WebSphere® eXtreme Scale Version 7.1
Guide de programmation

WebSphere eXtreme Scale - Guide de programmation

IBM

août 2010

LE PRESENT DOCUMENT EST LIVRE EN L'ETAT SANS AUCUNE GARANTIE EXPLICITE OU IMPLICITE. IBM DECLINE NOTAMMENT TOUTE RESPONSABILITE RELATIVE A CES INFORMATIONS EN CAS DE CONTREFACON AINSI QU'EN CAS DE DEFAUT D'APTITUDE A L'EXECUTION D'UN TRAVAIL DONNE.

Ce document est mis à jour périodiquement. Chaque nouvelle édition inclut les mises à jour. Les informations qui y sont fournies sont susceptibles d'être modifiées avant que les produits décrits ne deviennent eux-mêmes disponibles. En outre, il peut contenir des informations ou des références concernant certains produits, logiciels ou services non annoncés dans ce pays. Cela ne signifie cependant pas qu'ils y seront annoncés.

Pour plus de détails, pour toute demande d'ordre technique, ou pour obtenir des exemplaires de documents IBM, référez-vous aux documents d'annonce disponibles dans votre pays, ou adressez-vous à votre partenaire commercial.

Vous pouvez également consulter les serveurs Internet suivants :

- http://www.fr.ibm.com (serveur IBM en France)
- http://www.can.ibm.com (serveur IBM au Canada)
- http://www.ibm.com (serveur IBM aux Etats-Unis)

Compagnie IBM France
Direction Qualité
17, avenue de l'Europe
92275 Bois-Colombes Cedex

Table des matières

Figures

Tableaux

A propos de la *Guide de programmation*

La documentation de WebSphere eXtreme Scale inclut trois volumes qui fournissent les informations nécessaires pour utiliser, programmer et administrer le produit WebSphere eXtreme Scale.

Bibliothèque WebSphere eXtreme Scale

La bibliothèque WebSphere eXtreme Scale contient les documents suivants :

- Le *Guide d'administration* contient les informations nécessaires pour les administrateurs système et explique notamment comment planifier les déploiements d'application, planifier la capacité, installer et configurer le produit, démarrer et arrêter des serveurs, surveiller l'environnement et le sécuriser.
- Le *Guide de programmation* contient des informations destinées aux développeurs d'applications, sur la manière de développer des applications pour WebSphere eXtreme Scale à l'aide des informations d'API incluses.
- La *Présentation du produit* contient une vue de haut niveau des concepts de WebSphere eXtreme Scale, avec des scénarios d'utilisation et des tutoriels.

Pour télécharger les documents, accédez à la page de la bibliothèque de WebSphere eXtreme Scale.

Vous pouvez également accéder à ces informations dans le Centre de documentation de WebSphere eXtreme Scale.

A qui s'adresse ce document

Ce document est principalement destiné aux développeurs d'applications.

Structure de ce document

Ce document contient des informations sur les rubriques principales suivantes :

- Le **Chapitre 1** inclut des informations sur l'initiation à WebSphere eXtreme Scale.
- Le **Chapitre 2** inclut des informations sur la programmation de WebSphere eXtreme Scale.
- Le **Chapitre 3** inclut des informations sur l'accès aux données.
- Le **Chapitre 4** inclut des informations sur les plug-in et les API système.
- Le **Chapitre 5** inclut des informations sur l'intégration à l'infrastructure Spring.
- Le **Chapitre 6** inclut des informations sur l'API de sécurité.
- Le **Chapitre 7** inclut des informations sur l'API d'administration.
- Le **Chapitre 8** inclut des informations sur les considérations de performances.
- Le **Chapitre 9** inclut des informations sur l'identification et la résolution des problèmes.
- Le **Chapitre 10** inclut le glossaire du produit.

Obtention des mises à jour de ce document

Vous pouvez obtenir les mises à jour de ce document en téléchargeant la version la plus récente à partir de la page de la bibliothèque de WebSphere eXtreme Scale.

Comment envoyer vos commentaires

Contactez l'équipe chargée de la documentation. Avez-vous trouvé ce que vous recherchiez ? Ces informations étaient-elles précises et complètes ? Envoyez vos commentaires sur cette documentation par courrier électronique, à l'adresse wasdoc@us.ibm.com.

Chapitre 1. Initiation à WebSphere eXtreme Scale

Après avoir installé WebSphere eXtreme Scale dans un environnement autonome, utilisez les étapes suivantes pour vous initier à ses fonctions en tant que grille de données en mémoire.

L'installation autonome de WebSphere eXtreme Scale comprend un exemple que vous pouvez utiliser pour vérifier votre installation et pour constater comment une grille et un client eXtreme Scale peuvent être utilisés. L'exemple de l'initiation se trouve dans le répertoire *racine_installation*/ObjectGrid/gettingstarted.

Cet exemple offre une introduction rapide aux fonctionnalités et aux opérations de base de eXtreme Scale. L'exemple contient un interpréteur de commandes et des scripts par lots conçus pour commencer une simple grille sans efforts de personnalisation. En outre, un programme client contenant la source est fourni pour exécuter les fonctions CRUD (Create, Read, Update, Delete) dans cette grille de base.

Scripts et fonctions correspondantes

Cet exemple offre les quatre scripts suivants :

le script env.sh|bat est appelé par les autres scripts pour la définition de variables d'environnement requises. Il n'est normalement pas nécessaire de modifier ce script.

- UNIX Linux ./env.sh
- Windows env.bat

Le script runcat.sh|bat démarre le service de catalogue eXtreme Scale sur le système local.

- UNIX Linux ./runcat.sh
- Windows runcat.bat

Le script runcontainer.sh|bat démarre le serveur de conteneur. Vous pouvez exécuter ce script plusieurs fois en spécifiant des noms de serveurs uniques pour démarrer autant de conteneurs que vous le voulez. Ces instances peuvent interagir pour héberger des informations partitionnées et redondantes dans la grille.

- UNIX Linux ./runcontainer.sh *nom_serveur_unique*
- Windows runcontainer.bat *nom_serveur_unique*

Le script runclient.sh|bat exécute le client CRUD et démarre l'opération voulue.

- UNIX Linux ./runclient.sh *commande valeur1 valeur2*
- Windows runclient.sh *commande valeur1 valeur2*

Pour *commande*, utilisez l'une des options suivantes :

- Spécifiez i pour insérer *valeur2* dans la grille avec la clé *valeur1*
- Spécifiez u pour mettre à jour l'objet indexé par *valeur1* avec *value2*
- Spécifiez d pour supprimer l'objet indexé par *valeur1*

- Spécifiez `g` pour extraire et afficher l'objet indexé par *valeur1*

Remarque : Le fichier*racine_installation*`/ObjectGrid/ gettingstarted/src/ Client.java` est le programme client qui indique comment établir la connexion au serveur de catalogues, obtenir une instance ObjectGrid et utiliser l'API ObjectMap.

Etapes de base

Utilisez les étapes suivantes pour démarrer votre première grille et exécuter un client en vue d'interagir avec la grille.

1. Ouvrez une session terminal ou une fenêtre de ligne de commande.
2. La commande suivante permet d'accéder au répertoire `gettingstarted` :

   `cd` *racine_installation*`/ObjectGrid/gettingstarted`

   Remplacez *racine_installation* par le chemin d'accès au répertoire racine d'installation d'eXtreme Scale ou le chemin d'accès au fichier racine du répertoire d'essai *racine_installation* d'eXtreme Scale.
3. Exécutez le script suivant pour démarrer un processus de service de catalogue sur le système hôte local :
   - UNIX Linux `./runcat.sh`
   - Windows `runcat.bat`

   Le processus du service de catalogue s'exécute dans la fenêtre du terminal en cours.
4. Ouvrez une autre session terminal ou fenêtre de ligne de commande et exécutez la commande suivante pour démarrer une instance de serveur de conteneur :
   - UNIX Linux `./runcontainer.sh server0`
   - Windows `runcontainer.bat server0`

   Le serveur de conteneur s'exécute dans la fenêtre du terminal en cours. Vous pouvez répéter les étapes 5 et 6 si vous voulez démarrer plus d'instances de serveurs de conteneur pour prendre en charge la réplication.
5. Ouvrez une autre session terminal ou fenêtre de ligne de commande pour exécuter le client.
   - Ajoutez des données à la grille :
     - UNIX Linux `./runclient.sh i key1 helloWorld`
     - Windows `runclient.bat i key1 helloWorld`
   - Recherchez et affichez la valeur :
     - UNIX Linux `./runclient.sh g key1`
     - Windows `runclient.bat g key1`
   - Mettez la valeur à jour :
     - UNIX Linux `./runclient.sh u key1 goodbyeWorld`
     - Windows `runclient.bat u key1 goodbyeWorld`
   - Supprimez la valeur :
     - UNIX Linux `./runclient.sh d key1`
     - Windows `runclient.bat d key1`
6. Arrêtez le processus de service de catalogue et les serveurs de conteneur dans les fenêtres respectives en appuyant sur <ctrl+c>.

Définition d'une grille d'objets

L'exemple utilise les fichiers `objectgrid.xml` et `deployment.xml` disponibles dans le répertoire *racine_installation*`/ObjectGrid/gettingstarted/xml` pour démarrer un serveur de conteneur. Le fichier `objectgrid.xml` est le fichier XML du descripteur d'ObjectGrid et le fichier `deployment.xml` est le fichier XML du descripteur de la règle de déploiement de la grille d'objets. Les deux fichiers réunis définissent une topologie ObjectGrid répartie.

Fichier XML du descripteur d'ObjectGrid

Un fichier XML de descripteur d'ObjectGrid permet de définir la structure de la grille d'objets utilisée par l'application. Il contient une liste de configurations de mappes de sauvegarde. Ces mappes de sauvegarde constituent l'emplacement de stockage effectif des données en cache. L'exemple suivant présente un fichier d'exemple `objectgrid.xml`. Les premières lignes de ce fichier incluent l'en-tête requis de chaque fichier XML ObjectGrid. Ce fichier d'exemple définit la grille ObjectGrid avec les mappes de sauvegarde Map1 et Map2.

```
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">

    <objectGrids>
        <objectGrid name="Grid">
            <backingMap name="Map1" />
            <backingMap name="Map2" />
        </objectGrid>
    </objectGrids>

</objectGridConfig>
```

Fichier XML du descripteur de la règle de déploiement

Un fichier XML de descripteur de règle de déploiement est transmis à un serveur conteneur ObjectGrid lors du démarrage. La règle de déploiement doit être utilisée avec un fichier XML d'ObjectGrid et doit être compatible avec le fichier XML d'ObjectGrid qui lui est associé. Chaque élément objectgridDeployment de la règle de déploiement doit correspondre à un élément ObjectGrid de votre fichier XML d'ObjectGrid. Les éléments de la mappe de sauvegarde définis dans l'élément objectgridDeployment doivent être cohérents avec les mappes de sauvegarde contenues dans le fichier XML d'ObjectGrid. Chaque mappe de sauvegarde doit être référencée dans un seul et unique groupe de mappes.

Le fichier XML de descripteur de règle de déploiement est conçu pour être couplé avec le fichier XML d'ObjectGrid correspondant, le fichier `objectgrid.xml`. Dans l'exemple suivant, les premières lignes du fichier `deployment.xml` incluent l'en-tête requis de chaque fichier XML de règle de déploiement. Le fichier définit l'élément objectgridDeployment pour la grille ObjectGrid définie dans le fichier `objectgrid.xml`. Les mappes de sauvegarde Map1 et Map2 définies dans la grille ObjectGrid sont incluses dans le groupe de mappes mapSet pour lequel les attributs numberOfPartitions, minSyncReplicas et maxSyncReplicas sont configurés.

```
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy
 ../deploymentPolicy.xsd"
 xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">

    <objectgridDeployment objectgridName="Grid">
        <mapSet name="mapSet" numberOfPartitions="13" minSyncReplicas="0"
     maxSyncReplicas="1" >
```

```
            <map ref="Map1"/>
            <map ref="Map2"/>
        </mapSet>
    </objectgridDeployment>

</deploymentPolicy>
```

L'attribut numberOfPartitions de l'élément mapSet indique le nombre de partitions de l'élément mapSet. Il s'agit d'un attribut facultatif et sa valeur par défaut est égale à 1. Ce nombre doit être approprié pour la capacité anticipée de la grille.

L'attribut minSyncReplicas de l'élément mapSet vise à indiquer le nombre minimal de fragments répliques synchrones de chaque partition du groupe de mappes. Il s'agit d'un attribut facultatif et sa valeur par défaut est égale à 0. Le fragment primaire et le fragment réplique ne sont pas positionnés tant que le domaine ne peut pas prendre en charge le nombre minimal de fragments répliques synchrones. Pour prendre en charge la valeur minSyncReplicas, vous avez besoin d'un nombre de conteneurs égal à la valeur de minSyncReplicas plus un. Si le nombre de fragments répliques synchrones est inférieur à la valeur de minSyncReplicas, les transactions d'écrire ne sont plus autorisées pour cette partition.

L'attribut maxSyncReplicas de l'élément mapSet vise à indiquer le nombre maximal de fragments répliques synchrones de chaque partition du groupe de mappes. Il s'agit d'un attribut facultatif et sa valeur par défaut est égale à 0. Aucune autre réplique synchrone n'est placée pour une partition une fois qu'un domaine a atteint ce nombre de fragments répliques synchrones pour cette partition spécifique. L'ajout de conteneurs prenant en charge cette grille d'objets peut entraîner un nombre croissant de fragments répliques synchrones si la valeur maxSyncReplicas n'a pas déjà été atteinte. L'exemple définit la valeur maxSyncReplicas sur 1, ce qui signifie que le domaine place au maximum une réplique synchrone. Si vous démarrez plusieurs instances de serveurs de conteneur, seule une réplique synchrone sera placée dans une des instances de serveurs de conteneur.

Utilisation de la grille d'objets

Le fichier `Client.java` dans le répertoire *racine_installation*`/ObjectGrid/gettingstarted/src/` est le programme client qui illustre comment établir la connexion au serveur de catalogues, obtenir l'instance ObjectGrid et utiliser l'API ObjectMap.

Du point de vue d'une application client, l'utilisation de WebSphere eXtreme Scale englobe les étapes suivantes.

1. Connexion au service de catalogue via une instance ClientClusterContext.
2. Obtention d'une instance client ObjectGrid.
3. Obtention d'une instance Session.
4. Obtention d'une instance ObjectMap.
5. Utilisation des méthodes ObjectMap.

1. **Connexion au service de catalogue via une instance ClientClusterContext**

   Pour établir la connexion au serveur de catalogues, utilisez la méthode connect de l'API ObjectGridManager. La méthode connect utilisée nécessite uniquement le noeud final du serveur de catalogues dans le format `nomhôte:port`, tel que `localhost:2809`. Si la connexion au serveur de catalogues aboutit, la méthode connect renvoie une instance ClientClusterContext. L'instance ClientClusterContext est requise pour obtenir l'ObjectGrid à partir de l'API

ObjectGridManager. Le fragment de code suivant montre comment établir la connexion à un serveur de catalogues et comment obtenir une instance ClientClusterContext.

```
ClientClusterContext ccc = ObjectGridManagerFactory.getObjectGridManager().connect("localhost:2809", null, null);
```

2. **Obtention d'une instance ObjectGrid**

   Pour obtenir une instance ObjectGrid, utilisez la méthode getObjectGrid de l'API ObjectGridManager. La méthode getObjectGrid requiert l'instance ClientClusterContext et le nom de l'instance ObjectGrid. L'instance ClientClusterContext est obtenue pendant la connexion au serveur de catalogues. Le nom de l'ObjectGrid correspond à la grille `Grid` spécifiée dans le fichier `objectgrid.xml`. Le fragment de code suivant montre comment obtenir l'ObjectGrid en appelant la méthode getObjectGrid de l'API ObjectGridManager.

   ```
   ObjectGrid grid = ObjectGridManagerFactory.getObjectGridManager().getObjectGrid(ccc, "Grid");
   ```

3. **Obtention d'une instance Session**

   Vous pouvez obtenir une session de l'instance ObjectGrid obtenue. Une instance Session est requise pour obtenir l'instance ObjectMap et pour effectuer une démarcation de transaction. Le fragment de code suivant montre comment obtenir une instance Session en appelant la méthode getSession de l'API ObjectGrid.

   ```
   Session sess = grid.getSession();
   ```

4. **Obtention d'une instance ObjectMap**

   Après l'obtention d'une instance Session, vous pouvez obtenir une instance ObjectMap d'une instance Session en appelant la méthode getMap de l'API Session. Vous devez transmettre le nom de la mappe en tant que paramètre à la méthode getMap afin d'obtenir l'instance ObjectMap. Le fragment de code suivant montre comment obtenir l'instance ObjectMap en appelant la méthode getMap de l'API Session.

