjuna.ats.arjuna.coordinator.txReaperMode	NORMAL DYNAMIC
com.arjuna.ats.jts.contextPropMode	NONE CONTEXT INTERPOSITION

ANNEXE A. HISTORIQUE DE RÉVISION

Version 5.1.0-0.35.400 Rebuild with publican 4.0.0	**2013-10-31**	**Rüdiger Landmann**
Version 5.1.0-0.35 Fix diacritics	**August 7 2012**	**Ruediger Landmann**
Version 5.1.0-0.33 Rebuild for Publican 3.0	**2012-07-18**	**Anthony Towns**
Version 5.1.0-100 Traduction française	**Mon Jul 18 2011**	**Corina Roe**