   ```
   ObjectMap map1 = sess.getMap("Map1");
   ```

5. **Utilisation des méthodes ObjectMap**

   Après l'obtention d'une instance ObjectMap, vous pouvez utiliser l'API ObjectMap. Gardez à l'esprit que l'interface ObjectMap est une mappe transactionnelle et requiert une démarcation de transaction à l'aide des méthodes begin et commit de l'API Session. Si aucune démarcation de transaction explicite n'a lieu dans l'application, les opérations ObjectMap s'exécutent sans transactions de validation automatique.

   Le fragment de code suivant montre comment utiliser l'API ObjectMap avec des transactions de validation automatique.

   ```
   map1.insert(key1, value1);
   ```

   Le fragment de code suivant montre comment utiliser l'API ObjectMap avec une démarcation de transaction explicite.

   ```
   sess.begin();
   map1.insert(key1, value1);
   sess.commit();
   ```

Informations supplémentaires

Cet exemple illustre comment démarrer le serveur de catalogues et le serveur conteneur et comment utiliser l'API ObjectMap dans un environnement autonome. Vous pouvez également faire appel à l'API EntityManager.

Dans un environnement WebSphere Application Server dans lequel WebSphere eXtreme Scale est installé ou activé, une topologie de connexion en réseau constitue

le scénario le plus fréquent. Dans une topologie de connexion en réseau, le serveur de catalogues est hébergé dans le processus du gestionnaire de déploiement WebSphere Application Server et chaque instance WebSphere Application Server héberge automatiquement un serveur eXtreme Scale. Pour que la grille d'objets soit disponible automatiquement, il suffit que les applications Java™ Platform, Enterprise Edition contiennent le fichier XML du descripteur d'ObjectGrid et le fichier XML du descripteur de règle de déploiement ObjectGrid dans le répertoire `META-INF`. Une application peut ensuite se connecter à un serveur de catalogues disponible en local et obtenir une instance ObjectGrid.

Chapitre 2. Préparation du développement d'applications

Configurez votre environnement de développement et apprenez où trouver les détails concernant les interfaces de programmation utilisables.

Interfaces de programmation de WebSphere eXtreme Scale

WebSphere eXtreme Scale fournit un certain nombre de fonctionnalités accessibles par programmation à l'aide du langage Java via des interfaces de programmation d'applications (API) et des interfaces de programmation système.

API WebSphere eXtreme Scale

Lorsqu'il s'agit d'utiliser des API eXtreme Scale, il convient de distinguer soigneusement les opérations transactionnelles et celles qui ne le sont pas. Une opération transactionnelle est une opération qui s'effectue au sein d'une transaction. ObjectMap, EntityManager, Query et DataGrid sont des API transactionnelles qui sont contenues dans l'objet Session, lequel est un conteneur transactionnel. Les opérations non transactionnelles n'ont rien à voir avec une transaction. C'est le cas, par exemple, des opérations de configuration.

ObjectGrid, BackingMap, et les API de plug-in sont non transactionnels. ObjectGrid, BackingMap et d'autres API de configuration rentrent dans la catégorie des API ObjectGrid Core. Les plug-in permettent de personnaliser le cache pour réaliser les fonctions que l'on souhaite et ils rentrent dans la catégorie des API de programmation système. Dans eXtreme Scale,un plug-in est un composant qui fournit un certain type de fonctionnalité aux composants connectables d'eXtreme Scale (ObjectGrid et BackingMap). Une fonctionnalité représente une fonction ou une caractéristique spécifiques d'un composant eXtreme Scale (ObjectGrid, Session, BackingMap, ObjectMap, etc.). En général, les fonctionnalités sont configurables à l'aide d'API de configuration. Il peut arriver que des plug-in soient pré-intégrés mais vous pouvez très bien être amené à développer vos propres plug-in dans certains cas.

Vous pouvez normalement configurer ObjectGrid et BackingMap en fonction des besoins de votre application. Si l'application a des besoins spéciaux, vous pouvez envisager d'utiliser des plug-in spécialisés. WebSphere eXtreme Scale a des plug-in pré-intégrés qui répondent peut-être à vos besoins. Si, par exemple, vous avez besoin d'un modèle de réplication entre homologues entre deux instances ObjectGrid locales ou deux grilles eXtreme Scale réparties, vous pouvez très bien utiliser le plug-in pré-intégré JMSObjectGridEventListener. Si aucun des plug-in pré-intégrés n'arrive à résoudre vos problèmes métier, reportez-vous à l'API de programmation système pour produire vos propres plug-in.

ObjectMap est une API simple basée sur une mappe. Si les objets mis en cache sont simples et qu'aucune relation n'intervient, l'API ObjectMap est parfaitement indiquée pour votre application. Si les objets entretiennent des relations, il est conseillé d'utiliser l'API EntityManager qui prend en charge les relations de type graphes d'objets.

Query est un puissant mécanisme de recherche de données dans l'ObjectGrid. Session et EntityManager fournissent tous deux les fonctions classiques de requêtes.

L'API DataGrid offre des fonctionnalités puissantes de calcul dans un environnement eXtreme Scale réparti impliquant un grand nombre de machines, de fragments répliques et de partitions. Les applications peuvent exécuter une logique métier parallèlement à tous les noeuds d'un environnement eXtreme Scale réparti. L'application peut obtenir l'API DataGrid via l'API ObjectMap.

7.0.0.0 FIX 2+ Le service de données WebSphere eXtreme Scale REST est un service HTTP Java compatible avec Microsoft® WCF Data Services (ex-ADO.NET Data Services) et il implémente Open Data Protocol (OData). Le service de données REST autorise n'importe quel client HTTP à accéder à une grille eXtreme Scale. Il est compatible avec la prise en charge de WCF Data Services, qui est fournie avec Microsoft .NET Framework 3.5 SP1. Il est possible de développer des applications compatibles REST avec les outils fournis par Microsoft Visual Studio 2008 SP1. Pour plus de détails, voir le manuel eXtreme Scale REST Data Service — Guide d'utilisation.

Considérations liées au chargeur de classe et au chemin d'accès aux classes

Etant donné que eXtreme Scale stocke par défaut des objets Java dans le cache, vous devez définir des classes dans le chemin d'accès aux classes lorsque les données font l'objet d'un accès.

Plus précisément, le chemin d'accès aux classes des processus eXtreme Scale client et conteneur doivent inclure les classes ou les fichiers JAR lors du démarrage du processus. Lors de la conception d'une application à utiliser avec eXtreme Scale, séparez la logique métier des objets données persistantes.

Pour plus d'informations, consultez la rubrique Chargement de classes du WebSphere Application Server Centre de documentation de Network Deployment.

Pour obtenir des informations sur un paramètre Spring Framework, consultez la section relative à l'intégration à Spring Framework du manuel *Guide de programmation*.

Pour plus d'informations sur les paramètres associés à l'utilisation de l'agent d'instrumentation de WebSphere eXtreme Scale, consultez la rubrique relative à l'agent d'instrumentation du manuel *Guide de programmation*.

Configuration d'un environnement de développement

Configurez un environnement de développement intégré Eclipse pour générer et exécuter une application Java SE avec eXtreme Scale.

Procédure

- Configurez Eclipse pour générer et exécuter une application Java SE avec eXtreme Scale.
  1. Définissez une bibliothèque utilisateur pour permettre à votre application de référencer des API eXtreme Scale.
     a. Dans votre environnement Eclipse ou IBM® Rational Application Developer, cliquez sur **Fenêtre > Préférences**.
     b. Développez la branche **Java > Chemin de compilation** et sélectionnez **Bibliothèques utilisateur**. Cliquez sur **Nouveau**.

c. Sélectionnez la bibliothèque utilisateur eXtreme Scale. Cliquez sur **Ajouter des fichiers JAR**.
   1) Recherchez et sélectionnez les fichiers `objectgrid.jar` et `cglib.jar` dans le répertoire `racine_wxs/lib`. Cliquez sur **OK**.
   2) Pour inclure le Javadoc des API ObjectGrid, sélectionnez l'emplacement des Javadoc correspondant au fichier `objectgrid.jar` que vous avez ajouté à l'étape précédente. Cliquez sur **Editer**. Dans la zone du chemin du Javadoc, entrez l'adresse Web suivante :
      `http://www.ibm.com/developerworks/wikis/extremescale/docs/api/`
d. Cliquez sur **OK** pour appliquer les paramètres et refermez la fenêtre des préférences.

Les bibliothèques eXtreme Scale se trouvent à présent dans le chemin de compilation du projet.

2. Ajoutez la bibliothèque utilisateur à votre projet Java.
   a. Dans l'Explorateur de packages, cliquez sur le projet avec le bouton droit de la souris et sélectionnez **Propriétés**.
   b. Sélectionnez l'onglet **Bibliothèques**.
   c. Cliquez sur **Ajouter une bibliothèque**.
   d. Sélectionnez **Bibliothèque utilisateur**. Cliquez sur **Suivant**.
   e. Sélectionnez la bibliothèque utilisateur eXtreme Scale que vous avez précédemment configurée.
   f. Cliquez sur **OK** pour appliquer les modifications et refermez la fenêtre des propriétés.

- Exécutez une application Java SE avec eXtreme Scale avec Eclipse. Créez une configuration d'exécution pour exécuter votre application.
  1. Configurez Eclipse pour générer et exécuter une application Java SE avec eXtreme Scale. Dans le menu **Exécuter**, sélectionnez **Configurations d'exécution**.
  2. Cliquez avec le bouton droit de la souris sur la catégorie Application Java et sélectionnez **Nouvelle**.
  3. Sélectionnez la nouvelle configuration d'exécution, *Nouvelle_configuration*.
  4. Configurez le profil.
     – **Projet** (dans l'onglet principal) : *nom_votre_projet*
     – **Classe principale** (dans l'onglet principal) : *votre_classe_principale*
     – **Arguments VM** (dans l'onglet Arguments) :
       -Djava.endorsed.dirs=`racine_wxs/lib/endorsed`

     Des problèmes surgissent fréquemment avec les **arguments VM** car le chemin de `java.endorsed.dirs` doit être un chemin absolu sans variables ni raccourcis.

     D'autres problèmes usuels impliquent ORB (Object Request Broker). Il pourra vous arriver d'avoir l'erreur suivante. Pour plus d'informations, voir Configuration d'un ORB personnalisé.

     ```
     Caused by: java.lang.RuntimeException: The ORB that comes
     with the Sun Java implementation does not work with
     ObjectGrid at this time.
     ```

     Si les fichiers `objectGrid.xml` ou `deployment.xml` ne sont pas accessibles à l'application, vous risquez de rencontrer l'erreur suivante :

     ```
     Exception in thread "P=211046:O=0:CT"
     com.ibm.websphere.objectgrid.

     ObjectGridRuntimeException:
     ```

```
Cannot start OG container at
    Client.startTestServer(Client.java:161) at Client.
    main(Client.java:82) Caused by: java.lang.IllegalArgumentException:
    The objectGridXML must not be null at com.ibm.websphere.objectgrid.
    deployment.DeploymentPolicyFactory.createDeploymentPolicy
     (DeploymentPolicyFactory.java:55) at Client.startTestServer(Client.
    java:154) .. 1 more
```

5. Cliquez sur **Appliquer** et refermez la fenêtre ou cliquez sur **Exécuter**.

Exécution dans Rational Application Developer d'une application client ou serveur WebSphere eXtreme Scale avec Apache Tomcat

Qu'il s'agisse d'une application client ou d'une application serveur, vous devrez procéder fondamentalement de la même manière pour exécuter avec Apache Tomcat cette application dans Rational Application Developer. Dans le cas d'une application client, vous voudrez configurer et exécuter une application Web pour qu'elle utilise un client WebSphere eXtreme Scale dans Rational Application Developer. Procédez comme indiqué ci-après pour créer un projet Web pour l'exécution d'un service de catalogue ou d'un conteneur WebSphere eXtreme Scale. Dans le cas d'une application serveur, vous voudrez activer une application Java EE dans l'interface Rational Application Developer avec une installation autonome de WebSphere eXtreme Scale. Procédez comme indiqué ci-après pour configurer un projet d'application Java EE pour utiliser la bibliothèque de clients de WebSphere eXtreme Scale.

Avant de commencer

Installez la version d'évaluation de WebSphere eXtreme Scale ou le produit complet.

- Installez la version autonome de WebSphere eXtreme Scale 7.1.
- Téléchargez et extrayez la version d'évaluation de WebSphere eXtreme Scale.
- Installez Apache Tomcat 6.0 ou plus récent.
- Installez Rational Application Developer et créez une application Web Java EE.

Procédure

1. Ajoutez une bibliothèque d'exécution WebSphere eXtreme Scale à votre chemin de compilation Java EE.

   Application client Dans ce scénario, vous voudrez configurer et exécuter une application Web pour qu'elle utilise un client WebSphere eXtreme Scale dans Rational Application Developer.

   a. Cliquez sur **Fenêtre** → **Préférences** → **Java** → **Chemin de compilation** → **Bibliothèques utilisateur**. Cliquez sur **Nouveau**.

   b. Entrez `eXtremeScaleClient` comme **nom de bibliothèque utilisateur** et cliquez sur **OK**.

   c. Cliquez sur **Ajouter des fichiers JAR...** et allez au fichier `base_wxs/lib/ogclient.jar`. Cliquez sur **Ouvrir**.

   d. Facultatif : (Facultatif) Pour ajouter un Javadoc, sélectionnez l'emplacement de ce Javadoc et cliquez sur **Editer...**. Dans le chemin d'accès au Javadoc, vous pouvez entrer l'URL de la documentation des API ou télécharger cette documentation.

      - Pour utiliser la documentation en ligne, entrez `http://www.ibm.com/developerworks/wikis/extremescale/docs/api/` dans le chemin d'accès au Javadoc.

- Pour télécharger la documentation, allez à la WebSphere eXtreme Scalepage de téléchargement de la documentation des API. Comme chemin d'accès au Javadoc, entrez l'emplacement de votre disque dur où vous avez téléchargé la documentation.

e. Cliquez sur **OK**.

f. Cliquez sur **OK** pour refermer la boîte de dialogue Bibliothèques utilisateur.

g. Cliquez sur **Projet** → **Propriétés**.

h. Cliquez sur **Chemin de compilation Java**.

i. Cliquez sur **Ajouter une bibliothèque**.

j. Sélectionnez **Bibliothèque utilisateur**. Cliquez sur **Suivant**.

k. Cochez la bibliothèque **eXtremeScaleClient** et cliquez sur **Terminer**.

l. Cliquez sur **OK** pour refermer la boîte de dialogue des propriétés du projet.

Application serveur Dans ce scénario, vous voudrez configurer et exécuter une application Web pour qu'elle exécute dans Rational Application Developer un serveur WebSphere eXtreme Scale imbriqué.

a. Cliquez sur **Fenêtre** → **Préférences** → **Java** → **Chemin de compilation** → **Bibliothèques utilisateur**. Cliquez sur **Nouveau**.

b. Entrez `eXtremeScale` comme **nom de bibliothèque utilisateur** et cliquez sur **OK**.

c. Cliquez sur **Ajouter des fichiers JAR...** et allez au fichier base_wxs/lib/objectgrid.jar. Cliquez sur Ouvrir.

d. (Facultatif) Pour ajouter un Javadoc, sélectionnez l'emplacement de ce Javadoc et cliquez sur **Editer...**. Comme chemin d'accès au Javadoc, entrez `http://www.ibm.com/developerworks/wikis/extremescale/docs/api/`.

e. Cliquez sur **OK**.

f. Cliquez sur **OK** pour refermer la boîte de dialogue Bibliothèques utilisateur.

g. Cliquez sur **Projet** → **Propriétés**.

h. Cliquez sur **Chemin de compilation Java**.

i. Cliquez sur **Ajouter une bibliothèque**.

j. Sélectionnez **Bibliothèque utilisateur**. Cliquez sur **Suivant**.

k. Cochez la bibliothèque **eXtremeScaleClient** et cliquez sur **Terminer**.

l. Cliquez sur **OK** pour refermer la boîte de dialogue des propriétés du projet.

2. Définissez le serveur Tomcat pour notre projet.

a. Vérifiez que vous vous trouvez bien dans la perspective J2EE et cliquez sur l'onglet **Serveurs** dans le volet du bas. Vous pouvez également cliquer sur **Fenêtre** → **Afficher la vue** → **Serveurs**.

b. Cliquez avec le bouton droit de la souris dans le volet Serveurs et sélectionnez **Nouveau** → **Serveur**.

c. Sélectionnez **Apache, Tomcat v6.0 Server**. Cliquez sur **Suivant**.

d. Cliquez sur **Parcourir...** et sélectionnez racine_tomcat. Cliquez sur **OK**.

e. Cliquez sur **Suivant**.

f. Sélectionnez votre application Java EE dans le volet Disponibles à gauche et cliquez sur **Ajouter >** pour la faire passer dans le volet Configurées à droite sur le serveur, puis cliquez sur **Terminer**.

3. Résolvez les éventuelles erreurs restantes pour le projet. Toutes ces erreurs doivent apparaître dans le volet Problèmes. Si ce n'est pas le cas, procéder comme suit :

a. Cliquez sur **Projet** → **Nettoyer** → *nom_projet*. Cliquez sur **OK**. Compilez le projet.
b. Avec le bouton droit de la souris, cliquez sur le projet Java EE qui vous intéresse et sélectionnez **Chemin de compilation** → **Configurer le chemin de compilation**.
c. Cliquez sur l'onglet **Bibliothèques**. Vérifiez que le chemin est correctement configuré :
   - **Applications clients :** vérifiez qu'Apache Tomcat, eXtremeScaleClient et Java 1.5 JRE sont bien dans le chemin.
   - **Applications serveurs :** vérifiez qu'Apache Tomcat, eXtremeScale et Java 1.5 JRE sont bien dans le chemin.

4. Créez une configuration d'exécution pour exécuter votre application.
   a. Dans le menu **Exécuter**, sélectionnez **Configurations d'exécution**.
   b. Cliquez avec le bouton droit de la souris sur la catégorie Application Java et sélectionnez **Nouvelle**.
   c. Sélectionnez la nouvelle configuration d'exécution, *Nouvelle_configuration*.
   d. Configurez le profil.
      - **Projet** (dans l'onglet principal) : *nom_votre_projet*
      - **Classe principale** (dans l'onglet principal) : *votre_classe_principale*
      - **Arguments VM** (dans l'onglet Arguments) : `-Djava.endorsed.dirs=racine_wxs/lib/endorsed`

      Des problèmes surgissent fréquemment avec les **arguments VM** car le chemin de `java.endorsed.dirs` doit être un chemin absolu sans variables ni raccourcis.

      D'autres problèmes usuels impliquent ORB (Object Request Broker). Il pourra vous arriver d'avoir l'erreur suivante. Pour plus d'informations, voir Configuration d'un ORB personnalisé.

      ```
      Caused by: java.lang.RuntimeException: The ORB that comes with the Sun
      Java implementation does not work with ObjectGrid at this time.
      ```

      Si les fichiers `objectGrid.xml` ou `deployment.xml` ne sont pas accessibles à l'application, vous risquez de rencontrer l'erreur suivante :

      ```
      Exception in thread "P=211046:O=0:CT" com.ibm.websphere.objectgrid.
      ObjectGridRuntimeException:
      Cannot start OG container
       at Client.startTestServer(Client.java:161)
       at Client.main(Client.java:82)
      Caused by: java.lang.IllegalArgumentException: The objectGridXML must not
      be null at com.ibm.websphere.objectgrid.deployment.DeploymentPolicyFactory.
      createDeploymentPolicy
        (DeploymentPolicyFactory.java:55)
       at Client.startTestServer(Client.java:154)
       ... 1 more
      ```

5. Cliquez sur **Appliquer** et refermez la fenêtre ou cliquez sur **Exécuter**.

Que faire ensuite

Si vous avez configuré et exécuté une application Web pour qu'elle utilise un client WebSphere eXtreme Scale dans Rational Application Developer, c'est le moment de développer un servlet qui utilise les API WebSphere eXtreme Scale pour stocker et extraire des données d'une grille WebSphere eXtreme Scale distante.

Si vous avez activé une application Java EE dans l'interface Rational Application Developer avec une installation autonome de WebSphere eXtreme Scale, c'est le

moment de développer un servlet qui utilise les API système WebSphere eXtreme Scale pour démarrer et arrêter les services de catalogue.

Exécution dans Rational Application Developer d'une application client ou serveur intégrée avec WebSphere Application Server

Configurez et exécutez dans Rational Application Developer une application Java EE avec un client ou un serveur WebSphere eXtreme Scale avec l'environnement d'exécution WebSphere Application Server imbriqué. Si vous configurez un serveur, le démarrage de WebSphere Application Server démarrera automatiquement WebSphere eXtreme Scale.

Avant de commencer

La procédure qui suit concerne WebSphere Application Server version 7.0 avec Rational Application Developer version 7.5. Elle peut varier si l'on utilise des versions différentes de ces produits.

Installez Rational Application Developer avec les extensions WebSphere Application Server d'environnement de test.

Installez le client ou le serveur WebSphere eXtreme Scale dans l'environnement de test WebSphere Application Server version 7.0 dans le répertoire *base_rad*\runtimes\base_v7.

Procédure

1. Définissez le serveur eXtreme Scale qui est intégré à WebSphere Application Server pour votre projet.
   a. Dans la perspective J2EE, cliquez sur **Fenêtre > Afficher la vue > Serveurs.**
   b. Cliquez avec le bouton droit de la souris dans le volet **Serveurs**. Sélectionnez **Nouveau > Serveur**.
   c. Sélectionnez **IBM WebSphere Application Server v7.0**. Cliquez sur Suivant.
   d. Sélectionnez le profil à utiliser. Le profil par défaut est was70profile1.
   e. Entrez le nom du serveur. Le nom par défaut est server1.
   f. Cliquez sur **Suivant**.
   g. Sélectionnez votre application Java EE dans le volet **Disponibles**. Cliquez sur **Ajouter >** pour la faire passer dans le volet **Configurées** sur le serveur. Cliquez sur **Terminer**.
2. Pour exécuter l'application Java EE, démarrez le serveur d'applications. Avec le bouton droit de la souris, cliquez sur **WebSphere Application Server v7.0** et sélectionnez **Démarrer**.

Chapitre 3. Accès aux données avec les applications clients.

Après avoir configuré votre environnement de développement, vous pouvez commencer à développer des applications qui créent, accèdent et gèrent les données de votre grille de données.

Pourquoi et quand exécuter cette tâche

Du point de vue d'une application client, l'utilisation de WebSphere eXtreme Scale passe par les étapes suivantes :

- connexion au service de catalogue via une instance ClientClusterContext
- obtention d'une instance client ObjectGrid
- obtention d'une instance Session
- obtention d'une instance ObjectMap
- utilisation des méthodes ObjectMap

Interface ObjectGrid

Les méthodes suivantes permettent d'interagir avec une instance ObjectGrid.

Création et initialisation

Pour connaître la procédure de création d'instance ObjectGrid, reportez-vous à la rubrique consacrée à l'interface ObjectGrid. IL existe deux méthodes distinctes pour créer une instance ObjectGrid : par programmation ou à l'aide de fichiers XML de configuration. Pour plus d'informations, voir la documentation de l'API.

Méthodes get ou set et méthodes factory

Toutes les méthodes set doivent être appelées avant l'initialisation de l'instance ObjectGrid. Tout appel à une méthode set après celui de la méthode initialize donne une exception java.lang.IllegalStateException. Chacune des méthodes getSession de l'interface ObjectGrid appelle également la méthode initialize de manière implicite. Il faut donc appeler les méthodes set avant tout appel aux méthodes getSession. Seule exception à cette règle : la définition, l'ajout et la suppression d'objets ObjectGridEventListener. En effet, ces objets ont le droit d'être traités après la fin de l'initialisation.

L'interface ObjectGrid contient les principales méthodes suivantes.

Tableau 1. ObjectGrid (interface). Les principales méthodes de l'interface ObjectGrid.

Méthode	Description
BackingMap defineMap(String name);	defineMap : méthode factory permettant de définir une mappe de sauvegarde portant un nom exclusif. Pour plus d'informations sur les mappes de sauvegarde, voir l'interface BackingMap.
BackingMap getMap(String name);	getMap : retourne une mappe de sauvegarde précédemment définie par l'appel à defineMap. Cette méthode permet de configurer la mappe de sauvegarde, si elle ne l'a pas déjà été via une configuration XML.
BackingMap createMap(String name);	createMap : crée une mappe de sauvegarde, sans la mettre en cache pour qu'elle soit utilisée par cet ObjectGrid. Cette méthode s'utilise avec la méthode setMaps(List) de l'interface ObjectGrid, laquelle méthode met en cache des mappes de sauvegarde pour qu'elles soient utilisées par cet ObjectGrid. Ces méthodes s'utilisent lorsqu'on configure un ObjectGrid avec la structure Spring.

Tableau 1. ObjectGrid (interface) (suite). Les principales méthodes de l'interface ObjectGrid.

Méthode	Description
void setMaps(List mapList);	setMaps : supprime toutes les mappes de sauvegarde qui ont été précédemment définies dans cet ObjectGrid et les remplace par la liste des mappes qui est fournie.
public Session getSession() throws ObjectGridException, TransactionCallbackException;	getSession : retourne un objet Session, qui fournit des fonctionnalités begin, commit, rollback pour une unité d'oeuvre. Pour plus d'informations sur les objets Session, voir l'interface Session.
Session getSession(CredentialGenerator cg);	getSession(CredentialGenerator cg) : obtient une session avec un objet CredentialGenerator. Cette méthode ne peut être appelée que par le client ObjectGrid dans un environnement client/serveur.
Session getSession(Subject subject);	getSession(Subject subject) : autorise, pour obtenir un objet Session, l'utilisation d'un objet Subject spécifique à la place de l'objet configuré dans l'ObjectGrid.
void initialize() throws ObjectGridException;	initialize : l'ObjectGrid est initialisé et utilisable de manière générale. Cette méthode est appelée de manière implicite lorsque la méthode getSession est appelée si l'ObjectGrid n'est pas à l'état initialisé.
void destroy();	destroy : la structure est désassemblée et est inutilisable après l'appel de cette méthode.
void setTxTimeout(int timeout);	setTxTimeout : cette méthode permet de fixer la durée en secondes pendant laquelle a le droit de s'effectuer une transaction, qui est démarrée par une session créée par cette instance ObjectGrid. Si une transaction n'est pas terminée une fois ce délai écoulé, la session qui a démarré la transaction est marquée comme "timed out". Du fait de ce marquage, la prochaine méthode ObjectMap à être invoquée par la session expirée donne lieu à une exception . La session est marquée comme "rollback only", ce qui provoque son annulation même si l'application appelle la méthode commit au lieu de la méthode rollback après l'interception par l'application de l'exception TransactionTimeoutException. Un délai d'expiration de 0 indique que la transaction dispose d'une durée illimitée pour s'effectuer. La transaction n'expire pas si une valeur de 0 est utilisée. Si cette méthode n'est pas appelée, toute session retournée par la méthode getSession de cette interface a par défaut une valeur de délai d'expiration de transaction égale à 0. Une application peut substituer sa propre valeur de délai d'expiration sur la base d'un objet Session en utilisant la méthode setTransactionTimeout de l'interface com.ibm.websphere.objectgrid.Session. Vous pouvez également configurer ce délai d'expiration dans le fichier `objectGrid.xml` en cas de scénario réparti.
int getTxTimeout();	getTxTimeout : retourne la valeur en secondes du délai d'expiration des transactions. Cette méthode retourne la même valeur que celle qui est passée comme paramètre timeout dans la méthode setTxTimeout. Si la méthode setTxTimeout n'a pas été appelée, la méthode retourne 0 pour indiquer que la transaction dispose d'une durée illimitée pour s'exécuter.
public int getObjectGridType();	Retourne le type de l'ObjectGrid. La valeur retournée équivaut à l'une des constantes déclarées dans cette interface : LOCAL, SERVER ou CLIENT.
public void registerEntities(Class[] entities);	Permet d'enregistrer une ou plusieurs entités à partir des métadonnées de classe. L'enregistrement des entités est obligatoire avant l'initialisation de l'ObjectGrid pour lier une entité à une mappe de sauvegarde et à tous les index définis. Cette méthode peut être appelée à plusieurs reprises.
public void registerEntities(URL entityXML);	Permet d'enregistrer une ou plusieurs entités à partir d'un fichier XML d'entités. L'enregistrement des entités est obligatoire avant l'initialisation de l'ObjectGrid pour lier une entité à une mappe de sauvegarde et à tous les index définis. Cette méthode peut être appelée plusieurs fois.
void setQueryConfig(QueryConfig queryConfig);	Permet de définir l'objet QueryConfig pour cet ObjectGrid. Un objet QueryConfig fournit des configurations de requêtes pour l'exécution de requêtes sur des objets dans les mappes de cet ObjectGrid.
//Mots clés	
void associateKeyword(Serializable parent, Serializable child);	Obsolète : utilisez les fonctions Index ou de requête pour obtenir des objets ayant des attributs spécifiques. La méthode associateKeyword fournit un mécanisme flexible d'invalidation à base de mots clés. Cette méthode relie les deux mots clés dans une relation unidirectionnelle. Si le parent est invalidé, l'enfant l'est également. En revanche, l'invalidation de l'enfant n'a aucun impact sur le parent.
//Sécurité	
void setSecurityEnabled()	setSecurityEnabled : active la sécurité. La sécurité est désactivée par défaut.

Tableau 1. ObjectGrid (interface) (suite). Les principales méthodes de l'interface ObjectGrid.

Méthode	Description
void setPermissionCheckPeriod(long period);	setPermissionCheckPeriod : cette méthode accepte un seul paramètre qui indique à quelle fréquence vérifier le droit d'accès utilisé par un client. Une valeur de 0 pour ce paramètre demande au mécanisme d'autorisation, qu'il s'agisse de l'autorisation JAAS ou d'une autorisation personnalisée, de vérifier si le sujet courant dispose bien des droits d'accès. Selon l'implémentation des autorisations, cette stratégie risque d'être la cause de problèmes de performances. Mais ce type d'autorisation est disponible si nécessaire. Par ailleurs, une valeur inférieure à 0 indique le nombre de millisecondes pendant lequel un ensemble de droits d'accès doit être placé en cache avant de retourner au mécanisme d'autorisation pour être réactualisé. Ce paramètre fournit de bien meilleures performances mais, si les droits d'accès dorsaux viennent à être modifiés pendant ce délai, l'ObjectGrid risque d'autoriser ou au contraire d'interdire l'accès même si le fournisseur dorsal de sécurité a été modifié.
void setAuthorizationMechanism(int authMechanism);	setAuthorizationMechanism : permet de définir le mécanisme d'autorisation. La valeur par défaut est SecurityConstants.JAAS_AUTHORIZATION.
setMapAuthorization(MapAuthorization ma);	Obsolète : utilisez plutôt la méthode setObjectGridAuthorization (ObjectGridAuthorization) pour connecter des autorisations personnalisées. setMapAuthorization : permet de définir le plug-in MapAuthorization pour cette instance ObjectGrid. Ce plug-in permet d'autoriser les accès ObjectMap ou JavaMap aux principaux qui sont contenus dans l'objet Subject. La plupart du temps, ce plug-in est implémenté pour extraire les principaux d'un objet Subject et vérifier ensuite si les droits d'accès spécifiés sont ou non accordés à ces principaux.
setSubjectSource(SubjectSource ss);	setSubjectSource : permet de définir le plug-in SubjectSource. Ce plug-in permet d'obtenir un objet Subject qui représente le client ObjectGrid. Ce Subject est utilisé pour l'autorisation ObjectGrid. La méthode SubjectSource.getSubject est appelée par l'environnement d'exécution ObjectGrid lorsque la méthode ObjectGrid.getSession est utilisée pour obtenir une session et que la sécurité est activée. Ce plug-in est utile pour un client déjà authentifié : il permet d'extraire l'objet Subject authentifié et de le transmettre à l'instance ObjectGrid. Aucune autre authentification n'est nécessaire.
setSubjectValidation(SubjectValidation sv);	setSubjectValidation : permet de définir le plug-in SubjectValidation pour cette instance ObjectGrid. Ce plug-in permet de valider qu'un Subject javax.security.auth.Subject qui est transmis à l'ObjectGrid est un Subject valide qui n'a pas été falsifié. Une implémentation de ce plug-in nécessite la prise en charge du créateur d'objets Subject, car le créateur est le seul à savoir si l'objet Subject a été ou non falsifié. Mais il peut arriver qu'un créateur de Subject n'ait pas les moyens de savoir si le Subject a été falsifié. Dans ce cas, ce plug-in ne doit pas être utilisé.
void setObjectGridAuthorization(ObjectGridAuthorization ogAuthorization);	Permet de définir l'ObjectGridAuthorization pour cette instance ObjectGrid. Passer un null à cette méthode supprime d'un appel précédent à cette méthode un objet ObjectGridAuthorization précédemment défini et indique que cet <code>ObjectGrid</code> n'est associé à aucun objet ObjectGridAuthorization. Cette méthode ne doit être utilisée que lorsque la sécurité de l'ObjectGrid est activée. Si cette sécurité est désactivée, l'objet ObjectGridAuthorization qui est fourni ne sera pas utilisé. Un plug-in ObjectGridAuthorization permet d'autoriser l'accès à l'ObjectGrid et aux mappes. A partir de XD 6.1, setMapAuthorization est déprécié et l'utilisation de setObjectGridAuthorization est préconisée. Mais, si les deux plug-in MapAuthorization et ObjectGridAuthorization sont utilisés, l'ObjectGrid utilisera le MapAuthorization fourni pour autoriser les accès aux mappes, même si ce plug-in est déprécié.

Interface ObjectGrid : plug-in

L'interface ObjectGrid comporte plusieurs points optionnels de connexion de plug-in, qui étendent les possibilités d'interactions.

```
void addEventListener(ObjectGridEventListener cb);
void setEventListeners(List cbList);
void removeEventListener(ObjectGridEventListener cb);
void setTransactionCallback(TransactionCallback callback);
int reserveSlot(String);
// Plug-in de sécurité
void setSubjectValidation(SubjectValidation subjectValidation);
void setSubjectSource(SubjectSource source);
void setMapAuthorization(MapAuthorization mapAuthorization);
```

- ObjectGridEventListener : une interface ObjectGridEventListener permet de recevoir des notifications lorsqu'il se produit des événements importants dans l'ObjectGrid : initialisation de l'ObjectGrid, début de transaction, fin de transaction et destruction d'un ObjectGrid. Pour pouvoir être à l'écoute de ces

événements, vous devez créer une classe qui implémente l'interface ObjectGridEventListener, classe que vous ajouterez à l'ObjectGrid. Ces programmes d'écoute sont associés à chaque session. Pour plus d'informations, voir Interfaces Listener et Session.

- TransactionCallback : une interface de programme d'écoute TransactionCallback autorise des événements transactionnels (begin, commit et rollback) à envoyer des signaux à cette interface. En général, une interface listener TransactionCallback est utilisé avec un Loader. Pour plus d'informations, voir Plug-in TransactionCallback et Loaders. Ces événements permettent alors de coordonner les transactions avec une ressource externe ou au sein de plusieurs loaders.
- reserveSlot : autorise les plug-in de cet ObjectGrid à réserver des emplacements utilisables dans les instances d'objets qui, comme TxID, disposent d'emplacements.
- SubjectValidation. Si la sécurité est activée, ce plug-in permet de valider une classe javax.security.auth.Subject qui est transmise à l'ObjectGrid.
- MapAuthorization. Si la sécurité est activée, ce plug-in permet d'autoriser les accès ObjectMap aux principaux qui sont représentés par l'objet Subject.
- SubjectSource. Si la sécurité est activée, ce plug-in permet d'obtenir un objet Subject qui représente le client ObjectGrid. Ce Subject est alors utilisé pour l'autorisation ObjectGrid.

Pour plus d'informations sur les plug-in, voir l'introduction aux plug-in dans le *Guide de programmation*.

Interface BackingMap

Chaque instance ObjectGrid contient une collection d'objets BackingMap. Les méthodes defineMap ou createMap de l'interface ObjectGrid permettent de nommer et d'ajouter chaque mappe de sauvegarde à une instance ObjectGrid. Ces méthodes retournent une instance BackingMap qui sert alors à définir le comportement de la mappe correspondante. L'on peut considérer les objets BackingMap comme des caches en mémoire des données validées d'une mappe individuelle.

Interface Session

L'interface Session sert à commencer une transaction et à obtenir l'ObjectMap ou le JavaMap requis pour assurer l'interaction transactionnelle entre une application et un objet BackingMap. Mais les modifications subies par la transaction ne sont appliquées à l'objet BackingMap qu'après la validation de la transaction. L'on peut considérer les objets BackingMap comme des caches en mémoire des données validées d'une mappe individuelle. Pour plus d'informations, voir dans le *Guide de programmation* les explications concernant l'utilisation des objets Session pour accéder aux données.

L'interface BackingMap fournit des méthodes pour la définition des attributs BackingMap. Certaines des méthodes set apporte une extensibilité aux mappes de sauvegarde via plusieurs plug-in personnalisés conçus à cet effet. La liste qui suit répertorie les méthodes set permettant de définir des attributs et fournissant une prise en charge de ces plug-in personnalisés :

```
// Pour la définition des attributs de BackingMap.
public void setReadOnly(boolean readOnlyEnabled);
public void setNullValuesSupported(boolean nullValuesSupported);
public void setLockStrategy( LockStrategy lockStrategy );
```

```
public void setCopyMode(CopyMode mode, Class valueInterface);
public void setCopyKey(boolean b);
public void setNumberOfBuckets(int numBuckets);
public void setNumberOfLockBuckets(int numBuckets);
public void setLockTimeout(int seconds);
public void setTimeToLive(int seconds);
public void setTtlEvictorType(TTLType type);
public void setEvictionTriggers(String evictionTriggers);

// For setting an optional custom plug-in provided by application.
public abstract void setObjectTransformer(ObjectTransformer t);
public abstract void setOptimisticCallback(OptimisticCallback checker);
public abstract void setLoader(Loader loader);
public abstract void setPreloadMode(boolean async);
public abstract void setEvictor(Evictor e);
public void setMapEventListeners( List /*MapEventListener*/ eventListenerList );
public void addMapEventListener(MapEventListener eventListener );
public void removeMapEventListener(MapEventListener eventListener );
public void addMapIndexPlugin(MapIndexPlugin index);
public void setMapIndexPlugins(List /\* MapIndexPlugin \*/ indexList );
public void createDynamicIndex(String name, boolean isRangeIndex,
String attributeName, DynamicIndexCallback cb);
public void createDynamicIndex(MapIndexPlugin index, DynamicIndexCallback cb);
public void removeDynamicIndex(String name);
```

Il existe une méthode get pour chacune des méthodes set de la liste ci-dessus.

Attributs BackingMap

Toute mappe de sauvegarde possède les attributs suivant qu'il est possible de définir pour modifier ou pour contrôler son comportement :

- ReadOnly : indique si la mappe est en lecture seule ou en lecture-écriture. Si cet attribut n'est jamais défini pour la mappe, celle-ci est par défaut une mappe en lecture-écriture. Lorsqu'une mappe est définie comme en lecture seule, ObjectGrid optimise les performances pour ne procéder à des lectures que lorsque c'est possible.
- NullValuesSupported : indique s'il est possible de placer une valeur null dans la mappe. Si cet attribut n'est jamais défini, la mappe n'accepte jamais de valeurs null. Si les valeurs null sont admises par la mappe, une opération get qui renvoie la valeur null peut signifier que la valeur est null ou que la mappe ne contient pas la clé spécifiée par l'opération get.
- LockStrategy : détermine si un gestionnaire de verrouillage est utilisée par cette mappe de sauvegarde. Si un gestionnaire est utilisé, l'attribut LockStrategy sert à indique quelle approche est adoptée pour le verrouillage des entrées de la mappe : verrouillage optimiste ou verrouillage pessimiste. Si cet attribut n'est pas défini, c'est la stratégie de verrouillage optimiste qui est utilisée. Voir la rubrique Verrouillage pour en savoir davantage sur les stratégies de verrouillage prises en charge.
- CopyMode : détermine si une copie d'un objet value est effectuée par la mappe de sauvegarde lorsqu'une valeur est lue dans la mappe ou qu'elle est placée dans la mappe pendant le cycle de validation d'une transaction. Plusieurs modes de copie sont possibles pour permettre à l'application de trouver un compromis entre les performances et l'intégrité des données. Si cet attribut n'est pas défini, c'est le mode COPY_ON_READ_AND_COMMIT qui est utilisé. Si ce mode de copie n'offre pas les meilleures performances, il assure la meilleure protection contre les problèmes d'intégrité des données. Si la mappe de sauvegarde est associée à une entité d'API EntityManager, le paramètre CopyMode ne prend effet que si la valeur est définie comme COPY_TO_BYTES. Pour tout autre mode de copie, la valeur est toujours NO_COPY. Pour remplacer le mode de copie

d'une mappe d'entités, utilisez la méthode ObjectMap.setCopyMode. Pour plus d'informations sur les modes de copie, voir dans le *Guide de programmation* les explications concernant les pratiques recommandées pour les modes de copie.

- CopyKey : détermine si la mappe de sauvegarde effectue une copie d'un objet key lors de la première création d'une entrée dans la mappe. L'action par défaut est de ne pas faire de copie des objets key car les clés sont normalement des objets non modifiables.
- NumberOfBuckets : indique le nombre de compartiments de table de hachage que devra utiliser la mappe de sauvegarde. L'implémentation de BackingMap utilise une table de hachage. S'il existe un grand nombre d'entrées dans la mappe de sauvegarde, un plus grand nombre de compartiments sera synonyme de meilleures performances. Le nombre de clés partageant le même compartiment diminue au fur et à mesure que croît le nombre de compartiments. Mais un plus grand nombre de compartiments est également synonyme d'un plus grand nombre d'accès simultanés. Cet attribut est utile pour une optimisation fine des performances. Une valeur non définie est utilisée par défaut si l'application ne définit pas l'attribut NumberOfBuckets.
- NumberOfLockBuckets : indique le nombre de compartiments de verrouillage que devra utiliser le gestionnaire de verrouillage pour cette mappe de sauvegarde. Lorsque la stratégie de verrouillage est définie comme OPTIMISTIC ou PESSIMISTIC, un gestionnaire de verrouillage est créé pour la mappe de sauvegarde. Le gestionnaire de verrouillage utilise une table de hachage pour rechercher les entrées verrouillées par une ou plusieurs transactions. S'il existe un grand nombre d'entrées dans la table de hachage, les performances seront d'autant meilleures que les compartiments de verrouillage seront plus nombreux car le risque de collisions de clés au sein du même compartiment diminue au fur et à mesure que croît le nombre de compartiments. Mais un plus grand nombre de compartiments de verrouillage est également synonyme d'un plus grand nombre d'accès simultanés. Lorsque l'attribut LockStrategy a la valeur NONE, aucun gestionnaire de verrouillage n'est utilisé par cette mappe de sauvegarde. Dans ce cas, la définition de numberOfLockBuckets n'a aucun effet. Si cet attribut n'est pas défini, c'est une valeur indéfinie qui est utilisée.
- LockTimeout : utilisé lorsque la mappe de sauvegarde utilise un gestionnaire de verrouillage. La mappe de sauvegarde utilise un gestionnaire de verrouillage lorsque l'attribut LockStrategy a la valeur OPTIMISTIC ou PESSIMISTIC. La valeur de cet attribut est définie en secondes et elle détermine pendant combien de temps le gestionnaire de verrouillage attend l'octroi d'un verrou. Si cet attribut n'est pas défini, une valeur de 15 secondes est utilisée pour LockTimeout. Voir Verrouillage pessimiste pour les détails des exceptions de délai d'attente de verrou qui peuvent se produire.
- TtlEvictorType : toute mappe de sauvegarde a son propre expulseur pré-intégré, qui utilise un algorithme temporel pour déterminer quelles entrées expulser de la mappe. Par défaut, l'expulseur pré-intégré n'est pas actif. Vous pouvez l'activer en appelant la méthode setTtlEvictorType avec l'une de ces valeurs : CREATION_TIME, LAST_ACCESS_TIME, LAST_UPDATE_TIME ou NONE. CREATION_TIME indique que l'expulseur ajoute l'attribut TimeToLive à l'heure à laquelle l'entrée a été créée dans la mappe afin de déterminer à quel moment l'expulseur doit expulser l'entrée de la mappe. LAST_ACCESS_TIME (ou LAST_UPDATE_TIME) indique que l'expulseur ajoute l'attribut TimeToLive à l'heure du dernier accès (ou du dernier accès *et* de la dernière modification apportée) à l'entrée de mappe par une transaction qu'exécute l'application. L'entrée de mappe n'est expulsée que si aucune transaction n'y a accédé pendant la période de temps spécifiée par l'attribut TimeToLive. Enfin, NONE indique à l'expulseur de rester inactif et de ne jamais expulser aucune des entrées de la

mappe. Si cet attribut n'est jamais défini, c'est la valeur NONE qui est utilisée et l'expulseur n'est pas actif. Voir la section Expulseurs pour plus de détails sur l'expulseur pré-intégré.

- TimeToLive : cet attribut sert à spécifier le nombre de secondes nécessaires à l'expulseur pour ajouter la date de création ou de dernier accès pour chacune des entrées (voir les explications concernant l'attribut TtlEvictorType). Si cet attribut n'est jamais défini, c'est la valeur spéciale de zéro qui est utilisée pour indiquer que la durée de vie est infinity. Si cet attribut a la valeur infinity, les entrées de mappe ne sont jamais expulsées.

L'exemple qui suit montre comment définir la mappe de sauvegarde someMap dans l'instance ObjectGrid someGrid et comment définir divers attributs de cette mappe à l'aide des méthodes set de l'interface BackingMap :

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.LockStrategy;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;

...

ObjectGrid og =
ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("someGrid");
BackingMap bm = objectGrid.getMap("someMap");
bm.setReadOnly( true ); // override default of read/write
bm.setNullValuesSupported(false); // override default of allowing Null values
bm.setLockStrategy( LockStrategy.PESSIMISTIC ); // override default of OPTIMISTIC
bm.setLockTimeout( 60 ); // override default of 15 seconds.
bm.setNumberOfBuckets(251); // override default (prime numbers work best)
bm.setNumberOfLockBuckets(251); // override default (prime numbers work best)
```

Plug-in BackingMap

L'interface BackingMap comporte plusieurs points optionnels de connexion de plug-in, qui étendent les possibilités d'interactions avec l'ObjectGrid.

- **Plug-in ObjectTransformer:** pour un certain nombre d'opérations de mappage, une mappe peut avoir besoin de sérialiser, désérialiser ou copier une clé ou une valeur d'une entrée de la mappe. Celle-ci peut effectuer ces actions en fournissant une implémentation par défaut de l'interface ObjectTransformer. Une application peut améliorer les performances en fournissant un plug-in ObjectTransformer personnalisé qui sera utilisé par la mappe pour sérialiser, désérialiser ou copier une clé ou une valeur d'une entrée de la mappe. Pour plus d'informations, voir dans le *Guide de programmation* les explications concernant le plug-in ObjectTransformer.
- **Plug-in Evictor :** l'expulseur pr-intégré utilise un algorithme temporel pour décider du moment où une entrée doit être expulsée de la mappe. Certaines applications peuvent avoir besoin d'utiliser un algorithme différent pour décider de ce moment. Le plug-in Evictor met un expulseur personnalisé à la disposition de la mappe. Le plug-in Evictor vient s'ajouter à l'expulseur pré-intégré. Il ne le remplace pas. ObjectGrid fournit un plug-in Evictor personnalisé qui implémente des algorithmes bien connus comme "utilisée le plus récemment" ou "utilisée le moins fréquemment". Les applications peuvent utiliser l'un des plug-in Evictor fournis ou fournir leur propre plug-in. Voir dans le *Guide de programmation* les explications concernant l'expulsion.
- **Plug-in MapEventListener plug-in :** une application peut vouloir être au courant d'événements affectant la mappe de sauvegarde, comme l'expulsion d'une entrée ou le préchargement d'une mappe. La mappe de sauvegarde appelle des méthodes du plug-in MapEventListener pour notifier les

applications des événements qui lui sont arrivés. La méthode setMapEventListener permet aux applications de recevoir des notifications des divers événements arrivant à la mappe de sauvegarde grâce à un ou plusieurs plug-in MapEventListener personnalisés qu'elle fournit à la mappe. L'application peut modifier les objets MapEventListener à l'aide des méthodes addMapEventListener ou removeMapEventListener. Pour plus d'informations, voir dans le *Guide de programmation* les explications concernant the plug-in MapEventListener.

- **Plug-in Loader :** une mappe de sauvegarde est en fait un cache en mémoire d'un objet Map. Un plug-in Loader est une option utilisée par la mappe de sauvegarde pour déplacer les données entre la mémoire et un stockage de persistance. Ainsi, un Loader JDBC (Java Database Connectivity) pourra être utilisé pour échanger des données dans les deux sens entre une mappe de sauvegarde et une ou plusieurs tables d'une base de données relationnelle. Il n'est en effet pas nécessaire d'utiliser une base de données relationnelle comme stockage de persistance d'une mappe de sauvegarde. Le Loader peut également servir à l'échange de données entre une mappe de sauvegarde et un fichier, entre une mappe de sauvegarde et un certain nombre d'éléments : mappe Hibernate, bean entity JEE (Java 2 Platform, Enterprise Edition), autre serveur d'applications, etc. Pour chacune des technologies qu'elle utilise, l'application doit fournir un plug-in Loader personnalisé pour transférer les données entre la mappe de sauvegarde et le stockage de persistance. Si aucun Loader n'est fourni, la mappe de sauvegarde devient un simple cache en mémoire. Pour plus d'informations, voir dans le *Guide de programmation* les explications concernant l'utilisation d'un Loader.
- **Plug-in OptimisticCallback :** lorsque l'attribut LockStrategy d'une mappe de sauvegarde a pour valeur OPTIMISTIC, la mappe de sauvegarde ou un plug-in Loader doivent effectuer des opérations de comparaison pour les valeurs contenues dans la mappe. Le plug-in OptimisticCallback est alors utilisé par la mappe ou par le Loader pour effectuer des opérations de comparaison en appliquant une vérification optimiste des versions. Pour plus d'informations, voir dans le *Guide de programmation* les explications concernant le plug-in OptimisticCallback.
- **Plug-in MapIndexPlugin :** un plug-in A MapIndexPlugin (ou un Index en abrégé), est une option utilisée par la mappe de sauvegarde pour élaborer un index en fonction de l'attribut de l'objet stocké, qui est spécifié. L'index permet à l'application de trouver des objets à partir d'une valeur ou d'une plage de valeurs spécifiques. Il existe deux types d'index : statiques et dynamiques. Pour plus d'informations, voir Indexation.

Pour plus d'informations concernant les plug-in, voir l'introduction aux plug-in dans le *Guide de programmation*.

Connexion à un ObjectGrid réparti

Vous pouvez vous connecter à un ObjectGrid réparti avec un point de contact de connexion pour le service de catalogue. Vous devez connaître le nom d'hôte et le port de point de contact du serveur de catalogues auquel vous souhaitez vous connecter.

Pour vous connecter à une grille répartie, vous devez avoir configuré votre environnement côté serveur avec un service de catalogue et des serveurs conteneurs.

La méthode getObjectGrid(ClientClusterContext ccc, String objectGridName) se connecte au service de catalogue spécifié et renvoie une instance ObjectGrid client correspondant à une instance ObjectGrid côté serveur.

Le fragment de code ci-après montre comment se connecter à une grille répartie.

```
// Créez une instance ObjectGridManager.

ObjectGridManager ogm = ObjectGridManagerFactory.getObjectGridManager();

// Obtenez un contexte de cluster client en vous connectant à un ObjectGrid
// réparti basé sur un serveur de catalogues.  Vous devez fournir un
// point de contact de connexion pour votre serveur de catalogues au format
// nomHôte:portNoeudFinal.  Le nom d'hôte représente la machine
// où se trouve le serveur de catalogues et le port du point de contact,
// le port d'écoute du serveur de catalogue, dont la valeur par défaut
// est 2809.

ClientClusterContext ccc = ogm.connect("localhost:2809", null, null);

// Obtenez un ObjectGrid en utilisant le gestionnaire de grille d'objets et en
// fournissant le contexte du cluster client.

ObjectGrid og = ogm.getObjectGrid(ccc, "objectgridName");
```

Interagir avec un ObjectGrid à l'aide d'ObjectGridManager

La classe ObjectGridManagerFactory et l'interface ObjectGridManager fournissent un mécanisme permettant de créer des instances ObjectGrid, d'y accéder et de les mettre en cache. La classe ObjectGridManagerFactory est une classe auxiliaire statique permettant d'accéder à l'interface ObjectGridManager qui est un singleton. L'interface ObjectGridManager inclut plusieurs méthodes de simplification permettant de créer des instances d'un objet ObjectGrid. L'interface ObjectGridManager facilite également la création et la mise en cache d'instances ObjectGrid accessibles à plusieurs utilisateurs.

Modèle de programmation

Pour pouvoir utiliser eXtreme Scale comme une grille de données en mémoire, vous devez créer des instances ObjectGrid et interagir avec elles. Pour cela, vous pouvez utiliser des méthodes comme celles énumérées ci-après :

- méthodes createObjectGrid
- méthodes getObjectGrid
- méthodes removeObjectGrid
- contrôle du cycle de vie d'un ObjectGrid

Méthodes createObjectGrid

Nous allons découvrir les sept méthodes createObjectGrid de l'interface ObjectGridManager. Chacune de ces méthodes crée une instance locale d'ObjectGrid.

Instance locale en mémoire

Le fragment de code qui suit montre comment obtenir et configurer une instance ObjectGrid locale avec eXtreme Scale.

```
// Obtain a local ObjectGrid reference
        // you can create a new ObjectGrid, or get configured ObjectGrid
        // defined in ObjectGrid xml file
```

```
   ObjectGridManager objectGridManager =
ObjectGridManagerFactory.getObjectGridManager();
   ObjectGrid ivObjectGrid =
objectGridManager.createObjectGrid("objectgridName");

   // Add a TransactionCallback into ObjectGrid
   HeapTransactionCallback tcb = new HeapTransactionCallback();
   ivObjectGrid.setTransactionCallback(tcb);

   // Define a BackingMap
   // if the BackingMap is configured in ObjectGrid xml
   // file, you can just get it.
   BackingMap ivBackingMap = ivObjectGrid.defineMap("myMap");

   // Add a Loader into BackingMap
   Loader ivLoader = new HeapCacheLoader();
   ivBackingMap.setLoader(ivLoader);

   // initialize ObjectGrid
   ivObjectGrid.initialize();

   // Obtain a session to be used by the current thread.
   // Session can not be shared by multiple threads
   Session ivSession = ivObjectGrid.getSession();

   // Obtaining ObjectMap from ObjectGrid Session
   ObjectMap objectMap = ivSession.getMap("myMap");
```

Configuration partagée par défaut

Le code qui suit est un exemple simple montrant comment créer un ObjectGrid devant être partagé entre un grand nombre d'utilisateurs.

```
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
final ObjectGridManager oGridManager=
  ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid employees =
 oGridManager.createObjectGrid("Employees",true);
employees.initialize();
employees.
/*sample continues..*/
```

Ce fragment de code Java code créait et mettait en cache l'ObjectGrid Employees. Cet ObjectGrid était initialisé avec la configuration par défaut et était prêt à l'emploi. Le second paramètre de la méthode createObjectGrid avait la valeur true, ce qui obligeait l'ObjectGridManager à mettre en cache l'instance ObjectGrid qu'il créait. Avec une valeur false pour ce paramètre, l'instance ne serait pas mise en cache. Chaque instance ObjectGrid a un nom et l'instance peut être partagée entre un grand nombre de clients à partir de ce nom.

Si l'instance ObjectGrid est utilisé en partage d'égal à égal, la mise en cache doit être égale à true. Pour plus d'informations sur le partage d'égal à égal, voir Répartir les modifications entre des machines virtuelles Java homologues.

Configuration XML

WebSphere eXtreme Scale est extrêmement configurable. L'exemple qui précède montre comment créer un simple ObjectGrid sans aucune configuration. L'exemple qui vient montre comment créer une instance ObjectGrid préconfigurée à partir d'un fichier XML de configuration. Il existe deux manières de configurer une

instance ObjectGrid : par programmation ou à partir d'un fichier XML de configuration. Il est également possible de configurer ObjectGrid en combinant les deux approches. L'interface ObjectGridManager autorise la création d'une instance ObjectGrid à partir de la configuration XML. L'interface ObjectGridManager comporte plusieurs méthodes qui acceptent une URL comme argument. Chaque fichier XML qui est passé à ObjectGridManager doit être validé par rapport au schéma. La validation XML ne peut être désactivée que lorsque le fichier a été précédemment validé et qu'aucune modification n'est intervenue depuis la dernière validation du fichier. Désactiver la validation fait certes gagner un peu de temps système, mais au prix du risque d'utiliser un fichier XML non valide. Le JDK IBM 1.4.2 intègre la validation XML. Si l'on utilise un JDK qui n'intègre pas la validation XML, il peut être nécessaire d'utiliser Apache Xerces pour valider le code XML.

Le fragment de code Java qui suit montre comment passer un fichier XML de configuration afin de créer un ObjectGrid.

```
import java.net.MalformedURLException;
import java.net.URL;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
boolean validateXML = true; // turn XML validation on
boolean cacheInstance = true; // Cache the instance
String objectGridName="Employees"; // Name of Object Grid URL
allObjectGrids = new URL("file:test/myObjectGrid.xml");
final ObjectGridManager oGridManager=
 ObjectGridManagerFactory.getObjectGridManager();
 ObjectGrid employees =
 oGridManager.createObjectGrid(objectGridName, allObjectGrids,
  bvalidateXML, cacheInstance);
```

Le fichier XML peut contenir des informations de configuration de plusieurs ObjectGrids. Le fragment de code précédent retournait spécifiquement un ObjectGrid Employees, en présupposant que la configuration d'Employees était définie dans le fichier. Pour la syntaxe XML, voir Configuration d'ObjectGrid. Il existe sept méthodes createObjectGrid, qui sont toutes documentées dans le bloc de code qui suit.

```
/**
 * Méthode de fabrique simple permettant de retourner une instance
 * d'Object Grid. Attribution d'un nom exclusif.
 * L'instance n'est pas mise en cache.
 * Les utilisateurs peuvent se servir de {@link ObjectGrid#setName(String)}
 * pour modifier le nom de l'ObjectGrid.
 *
 * @return ObjectGrid instance d'ObjectGrid portant le nom exclusif attribué
 * @throws ObjectGridException toute erreur rencontrée pendant
 * la création de l'ObjectGrid
 */
public ObjectGrid createObjectGrid() throws ObjectGridException;

/**
 * Méthode de fabrique simple permettant de retourner une instance
 * du nom spécifié. Les instances peuvent être mises en cache. Si un ObjectGrid
 * de ce nom a déjà été mis en cache, une ObjectGridException
 * sera levée.
 *
 * @param objectGridName le nom de l'ObjectGrid à créer
 * @param cacheInstance true, si l'instance doit être mise en cache
 * @return instance ObjectGrid
 * @this le nom a déjà été mis en cache
 * ou toute erreur survenue durant la création de l'ObjectGrid
```

```
 */
public ObjectGrid createObjectGrid(String objectGridName, boolean cacheInstance)
  throws ObjectGridException;

/**
 * Crée une instance ObjectGrid du nom ObjectGrid spécifié. L'instance
 * ObjectGrid créée sera mise en cache
 * @param objectGridName le nom de l'instance ObjectGrid à créer
 * @return an ObjectGrid instance
 * @throws ObjectGridException si un ObjectGrid de ce nom a déjà
 * été mis en cache ou si une erreur a été rencontrée pendant la
 *  création de ObjectGrid
 */
public ObjectGrid createObjectGrid(String objectGridName)
  throws ObjectGridException;

/**
 * Crée une instance ObjectGrid à partir du nom ObjectGrid
 * et du fichier XML spécifiés. L'instance ObjectGrid définie dans le fichier
 * XML avec le nom ObjectGrid spécifié sera créée et retournée. Si cet
 * ObjectGrid est introuvable dans le fichier XML, une exception sera levée.
 *
 * Cette instance ObjectGrid peut être mise en cache.
 *
 * Si l'URL est null, elle sera tout bonnement ignorée. Dans ce cas, cette méthode
 * se comporte exactement comme {@link #createObjectGrid(String, boolean)}.
 *
 * @param objectGridName le nom de l'instance ObjectGrid à retourner. Ce nom
 * ne doit pas être null.
 * @param xmlFile URL vers un fichier XML correctement formé basé sur le
 * schéma ObjectGrid.
 * @param enableXmlValidation si true, le XML est validé
 * @param cacheInstance valeur booléenne indiquant
 * si la ou les instances
 * définies dans le XML seront ou non mises en cache. Si true, la ou les instances
 * seront mises en cache.
 *
 * @throws ObjectGridException si un ObjectGrid de ce nom
 * a déjà été mis en cache ou si aucun nom d'ObjectGrid n'a été trouvé
 * dans le fichier XML
 * ou si une erreur quelconque est survenue durant la création de l'ObjectGrid
 * @return an ObjectGrid instance
 * @see ObjectGrid
 */
public ObjectGrid createObjectGrid(String objectGridName, final URL xmlFile,
final boolean enableXmlValidation, boolean cacheInstance)
 throws ObjectGridException;

/**
 * Traite un fichier XML et crée une liste d'objets
 * à partir de ce fichier.
 * Ces instances ObjectGrid peuvent être mises en cache.
 * Une ObjectGridException sera levée pour toute tentative de
 * mettre en cache un nouvel ObjectGrid
 * qui aura le même nom qu'un an ObjectGrid déjà présent dans le cache.
 *
 * @param xmlFile le fichier qui définit un ou plusieurs
 * ObjectGrids
 * @param enableXmlValidation si true, le XML est validé
 * par rapport au schéma
 * @param cacheInstances si true, mise en cache de toutes les instances ObjectGrid
 * créées à partir du fichier
 * @return instance ObjectGrid
 * @throws ObjectGridException si tentative de créer et de mettre en cache
 * un ObjectGrid du même nom
 * qu'un ObjectGrid déjà mis en cache ou pour toute autre erreur
 * survenue pendant
```

```
 * la création de l'ObjectGrid
 */
public List createObjectGrids(final URL xmlFile, final boolean enableXmlValidation,
boolean cacheInstances) throws ObjectGridException;

/** Crée tous les ObjectGrids trouvés dans le fichier XML. Celui-ci sera
 * validé par rapport au schéma. Chaque instance ObjectGrid créée sera
 * mise en cache. Une ObjectGridException sera levée pour toute tentative
 * de mettre en cache un nouvel ObjectGrid qui aura le même nom
 * qu'un ObjectGrid déjà présent dans le cache.
 * @param xmlFile Le fichier XML à traiter. Les ObjectGrids seront créés
 * à partir du contenu de ce fichier.
 * @return liste des instances ObjectGrid qui ont été créées
 * @throws ObjectGridException si un ObjectGrid de même nom
 * que l'un de ceux trouvés dans le XML a déjà été mis en cache
 * ou si une autre erreur est survenue durant la création de l'ObjectGrid
 */
public List createObjectGrids(final URL xmlFile) throws ObjectGridException;

/**
 * Traite le fichier XML et crée une seule instance ObjectGrid
 * avec l'objectGridName spécifié uniquement si un ObjectGrid de ce nom est trouvé
 * dans le fichier. S'il n'existe aucun ObjectGrid de ce nom défini
 * dans le fichier XML, une ObjectGridException
 * sera levée. L'instance ObjectGrid créée sera mise en cache.
 * @param objectGridName nom de l'ObjectGrid à créer. Cet ObjectGrid
 * doit être défini dans le fichier XML.
 * @param xmlFile Le fichier XML à traiter.
 * @return Nouvel objet ObjectGrid créé
 * @throws ObjectGridException si un ObjectGrid de ce nom
 * a déjà été mis en cache ou si aucun nom d'ObjectGrid n'a été trouvé
 * dans le fichier XML ou si une erreur quelconque est survenue durant la
 * création de l'ObjectGrid
 */
public ObjectGrid createObjectGrid(String objectGridName, URL xmlFile)
  throws ObjectGridException;
```

Le client reste bloqué pendant l'appel à une méthode getObjectGrid

Il peut arriver que le client semble bloqué pendant l'appel à la méthode getObjectGrid dans ObjectGridManager ou qu'il lève une exception com.ibm.websphere.projector.MetadataException. Le référentiel EntityMetadata n'est pas disponible et le délai d'attente a été dépassé. La raison est que le client attend que les métadonnées d'entité sur le serveur ObjectGrid deviennent disponibles. Cette erreur peut se produire lorsqu'un conteneur a été démarré mais que le nombre initial de conteneurs ou le nombre minimum de fragments répliques synchrones n'a pas encore été atteint. Examinez la règle de déploiement de l'ObjectGrid et vérifiez que le nombre de conteneurs actifs est supérieur ou égal aussi bien à l'attribut numInitialContainers qu'à l'attribut minSyncReplicas dans le fichier descripteur de la règle du déploiement.

Méthodes getObjectGrid

Les méthodes ObjectGridManager.getObjectGrid permettent d'extraire les instances mises en cache.

Extraction d'une instance mise en cache

Etant donné que l'instance de l'ObjectGrid Employees (Employés) a été mise en cache par l'interface ObjectGridManager, un autre utilisateur peut y accéder à l'aide du fragment de code suivant :

```
ObjectGrid myEmployees = oGridManager.getObjectGrid("Employees");
```

Vous trouverez ci-dessous deux méthodes getObjectGrid renvoyant des instances d'ObjectGrid mises en cache :

- **Extraction de toutes les instances mises en cache**

  Pour obtenir toutes les instances d'ObjectGrid mises en cache précédemment, utilisez la méthode getObjectGrids qui renvoie une liste de chaque instance. S'il n'existe aucune instance mise en cache, la méthode renvoie null.
- **Extraction d'une instance mise en cache par nom**

  Pour obtenir une seule instance d'ObjectGrid mise en cache, utilisez la méthode getObjectGrid(String objectGridName) en transmettant le nom de l'instance mise en cache à la méthode. La méthode renvoie l'instance d'ObjectGrid portant le nom spécifié ou renvoie la valeur null s'il n'existe aucune instance d'ObjectGrid avec ce nom.

Remarque : Vous pouvez également utiliser la méthode getObjectGrid pour établir la connexion à une grille répartie. Pour plus d'informations, reportez-vous à la rubrique «Connexion à un ObjectGrid réparti», à la page 22.

Méthodes removeObjectGrid

Deux méthodes removeObjectGrid différentes permettent de retirer du cache des instances ObjectGrid.

Retirer une instance ObjectGrid

Pour retirer du cache des instances ObjectGrid, vous avez le choix entre deux méthodes removeObjectGrid. ObjectGridManager ne conserve pas de référence des instances qui sont retirées. Il existe deux méthodes pour le retrait des instances. L'une de ces méthodes accepte un paramètre booléen. Si ce paramètre a la valeur `true`, la méthode destroy est appelée dans l'ObjectGrid. L'appel à la méthode destroy dans l'ObjectGrid arrête celui-ci et libère toutes les ressources qu'il utilise. Voici comment utiliser les deux méthodes removeObjectGrid :

```
/**
 * Remove an ObjectGrid from the cache of ObjectGrid instances
 *
 * @param objectGridName the name of the ObjectGrid instance to remove
 * from the cache
 *
 * @throws ObjectGridException if an ObjectGrid with the objectGridName
 * was not found in the cache
 */
public void removeObjectGrid(String objectGridName) throws ObjectGridException;

/**
 * Remove an ObjectGrid from the cache of ObjectGrid instances and
 * destroy its associated resources
 *
 * @param objectGridName the name of the ObjectGrid instance to remove
 * from the cache
 *
 * @param destroy destroy the objectgrid instance and its associated
 * resources
 *
 * @throws ObjectGridException if an ObjectGrid with the objectGridName
 * was not found in the cache
 */
public void removeObjectGrid(String objectGridName, boolean destroy)
 throws ObjectGridException;
```

Contrôle du cycle de vie d'une instance ObjectGrid

Vous pouvez utiliser l'interface ObjectGridManager pour contrôler le cycle de vie d'une instance ObjectGrid à l'aide d'un bean de démarrage ou d'un servlet.

Gestion du cycle de vie à l'aide d'un bean de démarrage

Un bean de démarrage permet de contrôler le cycle de vie d'une instance ObjectGrid. Un bean de démarrage est chargé au démarrage d'une application. Avec un bean de démarrage, le code peut être exécuté chaque fois qu'une application démarre ou s'arrête de manière imprévue. Pour créer un bean de démarrage, utilisez l'interface home com.ibm.websphere.startupservice.AppStartUpHome et l'interface éloignée com.ibm.websphere.startupservice.AppStartUp. Implémentez les méthodes start et stop sur le bean. La méthode start est appelée chaque fois que l'application démarre. La méthode stop est appelée à l'arrêt de l'application. La méthode start permet de créer des instances ObjectGrid. La méthode stop permet de supprimer des instances ObjectGrid. Voici un fragment de code illustrant cette gestion du cycle de vie d'ObjectGrid dans un bean de démarrage :

```
public class MyStartupBean implements javax.ejb.SessionBean {
    private ObjectGridManager objectGridManager;

    /* Les méthodes de l'interface SessionBean ont été retirées
     * de cet exemple afin de limiter la taille de ce dernier */

    public boolean start(){
        // Démarrage du bean de démarrage
        // Cette méthode est appelée au démarrage de l'application
        objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
        try {
            // créez 2 instances ObjectGrid et placez-les en cache
            ObjectGrid bookstoreGrid =
      objectGridManager.createObjectGrid("bookstore", true);
            bookstoreGrid.defineMap("book");
            ObjectGrid videostoreGrid =
      objectGridManager.createObjectGrid("videostore", true);
            // dans la machine virtuelle Java.
            // Ces instances ObjectGrid peuvent maintenant être extraite
            //de l'ObjectGridManager à l'aide de la méthide getObjectGrid(String)
        } catch (ObjectGridException e) {
            e.printStackTrace();
            return false;
        }

        return true;
    }

    public void stop(){
        // Arrêt du bean de démarrage
        // Cette méthode est appelée à l'arrêt de l'application
        try {
            // Supprimez et détruisez les instances ObjectGrid en cache
            objectGridManager.removeObjectGrid("bookstore", true);
            objectGridManager.removeObjectGrid("videostore", true);
        } catch (ObjectGridException e) {
            e.printStackTrace();
        }
    }
}
```

Une fois que la méthode start a été appelée, les instances ObjectGrid nouvellement créées sont extraites de l'interface ObjectGridManager. Par exemple, si un servlet est inclus dans l'application, il accède à eXtreme Scale à l'aide du fragment de code suivant :

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.getObjectGrid("bookstore");
ObjectGrid videostoreGrid = objectGridManager.getObjectGrid("videostore");
```

Gestion du cycle de vie avec un servlet

Pour gérer le cycle de vie d'une instance ObjectGrid dans un servlet, vous pouvez utiliser la méthode init afin de créer une instance ObjectGrid et la méthode destroy afin de supprimer l'instance ObjectGrid. Si l'instance ObjectGrid est placée en cache, elle est extraite et manipulée dans le code du servlet. Vous trouverez ci-après un exemple de code illustrant la création, la manipulation et la destruction d'une instance ObjectGrid dans un servlet :

```
public class MyObjectGridServlet extends HttpServlet implements Servlet {
    private ObjectGridManager objectGridManager;

    public MyObjectGridServlet() {
        super();
    }

    public void init(ServletConfig arg0) throws ServletException {
        super.init();
        objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
        try {
            // créez et placez en cache une instance ObjectGrid intitulée bookstore
            ObjectGrid bookstoreGrid =
      objectGridManager.createObjectGrid("bookstore", true);
            bookstoreGrid.defineMap("book");
        } catch (ObjectGridException e) {
            e.printStackTrace();
        }
    }

    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        ObjectGrid bookstoreGrid = objectGridManager.getObjectGrid("bookstore");
        Session session = bookstoreGrid.getSession();
        ObjectMap bookMap = session.getMap("book");
        // effectuez des opérations sur l'instance ObjectGrid en cache
        // ...
    }

    public void destroy() {
        super.destroy();
        try {
            // supprimez et détruisez l'instance ObjectGrid bookstore mise en cache
            objectGridManager.removeObjectGrid("bookstore", true);
        } catch (ObjectGridException e) {
            e.printStackTrace();
        }
    }
}
```

Accès au fragment ObjectGrid

WebSphere eXtreme Scale atteint un haut débit de traitement en plaçant la logique au niveau des données et en renvoyant uniquement les résultats au client.

La logique d'application d'une machine virtuelle Java (JVM) du client doit extraire les données de la JVM du serveur et les replacer une fois la transaction validée. Ce processus ralentit le débit auquel sont traitées les données. Si la logique d'application se trouve sur la même JVM que le fragment contenant les données, les coûts de temps d'attente du réseau et de conversion des paramètres sont éliminés et peuvent signifier une nette amélioration des performances.

Référence locale aux données du fragment

Les API ObjectGrid proposent une session à la méthode côté serveur. Cette session est une référence directe aux données de ce fragment. Aucune logique de routage n'est placée sur ce chemin. La logique d'application peut utiliser directement les données de ce fragment. La session ne peut pas être utilisée pour accéder aux données dans une autre partition, car aucune logique de routage n'existe.

Un plug-in Loader constitue également un moyen de recevoir un événement lorsqu'un fragment devient une partition principale. Une application peut implémenter un plug-in Loader et l'interface ReplicaPreloadController. La méthode de vérification de l'état de chargement est appelée uniquement lorsqu'un fragment devient primaire. La session fournie à cette méthode est une référence locale aux données des fragments. Cette approche est généralement utilisée lorsqu'une partition principale doit démarrer des unités d'exécution ou s'abonner à une matrice de messages pour le trafic lié à la partition. Elle peut démarrer une unité d'exécution pour être à l'écoute des messages d'une mappe locale utilisant l'API getNextKey.

Optimisation de client-serveur regroupé

Si une application utilise les API du client pour accéder à une partition groupée avec la JVM qui contient le client, le réseau est omis, mais des conversions de paramètres ont encore lieu en raison des problèmes d'implémentation actuels. Si une grille partitionnée est utilisée, aucun impact sur les performances de l'application n'est constaté, car le nombre (N-1)/N d'appels conduit vers une JVM différente. Si vous avez besoin d'un accès local permanent à un fragment, utilisez les API Loader ou ObjectGrid pour appeler cette logique.

Accès aux données dans WebSphere eXtreme Scale

Une fois qu'une application dispose d'une référence à une instance ObjectGrid ou d'une connexion client à une grille distante, vous pouvez accéder aux données et interagir avec celles-ci dans votre configuration WebSphere eXtreme Scale. Avec l'API ObjectGridManager, utilisez l'une des méthodes createObjectGrid pour créer une instance locale ou la méthode getObjectGrid pour une instance client associée à une grille répartie.

L'unité d'exécution d'une application requiert sa propre Session. Lorsqu'une application souhaite utiliser la grille d'objets dans une unité d'exécution, il lui suffit d'appeler l'une des méthodes getSession pour obtenir une unité d'exécution. Cette opération consomme peu de ressources, car il n'est généralement pas nécessaire de la mettre en pool. Si l'application utilise une structure d'injection de dépendance telle que Spring, vous pouvez injecter une Session dans le bean d'une application lorsque cette opération est nécessaire.

Lorsque vous avez obtenu une Session, l'application peut accéder aux données stockées dans des mappes de la grille d'objets. Si la grille utilise des entités, vous pouvez utiliser l'API EntityManager, que vous pouvez obtenir à l'aide de la

méthode Session.getEntityManager. L'interface EntityManager étant plus proche des spécifications Java, elle est plus simple que l'API fondée sur les mappes. Toutefois, elle consomme davantage de ressources car elle effectue un suivi des modifications apportées aux objets. L'API fondée sur les mappes s'obtient à l'aide de la méthode Session.getMap.

WebSphere eXtreme Scale utilise des transactions. Lorsqu'une application interagit avec une Session, elle doit se trouver dans le contexte d'une transaction. Une transaction est initiée, puis validée ou annulée à l'aide des méthodes Session.begin, Session.commit et Session.rollback dans l'objet Session. Les applications peuvent également utiliser le mode de validation automatique, dans lequel la Session initie et valide automatiquement une transaction lorsque l'application interagit avec des mappes. Le mode de validation automatique est toutefois plus lent.

Logique d'utilisation des transactions

Les transactions peuvent paraître lentes, mais eXtreme Scale les utilise toutefois pour les trois raisons suivantes :

1. Pour pouvoir annuler des modifications en cas d'exception ou si la logique métier requiert l'annulation d'une modification d'état.
2. Pour placer des verrous sur les données et les libérer tout au long de la durée de vie d'une transaction, afin que les modifications soient apportées de manière atomique (toutes les modifications sont appliquées, ou aucune).
3. Pour générer une unité atomique de réplication.

WebSphere eXtreme Scale permet à la Session de choisir dans quelle mesure une transaction est réellement nécessaire. Une application peut désactiver la prise en charge de l'annulation et le verrouillage, mais elle prend cette décision à ses dépens. Elle devra alors gérer elle-même l'absence de ces fonctions.

Une application peut par exemple désactiver le verrouillage en associant la stratégie de verrouillage de la mappe de sauvegarde à la valeur NONE. Cette stratégie est rapide, mais plusieurs transactions simultanées peuvent désormais modifier les mêmes données sans protection les unes des autres. L'application est responsable du verrouillage et de la cohérence des données lorsque NONE est utilisé.

Une application peut également modifier la façon dont les objets sont copiés lorsque la transaction y accède. L'application utilise la méthode ObjectMap.setCopyMode pour définir le mode de copie. Cette méthode vous permet de désactiver CopyMode. Cette opération est généralement utilisée pour les transactions en lecture seule si différentes valeurs peuvent être renvoyées pour le même objet au cours d'une même transaction. Différentes valeurs peuvent être renvoyées pour le même objet au cours d'une transaction.

Par exemple, si la transaction a appelé la méthode ObjectMap.get pour un objet à l'instant T1, elle a obtenu la valeur de cet instant précis. Si elle appelle à nouveau la méthode get à l'instant T2, une autre unité d'exécution peut avoir modifié cette valeur. Suite à cette modification, l'application voit une valeur différente. Si l'application modifie un objet récupéré à l'aide de la valeur CopyMode NONE, elle modifie directement la copie validée de cet objet. Dans ce mode, l'annulation de la transaction n'a aucune signification. Vous modifiez la seule copie dans la grille d'objets. Bien que l'utilisation de CopyMode NONE soit rapide, vous devez en mesurer les conséquences. Une application utilisant ce mode ne doit jamais annuler la transaction. Si elle le fait, les modifications ne sont pas reportées dans l'index *et*

ne sont pas répliquées si la réplication est activée. Les valeurs par défaut sont plus simples à utiliser et risquent d'entraîner moins d'erreurs. Si vous favorisez les performances au détriment de la fiabilité des données, l'application doit savoir comment réagir afin d'éviter tout problème non intentionnel.

ATTENTION :
Soyez prudent lorsque vous modifiez les valeurs associées au verrouillage ou à CopyMode. Si vous les modifiez, l'application se comporte de manière non prévisible.

Interaction avec les données stockées

Dès que vous avez obtenu une session, vous pouvez utiliser le fragment de code suivant pour insérer des données à l'aide de l'API Map.

```
Session session = ...;
ObjectMap personMap = session.getMap("PERSON");
session.begin();
Person p = new Person();
p.name = "John Doe";
personMap.insert(p.name, p);
session.commit();
```

Le même exemple, mais utilisant cette fois l'API EntityManager, est présenté ci-dessous. Cet exemple de code suppose que l'objet Person est mappé vers une entité.

```
Session session = ...;
EntityManager em = session.getEntityManager();
session.begin();
Person p = new Person();
p.name = "John Doe";
em.persist(p);
session.commit();
```

Ce modèle est conçu pour obtenir des références aux ObjectMaps pour les mappes avec lesquelles l'unité d'exécution va travailler, pour démarrer une transaction, utiliser les données, puis valider la transaction.

L'interface ObjectMap contient les opérations put, get et remove habituelles. Nous vous recommandons toutefois d'utiliser des noms d'opérations plus précis tels que get, getForUpdate, insert, update et remove. Ces méthodes permettent d'exécuter des opérations plus précises que les API Map habituelles.

Vous pouvez également utiliser la prise en charge de l'indexation, qui est flexible.

Voici un exemple de mise à jour d'un objet :

```
session.begin();
Person p = (Person)personMap.getForUpdate("John Doe");
p.name = "John Doe";
p.age = 30;
personMap.update(p.name, p);
session.commit();
```

L'application utilise normalement la méthode getForUpdate plutôt que la simple méthode get pour verrouiller un enregistrement. La méthode de mise à jour doit être appelée pour fournir la valeur mise à jour à la mappe. Si tel n'est pas le cas, la mappe n'est pas modifiée. Voici le même fragment de code, utilisant cette fois l'API EntityManager :

```
session.begin();
Person p = (Person)em.findForUpdate(Person.class, "John Doe");
p.age = 30;
session.commit();
```

L'API EntityManager est plus simple que l'API Map. Avec EntityManager, eXtreme Scale recherche l'entité et renvoie un objet géré à l'application. Celle-ci modifie l'objet et valide la transaction, eteXtreme Scale suit automatiquement les modifications apportées aux objets gérés lors de la validation et effectue les mises à jour nécessaires.

Transactions et partitions

Les transactions WebSphere eXtreme Scale permettent de mettre à jour une seule partition. Les transactions provenant d'un client peuvent lire plusieurs partitions, mais n'en mettent à jour qu'une. Si une application tente de mettre à jour deux partitions, la transaction échoue et est annulée. Une transaction utilisant un ObjectGrid imbriqué (logique de grille) ne dispose d'aucune fonction de routage et voit uniquement les données de la partition locale. Cette logique métier peut toujours obtenir une seconde session, qui est alors une véritable session client permettant d'accéder aux autres partitions. Cette transaction reste toutefois indépendante.

Requêtes et partitions

Si une transaction a déjà recherché une entité, elle est associée à la partition de cette entité. Les requêtes s'exécutant sur une transaction associée à une entité sont acheminées vers la partition associée.

Si une requête est exécutée sur une transaction avant d'être associée à une partition, vous devez définir l'ID de partition à utiliser pour cette requête. Cet ID est un nombre entier. La requête est alors acheminée vers cette partition.

Les requêtes exécutent des recherches dans une seule partition. Toutefois, vous pouvez utiliser les API DataGrid pour exécuter la même requête en parallèle sur toutes les partitions ou sur un sous-ensemble de partitions. Utilisez ces API pour rechercher une entrée pouvant se trouver sur n'importe quelle partition.

7.0.0.0 FIX 2+ Le service de données REST permet aux clients HTTP d'accéder à la grille WebSphere eXtreme Scale. Ce service est compatible avec WCF Data Services dans Microsoft .NET Framework 3.5 SP1. Pour plus d'informations, voir le guide d'utilisation du service de données eXtreme Scale REST.

.

Pratiques CopyMode recommandées

WebSphere eXtreme Scale effectue une copie de la valeur en appliquant l'un des six paramétrages possibles pour CopyMode. C'est à vous de déterminer lequel de ces paramétrages fonctionne le mieux pour les besoins de votre déploiement.

La méthode setCopyMode(CopyMode, valueInterfaceClass) de l'API BackingMap permet de définir le mode de copie d'après l'un des champs statiques finaux suivants qui sont définis dans la classe com.ibm.websphere.objectgrid.CopyMode.

Lorsqu'une application utilise l'interface ObjectMap pour obtenir une référence à une entrée de mappe, n'utilisez cette référence qu'au sein de la transaction

WebSphere eXtreme Scale qui s'est procuré cette référence. Son utilisation dans un autre transaction peut provoquer des erreurs. Si, par exemple, vous utilisez la stratégie de verrouillage pessimiste pour la mappe de sauvegarde, un appel à la méthode get ou à la méthode getForUpdate acquerra un verrou S (shared) ou U (update), selon la transaction. La méthode get retournera la référence à la valeur et le verrou obtenu sera libéré au terme de la transaction. La transaction doit appeler les méthodes get ou getForUpdate pour verrouiller l'entrée de mappe dans une autre transaction. Chaque transaction doit obtenir sa propre référence à la valeur en appelant les méthodes get ou getForUpdate au lieu de réutiliser la même référence dans plusieurs transactions.

CopyMode pour les mappes d'entités

Lorsqu'on utilise une mappe qui est associée à une entité d'API EntityManager, la mappe retourne toujours directement les objets tuple d'entités sans procéder à une copie, sauf si l'on utilise COPY_TO_BYTES comme mode de copie. Il est donc important de modifier CopyMode, faute de quoi le tuple ne serait pas copié de manière appropriée en cas de changement.

COPY_ON_READ_AND_COMMIT

Le mode COPY_ON_READ_AND_COMMIT est le mode par défaut. L'argument valueInterfaceClass est ignoré lorsque ce mode est utilisé. Ce mode s'assure qu'une application ne contient pas de référence à l'objet value qui se trouve dans la mappe de sauvegarde. A la place, l'application utilise toujours une copie de la valeur qui se trouve dans l'instance BackingMap. Le mode COPY_ON_READ_AND_COMMIT garantit que l'application ne pourra jamais endommager par inadvertance les données qui sont mises en cache dans la mappe de sauvegarde. Lorsqu'une transaction d'application appelle une méthode ObjectMap.get pour une clé donnée et qu'il s'agit du premier accès à l'entrée ObjectMap de cette clé, c'est une copie de la valeur qui est retournée. Lorsque la transaction est validée, toutes les modifications validées par l'application sont alors copiées vers la mappe de sauvegarde pour garantir que l'application ne dispose d'aucune référence à la valeur validée dans la mappe.

COPY_ON_READ

Par rapport au mode COPY_ON_READ_AND_COMMIT, le mode COPY_ON_READ donne de meilleures performances car il élimine la copie qui est effectuée lors de la validation des transactions. L'argument valueInterfaceClass est ignoré lorsque ce mode est utilisé. Pour conserver l'intégrité des données de la mappe de sauvegarde, l'application s'assure que toutes les références à une entrée sont supprimées une fois la transaction validée. Dans ce mode, la méthode ObjectMap.get retourne une copie de la valeur au lieu d'une référence à celle-ci afin de garantir que les modifications de la valeur opérées par l'application n'affecteront pas la valeur dans la mappe tant que la transaction n'est pas validée. Mais la différence est que, lors de la validation, il n'est effectué aucune copie des modifications. Ce qui est stocké dans la mappe de sauvegarde, c'est la référence à la copie, celle qui a été retournée par la méthode ObjectMap.get. Une fois la transaction validée, l'application détruit toutes les références aux entrées de la mappe. Si l'application ne détruit pas les références, les données mises en cache dans la mappe de sauvegarde risquent d'être endommagées. Si une application utilise ce mode et rencontre des problèmes, passez en mode COPY_ON_READ_AND_COMMIT pour voir si le problème persiste. S'il disparaît, c'est que l'application ne parvient pas à détruire toutes ses références après que la transaction a été validée.

COPY_ON_WRITE

Par rapport au mode COPY_ON_READ_AND_COMMIT, le mode COPY_ON_WRITE donne lui aussi de meilleures performances car il élimine la copie qui est effectuée lorsque la méthode ObjectMap.get est appelée pour la première fois par une transaction pour une clé donnée. La méthode ObjectMap.get retourne un proxy de la valeur au lieu d'une référence directe à l'objet value. Le proxy garantit qu'aucune copie de la valeur n'est effectuée tant que l'application n'appelle pas de méthode set sur l'interface Value qui est spécifiée comme argument valueInterfaceClass. Le proxy fournit une copie lors de l'écriture. Lors de la validation d'une transaction, la mappe de sauvegarde examine le proxy pour déterminer si une copie a été effectuée en tant que résultat de l'appel à une méthode set. Si c'est le cas, la référence à cette copie est stockée dans la mappe de sauvegarde. Ce mode présente donc un énorme avantage : une valeur n'est jamais copiée lors d'une lecture ou lors d'une validation lorsque la transaction ne fait jamais appel à une méthode set pour modifier la valeur.

Les modes COPY_ON_READ_AND_COMMIT et COPY_ON_READ procèdent tous les deux à une copie complète lorsqu'une valeur est extraite de la mappe d'objet. Ces modes ne sont pas optimaux si une application ne modifie que certaines des valeurs qui sont extraites dans une transaction. A cet égard, le mode COPY_ON_WRITE est beaucoup plus efficace, mais il requiert que l'application utilise un pattern simple. Les objets value sont tenus de prendre en charge une interface. L'application doit utiliser les méthodes de cette interface lorsqu'elle interagit avec la valeur dans une session eXtreme Scale. Si c'est le cas, eXtreme Scale crée des proxys pour les valeurs qui sont retournées à l'application. Le proxy a une référence qui est une valeur réelle. Si l'application n'effectue que des opérations de lecture, ces dernière s'exécutent toujours sur la copie réelle. Si l'application modifie un attribut dans l'objet, le proxy commence par créer une copie de l'objet réel, puis il effectue la modification dans cette copie. A partir de ce stade, le proxy n'intervient que sur cette copie. L'utilisation de la copie permet à l'opération de copie d'être totalement évitée pour les objets qui ne sont que lus par l'application. Toutes les opérations de modification doivent commencer par le préfixe set. Les JavaBeans™ Enterprise sont normalement programmés pour nommer de la sorte les méthodes qui modifient les attributs des objets. Il convient donc de respecter cette convention. Tous les objets qui sont modifiés sont copiés au moment même où ils sont modifiés par l'application. Ce scénario de lecture-écriture est le scénario le plus efficace pris en charge par eXtreme Scale. Pour configurer une mappe afin qu'elle utilise le mode COPY_ON_WRITE, utilisez l'exemple qui suit. Dans cet exemple, l'application stocke des objets Person qui sont indexés à l'aide du nom (name) présent dans la mappe. L'objet Person est représenté dans le fragment de code qui suit.

```
class Person {
    String name;
    int age;
    public Person() {
    }
    public void setName(String n)  {
        name = n;
    }
    public String getName() {
        return name;
    }
    public void setAge(int a) {
        age = a;
    }
```

```
    public int getAge() {
        return age;
    }
}
```

L'application n'utilise l'interface IPerson que lorsqu'elle interagit avec des valeurs qui sont extraites d'un ObjectMap. Modifiez l'objet pour qu'il utilise une interface comme dans l'exemple qui suit.

```
interface IPerson
{
    void setName(String n);
    String getName();
    void setAge(int a);
    int getAge();
}
// L'on modifie Person pour implémenter l'interface IPerson
class Person implements IPerson {
    ...
}
```

L'application a alors besoin de configurer la mappe de sauvegarde pour que celle-ci utilise le mode COPY_ON_WRITE, comme dans l'exemple qui suit :

```
ObjectGrid dg = ...;
BackingMap bm = dg.defineMap("PERSON");
// L'on utilise COPY_ON_WRITE pour cette mappe
// avec IPerson comme classe valueProxyInfo
bm.setCopyMode(CopyMode.COPY_ON_WRITE,IPerson.class);
// L'application doit alors utiliser
// le pattern suivant lorsqu'on utilise la mappe PERSON.
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
...
sess.begin();
// l'application transtype en IPerson et non en Person la valeur retournée
IPerson p = (IPerson)person.get("Billy");
p.setAge(p.getAge()+1);
...
// l'on fabrique un nouveau Person et on l'ajoute à Map
Person p1 = new Person();
p1.setName("Bobby");
p1.setAge(12);
person.insert(p1.getName(), p1);
sess.commit();
// le fragment suivant NE FONCTIONNERA PAS. Il se traduira par une
ClassCastExceptionsess.begin();
// l'erreur ici est que c'est Person qui est utilisé
// et non IPerson
Person a = (Person)person.get("Bobby");
sess.commit();
```

La première section montre l'application extrayant une valeur qui a été nommée Billy dans la mappe. L'application transtype la valeur retournée en objet IPerson et non en objet Person car le proxy qui est retourné implémente deux interfaces :

- l'interface spécifiée dans l'appel à la méthode BackingMap.setCopyMode
- l'interface com.ibm.websphere.objectgrid.ValueProxyInfo

Vous pouvez transtyper le proxy en deux types. La dernière partie du fragment de code montre ce qui n'est pas autorisé en mode COPY_ON_WRITE. L'application extrait l'enregistrement Bobby et essaie de le transtyper en objet Person. Cette action échoue avec une exception de transtypage de classe car le proxy qui est retourné n'est pas un objet Person. Le proxy retourné implémente en effet l'objet IPerson et l'interface ValueProxyInfo.

L'interface ValueProxyInfo et prise en charge des actualisations partielles : cette interface autorise une application à extraire soit la valeur en lecture seule validée qui est référencée par le proxy, soit l'ensemble des attributs qui ont été modifiés au cours de cette transaction.

```
public interface ValueProxyInfo {
    List /**/ ibmGetDirtyAttributes();
    Object ibmGetRealValue();
}
```

La méthode ibmGetRealValue retourne une copie en lecture seule de l'objet. L'application ne doit pas modifier cette valeur. La méthode ibmGetDirtyAttributes retourne une liste de chaînes représentant les attributs qui ont été modifiés par l'application au cours de cette transaction. ibmGetDirtyAttributes est essentiellement utilisé dans un loader JDBC (Java Database Connectivity) ou CMP. Les attributs mentionnés dans la liste sont les seuls qui ont besoin d'être actualisés, que ce soit dans l'instruction SQL ou dans l'objet mappé à la table, ce qui donne une bien plus grande efficacité au SQL généré par le loader. Lorsqu'une transaction copy on write est validée et si un loader est connecté, le loader peut transtyper vers l'interface ValueProxyInfo les valeurs des objets modifiés pour obtenir ces informations.

Gérer la méthode equals lors de l'utilisation de COPY_ON_WRITE ou de proxys : le code suivant construit un objet Person qu'il insère ensuite dans un ObjectMap. Ensuite, il extrait ce même objet à l'aide de la méthode ObjectMap.get. La valeur est transtypée vers l'interface. Si la valeur est transtypée vers l'interface Person, une exception ClassCastException est générée car la valeur retournée est un proxy qui implémente l'interface IPerson et ce n'est pas un objet Person. La vérification d'égalité échoue car l'on utilise l'opération == alors qu'il ne s'agit pas du même objet.

```
session.begin();
// new sur l'objet Person
Person p = new Person(...);
personMap.insert(p.getName, p);
// On l'extrait à nouveau (pensez à utililser l'interface pour le transtypage)
IPerson p2 = personMap.get(p.getName());
if(p2 == p) {
    // ils sont identiques
} else {
    // ils ne le sont pas
}
```

Autre point à prendre en considération : lorsqu'on doit remplacer la méthode equals. Comme le montre le fragment de code qui suit, la méthode equals doit vérifier que l'argument est un objet qui implémente l'interface IPerson et elle doit transtyper l'argument en IPerson. Comme l'argument peut très bien être un proxy qui implémente l'interface IPerson, vous devez utiliser les méthodes getAge et getName lorsque vous comparez l'égalité des variables d'instance.

```
{
    if ( obj == null ) return false;
    if ( obj instanceof IPerson ) {
        IPerson x = (IPerson) obj;
        return ( age.equals( x.getAge() ) && name.equals( x.getName() ) )
    }
    return false;
}
```

Configurations requises pour ObjectQuery et HashIndex : lorsqu'on utilise COPY_ON_WRITE avec le moteur de requête ObjectQuery ou avec un plug-in HashIndex, il est important de configurer le schéma ObjectQuery et le plug-in

HashIndex pour que ces derniers puissent accéder aux objets à l'aide des méthodes de propriétés, ce qui est l'accès par défaut. S'ils sont configurés pour utiliser un accès aux champs, le moteur de requête et l'index tenteront d'accéder aux champs de l'objet proxy, ce qui retournera toujours un null ou un 0 puisque l'instance d'objet est un proxy.

NO_COPY

Le mode NO_COPY autorise une application à s'assurer qu'elle ne modifie jamais d'objet value obtenu à l'aide d'une méthode ObjectMap.get en échange d'améliorations des performances. L'argument valueInterfaceClass est ignoré lorsque ce mode est utilisé. Si ce mode est utilisé, il n'est jamais effectué de copie de la valeur. Si l'application modifie des valeurs, les données de la mappe de sauvegarde sont endommagées. Le mode NO_COPY est essentiellement utile pour les mappes en lecture seule où les données ne sont jamais modifiées par l'application. Si l'application utilise ce mode et rencontre des problèmes, passez en mode COPY_ON_READ_AND_COMMIT pour voir si le problème persiste. S'il disparaît, c'est que l'application modifie la valeur retournée par la méthode ObjectMap.get, soit pendant la transaction, soit après que celle-ci a été validée. Toutes les mappes associées aux entités de 'API EntityManager utilisent automatiquement ce mode sans tenir compte de ce qui est spécifié dans la configuration d'eXtreme Scale.

Toutes les mappes associées aux entités de 'API EntityManager utilisent automatiquement ce mode sans tenir compte de ce qui est spécifié dans la configuration d'eXtreme Scale.

COPY_TO_BYTES

Il est possible de stocker des objets dans un format sérialisé au lieu du format POJO. COPY_TO_BYTES permet de réduire l'empreinte mémoire consommée par un graphe d'objets de taille importante. Pour des informations complémentaires, voir «Mappes de tableaux d'octets», à la page 40.

Utilisation incorrecte de CopyMode

Des erreurs se produisent lorsqu'une application tente d'améliorer les performances en utilisant les modes COPY_ON_READ, COPY_ON_WRITE ou NO_COPY décrits plus haut. Les erreurs intermittentes ne se produisent pas lorsqu'on passe au mode COPY_ON_READ_AND_COMMIT.

Problème

Le problème peut être dû à des données endommagées dans la mappe ObjectGrid, ce qui est la conséquence de la violation par l'application du contrat de programmation propre au mode de copie utilisé. L'altération des données peut provoquer des erreurs imprévisibles par intermittence ou d'une manière inexpliquée ou inattendue.

Solution

L'application doit respecter le contrat de programmation qui est énoncé pour le mode de copie utilisé. Dans les modes COPY_ON_READ et COPY_ON_WRITE, l'application utilise une référence à un objet valeur extérieur à la portée de la transaction à partir de laquelle la référence a été obtenue. Pour pouvoir utiliser ces modes, l'application doit supprimer la référence à l'objet value après la fin de la

transaction et obtenir une nouvelle référence à l'objet value à chaque transaction qui accède à cet objet. En mode NO_COPY, l'application ne doit jamais modifier l'objet value. Dans ce cas, soit programmez l'application pour qu'elle ne touche pas à l'objet value, soit faites-lui utiliser un autre mode de copie.

Mappes de tableaux d'octets

Vous pouvez stocker les paires clé-valeur de vos mappes dans un tableau d'octets plutôt que sous la forme d'un objet Java simple, ce qui réduit l'encombrement mémoire nécessaire à un graphe d'objets de grande taille.

Avantages

La quantité de mémoire nécessaire augmente avec le nombre d'objets d'un graphe. Lorsque vous réduisez un graphe complexe à un tableau d'octets, un seul objet est conservé dans le segment de mémoire, et non plusieurs. L'environnement d'exécution Java exécute ses recherches dans un nombre réduit d'objets lors de la récupération de place.

Le mécanisme de copie par défaut utilisé par WebSphere eXtreme Scale est la sérialisation, qui est un mécanisme coûteux. Si, par exemple, vous utilisez le mode de copie par défaut `COPY_ON_READ_AND_COMMIT`, une copie est faite au moment de la lecture et au moment de l'extraction. Avec un tableau d'octets, la valeur augmente en raison du nombre d'octets, mais aucune copie n'est exécutée au moment de la lecture, et la valeur est sérialisée en plusieurs octets, mais aucune copie n'est exécutée au moment de l'extraction. L'utilisation des tableaux d'octets résulte en une cohérence des données équivalente à celle qui serait obtenue avec le paramètre par défaut, mais en réduisant la mémoire utilisée.

Notez qu'un mécanisme de sérialisation optimisé est indispensable pour pouvoir constater une réduction de la mémoire consommée dans le cadre de l'utilisation des tableaux d'octets. Pour plus d'informations, voir «Performances de sérialisation», à la page 231.

Configuration des mappes de tableaux d'octets

Vous pouvez activer les mappes de tableaux d'octets à l'aide du fichier XML ObjectGrid et en associant l'attribut CopyMode utilisé par une mappe au paramètre COPY_TO_BYTES, comme dans l'exemple suivant :

```
<backingMap name="byteMap" copyMode="COPY_TO_BYTES" />
```

Pour plus d'informations, consultez la rubrique relative au fichier XML descripteur d'ObjectGrid du *Guide d'administration*.

Considérations

Vous devez réfléchir à l'éventuelle utilisation des mappes de tableaux d'octets dans un scénario donné. Même si celle-ci doit vous permettre de réduire la consommation de mémoire, l'utilisation du processeur risque d'augmenter.

La liste suivante recense les différents facteurs à prendre en compte avant d'opter pour l'utilisation de la fonction de mappes de tableaux d'octets.

Type de l'objet

Avec certains types d'objet, les mappes de tableaux d'octets ne permettent pas de réduire la consommation de mémoire. Il existe donc certains types d'objets pour lesquels l'utilisation de ces mappes n'est pas recommandée. Si vous utilisez comme valeur des encapsuleurs primitifs Java ou un objet Java simple ne contenant aucune référence à d'autres objets (se contentant de stocker des champs primitifs), le nombre d'objets Java est déjà aussi peu élevé que possible : il se limite à un. La quantité de mémoire utilisée par cet objet étant déjà optimisée, l'utilisation d'une mappe de tableaux d'octets n'est pas recommandée. Ce type de mappes convient mieux aux types d'objets contenant d'autres objets ou collections d'objets dans lesquels le nombre total d'objets simples Java est supérieur à un.

Par exemple, dans le cas d'un objet Customer associé à une adresse professionnelle, à une adresse personnelle et à une collection de commandes, vous pouvez réduire le nombre d'objets du segment de mémoire et le nombre d'octets utilisés par ces objets à l'aide des mappes de tableaux d'octets.

Adresse locale

Lorsque vous utilisez d'autres modes de copie, vous pouvez optimiser les applications lors des copies si les objets sont clonables avec l'ObjectTransformer par défaut ou lorsqu'un ObjectTransformer personnalisé est fourni avec une méthode copyValue optimisée. Par rapport aux autres modes de copie, le coût des copies des opérations de lecture, d'écriture ou de validation est supérieur lorsque l'accès aux objets se fait localement. Si, par exemple, un cache local existe dans une topologie répartie ou que vous accédiez directement à une instance ObjectGrid locale ou de serveur, la durée d'accès et de validation augmentent en cas d'utilisation des mappes de tableaux d'octets du fait du coût de la sérialisation. Dans le même type de topologie, ce coût augmente également si vous utilisez de agents de grille de données ou si vous accédez au serveur principalement lorsque vous utilisez le plug-in ObjectGridEventGroup.ShardEvents.

Interactions des plug-in

Avec les mappes de tableaux d'octets, les objets n'augmentent pas lors de la communication d'un client à un serveur, à moins que le serveur n'ait besoin du formulaire d'objet Java simple. Les plug-in qui interagissent avec la valeur de la mappe vont connaître une réduction de leurs performances en raison de l'augmentation nécessaire de la valeur.

Les plug-in utilisant LogElement.getCacheEntry ou LogElement.getCurrentValue devront subir ce coût supplémentaire. Si vous souhaitez obtenir la clé, vous pouvez utiliser LogElement.getKey, qui permet d'éviter les frais supplémentaires associés à la méthode LogElement.getCacheEntry().getKey. Les sections suivantes décrivent les plug-in dans le contexte de l'utilisation des tableaux d'octets.

Index et requêtes

Lorsque des objets sont stockés au format objet Java simple, le coût de l'indexation et de l'interrogation est minime car l'objet n'a pas besoin d'être augmenté. Lorsque vous utilisez une mappe de tableau d'objets, vous devez subir un coût supplémentaire lié à l'augmentation de l'objet. En général, si votre application utilise des index ou des requêtes, nous ne vous recommandons pas d'utiliser les mappes de tableaux d'octets, à moins que vous exécutiez uniquement des requêtes sur les attributs clés.

Verrouillage optimiste

Lorsque vous utilisez la stratégie de verrouillage optimiste, vous devrez subir un coût supplémentaire lors des opérations de mise à jour et d'invalidation. Vous devez en effet augmenter la valeur du serveur pour obtenir la valeur de la version permettant une vérification de la collision optimiste. Si vous utilisez le verrouillage optimiste uniquement pour garantir les opérations d'extraction, mais que vous n'ayez pas besoin de la vérification de collision optimiste, vous pouvez utiliser com.ibm.websphere.objectgrid.plugins.builtins.NoVersioningOptimisticCallback pour désactiver la vérification de la version.

Chargeur

Avec un Loader, vous devez également subir le coût lié à l'augmentation et à la resérialisation de la valeur. Vous pouvez cependant utiliser les mappes de tableaux d'octets avec un Loader, mais vous devez prendre en compte le coût des modifications à apporter dans ce genre de scénario. Vous pouvez par exemple utiliser la fonction de tableau d'octets dans le contexte d'un cache qui est principalement lu. Dans ce cas, l'avantage lié à un nombre réduit d'objets dans le segment de mémoire et à une utilisation moindre de la mémoire compense la perte provoquée par l'utilisation des tableaux d'octets pour les opérations d'insertion et de mise à jour.

ObjectGridEventListener

Lorsque vous utilisez la méthode transactionEnd dans le plug-in ObjectGridEventListener, vous devez supporter un coût supplémentaire côté serveur pour les demandes distantes lors de l'accès à un CacheEntry de LogElement ou à la valeur en cours. Si l'implémentation de la méthode n'accède pas à ces champs, ce coût supplémentaire n'est pas généré.

Utilisation des sessions pour accéder aux données de la grille

Les applications peuvent commencer et terminer les transactions par le biais de l'interface Session. L'interface Session permet également d'accéder aux interfaces ObjectMap et JavaMap basées sur l'application.

Chaque instance ObjectMap ou JavaMap est directement associée à un objet Session spécifique. Chaque unité d'exécution souhaitant accéder à eXtreme Scale doit préalablement obtenir une instance Session à partir de l'objet ObjectGrid. Une instance Session ne peut pas être partagée par plusieurs unités d'exécution. WebSphere eXtreme Scale n'utilise aucun stockage local d'unités d'exécution ; cependant, les restrictions de plate-forme risquent de limiter le passage d'une Session d'une unité d'exécution à une autre.

Méthodes

Les méthodes suivantes sont disponibles avec l'interface Session. Reportez-vous à la documentation de l'API pour plus d'informations sur les méthodes suivantes :

```
public interface Session {
    ObjectMap getMap(String cacheName) throws UndefinedMapException;

    void begin() throws TransactionAlreadyActiveException, TransactionException;

    void beginNoWriteThrough() throws TransactionAlreadyActiveException,
   TransactionException;

    public void commit() throws NoActiveTransactionException, TransactionException;

    public void rollback() throws NoActiveTransactionException, TransactionException;

    public void flush() throws TransactionException;
```

```
    TxID getTxID() throws NoActiveTransactionException;

    boolean isWriteThroughEnabled();

    void setTransactionType(String tranType);

    public void processLogSequence(LogSequence logSequence) throws
   NoActiveTransactionException, UndefinedMapException, ObjectGridException;

    ObjectGrid getObjectGrid();

    public void setTransactionTimeout(int timeout);
    public int getTransactionTimeout();
    public boolean transactionTimedOut();

    public boolean isCommitting();
    public boolean isFlushing();

    public void markRollbackOnly(Throwable t) throws NoActiveTransactionException;
    public boolean isMarkedRollbackOnly();
}
```

Méthode Get

Une application obtient une instance Session à partir d'un objet ObjectGrid à l'aide de la méthode ObjectGrid.getSession. L'exemple suivant illustre comment obtenir une instance Session :

```
ObjectGrid objectGrid = ...; Session sess = objectGrid.getSession();
```

Une fois l'instance Session obtenue, l'unité d'exécution garde une référence à la session pour son propre usage. Un nouvel objet Session est renvoyé à chaque appel de la méthode getSession.

Transactions et méthodes de session

Une session peut être utilisée pour commencer, valider ou annuler une transaction. Les opérations sur BackingMaps avec ObjectMaps et JavaMaps sont exécutées plus efficacement dans une transaction Session. Une fois une transaction commencée, toute modification apportée à un ou plusieurs mappes de sauvegarde dans cette transaction est stockée dans un cache de transaction spécial jusqu'à ce que la transaction soit validée. Lorsqu'une transaction est validée, les modifications en attente sont appliquées aux BackingMaps et Loaders et deviennent visibles par tous les clients de cet ObjectGrid.

WebSphere eXtreme Scale permet également de valider automatiquement les transactions. Si une opération ObjectMap est effectuée en dehors du contexte d'une transaction active, une transaction implicite est lancée avant l'opération et la transaction est automatiquement validée avant de rendre le contrôle à l'application.

```
Session session = objectGrid.getSession();
ObjectMap objectMap = session.getMap("someMap");
session.begin();
objectMap.insert("key1", "value1");
objectMap.insert("key2", "value2");
session.commit();
cobjectMap.insert("key3", "value3"); // auto-validation
```

Méthode Session.flush

La méthode Session.flush est utile lorsqu'un Loader est associé à une BackingMap. La méthode flush appelle le Loader avec l'ensemble de modifications actuel dans le cache de transaction. Le Loader applique les changements au dorsal. Les changements ne sont pas validés lorsque la méthode flush est appelée. Si une transaction de session est validée après l'appel de flush, seules les mises à jour postérieures à l'appel de flush sont appliquées au Loader. Si une transaction de

session est annulée après l'appel de la méthode flush, les modifications vidées sont annulées, de même que toutes les autres modifications en attente dans la transaction. Utilisez la méthode flush avec parcimonie car elle limite les opérations de traitement par lots pour un Loader. L'exemple ci-dessous illustre l'utilisation de la méthode Session.flush :

```
Session session = objectGrid.getSession();
session.begin();
// faites des modifications supplémentaires
...
session.flush(); // envoyez ces modifications vers le programme de chargement, mais
// ne validez pas ces modifications supplémentaires
...
session.commit();
```

Méthode NoWriteThrough

Certaines mappes eXtreme Scale sont sauvegardées par un Loader, qui fournit un stockage de persistance aux données de la mappe. Il est parfois utile de valider les données uniquement dans la mappe eXtreme Scale et de ne pas envoyer les données au Loader. L'interface de session fournit la méthode beginNoWriteThough dans ce but. La méthode beginNoWriteThrough commence une transaction comme la méthode begin. Avec la méthode beginNoWriteThrough, lorsque la transaction est validée, les données sont uniquement validées dans la mappe en mémoire eXtreme Scale et elles ne sont pas validées dans le stockage de persistance fourni par le Loader. Cette méthode est particulièrement utile lors du préchargement des données sur la mappe.

Lors de l'utilisation d'une instance ObjectGrid répartie, la méthode beginNoWriteThrough est utile pour effectuer des modifications dans le cache proche uniquement, sans modifier le cache éloigné du serveur. Si les données sont périmées dans le cache proche, l'utilisation de la méthode beginNoWriteThrough peut permettre d'invalider les entrées sur le cache proche sans les invalider sur le serveur.

L'interface Session fournit également la méthode isWriteThroughEnabled pour déterminer quel type de transaction est actuellement actif.

```
Session session = objectGrid.getSession();
session.beginNoWriteThrough();
// faites des modifications supplémentaires...
session.commit(); // ces modifications ne seront pas envoyées au Loader
```

Obtention de la méthode d'objet TxID

L'objet TxID est un objet opaque qui identifie la transaction active. Utilisez l'objet TxID pour les applications suivantes :

- Pour effectuer des comparaisons lorsque vous recherchez une transaction spécifique.
- Pour stocker des données partagées entre les objets TransactionCallback et Loader.

Reportez-vous au plug-in TransactionCallback et aux Loaders pour des informations supplémentaires sur la fonction d'attribut Object.

Méthode de suivi des performances

Si vous utilisez eXtreme Scale dans WebSphere Application Server, il peut être nécessaire de réinitialiser le type de transaction pour le suivi des performances.

Vous pouvez définir le type de transaction avec la méthode setTransactionType. Pour plus d'informations sur la méthode setTransactionType, reportez-vous à la section concernant le suivi des performances d'ObjectGrid par le biais de l'infrastructure d'analyse des performances (PMI) de WebSphere Application Server.

Exécution de la méthode LogSequence

WebSphere eXtreme Scale peut propager des ensembles de modifications de mappe en programmes d'écoute ObjectGrid pour répartir les mappes d'une machine virtuelle Java à une autre. Pour que le programme d'écoute puisse traiter les LogSequences plus facilement, l'interface Session fournit la méthode processLogSequence. Cette méthode examine chaque LogElement dans l'objet LogSequence et effectue l'opération appropriée, par exemple une insertion, une mise à jour, une invalidation, etc. sur la BackingMap identifiée par LogSequence MapName. Une session ObjectGrid doit être disponible avant l'appel de la méthode processLogSequence. L'application a également la responsabilité d'effectuer les appels de validation ou d'annulation appropriés pour terminer la session. L'auto-validation n'est pas disponible pour cet appel de méthode. Le traitement normal par l'ObjectGridEventListener récepteur au niveau de la JVM distante est de démarrer une session en utilisant la méthode beginNoWriteThrough, qui empêche la propagation sans fin des modifications , puis d'appeler la méthode processLogSequence, et enfin de valider et d'annuler la transaction.

```
// Utilisez l'objet Session transmis au cours de
//ObjectGridEventListener.initialization...
session.beginNoWriteThrough();
// traitez la LogSequence reçue
try {
 session.processLogSequence(receivedLogSequence);
} catch (Exception e) {
 session.rollback(); throw e;
}
// validez les modifications
session.commit();
```

Méthode markRollbackOnly

Cette méthode est utilisée pour marquer la transaction actuelle en tant que "rollback only" (annulation uniquement). En marquant la transaction "rollback only", vous vous assurez que, même si la méthode de validation est appelée par une application, la transaction est annulée. Cette méthode est généralement utilisée par ObjectGrid lui-même ou par l'application lorsqu'une corruption de données est susceptible de se produire si la transaction est validée. Une fois cette méthode appelée, l'objet Throwable transmis à cette méthode est chaîné à l'exception com.ibm.websphere.objectgrid.TransactionException qui résulte de la méthode de validation si elle est appelée sous une session précédemment marquée comme "rollback only". Tout appel ultérieur de cette méthode pour une transaction déjà marquée en tant que "rollback only" est ignoré. Cela signifie que seul le premier appel transmettant une référence Throwable non nul est utilisé. Une fois la transaction marquée terminée, la marque "rollback only" est supprimée afin que la prochaine transaction lancée par la session soit validée.

Méthode isMarkedRollbackOnly

Renvoie un résultat si Session est marquée "rollback only". Cette méthode renvoie la valeur booléenne True si et uniquement si la méthode markRollbackOnly a été précédemment appelée sur cette Session et si la transaction commencée par cette Session est toujours active.

Méthode setTransactionTimeout

Définissez le délai d'attente de la prochaine transaction démarrée par cette Session sur un nombre spécifique de secondes. Cette méthode n'affecte pas le délai d'attente des transactions précédemment commencées par cette Session. Cela affecte uniquement les transactions lancées après l'appel de la méthode. Si cette méthode n'est jamais appelée, la valeur de délai d'attente passée à la méthode setTxTimeout de la méthode com.ibm.websphere.objectgrid.ObjectGrid est utilisée.

Méthode getTransactionTimeout

Cette méthode renvoie la valeur de délai d'attente de la transaction, exprimée en secondes. La dernière valeur passée en tant que valeur de délai d'attente à la méthode setTransactionTimeout est renvoyée par cette méthode. Si la méthode setTransactionTimeout n'est jamais appelée, la valeur de délai d'attente passée à la méthode setTxTimeout de la méthode com.ibm.websphere.objectgrid.ObjectGrid est utilisée.

transactionTimedOut

Cette méthode renvoie une valeur booléenne si le délai d'attente de la transaction actuelle commencée par cette Session a été dépassé.

Méthode isFlushing

La méthode renvoie la valeur booléenne True si et uniquement si toutes les modifications de transaction sont vidées vers le plug-in Loader comme conséquence de la méthode de vidage de l'interface Session appelée. Cette méthode peut être utile à un plug-in Loader lorsqu'il doit savoir pourquoi la méthode batchUpdate a été appelée.

Méthode isCommitting

Cette méthode renvoie la valeur booléenne True si et uniquement si toutes les modifications de transaction sont validées comme conséquence de la méthode de validation de l'interface de Session appelée. Cette méthode peut être utile à un plug-in Loader lorsqu'il doit savoir pourquoi la méthode batchUpdate a été appelée.

Méthode setRequestRetryTimeout

Cette méthode définit, en millisecondes, la valeur de délai d'attente avant la prochaine tentative de requête pour la session. Si le client définit une valeur de délai d'attente avant la prochaine tentative de requête, le paramètre de la session remplace la valeur du client.

Méthode getRequestRetryTimeout

Cette méthode récupère le paramètre de délai d'attente avant la prochaine tentative de requête sur la session. La valeur `-1` indique que le délai d'attente n'est pas défini. La valeur `0` indique qu'il est en mode d'interruption immédiate. Une valeur

supérieure à 0 indique le paramètre de délai d'attente, en millisecondes.

SessionHandle pour le routage

Lorsque vous utilisez une stratégie de positionnement de partition par conteneur, vous pouvez utiliser une instance SessionHandle. Une instance SessionHandle contient des informations de partition pour la session en cours et peut être réutilisée pour une nouvelle session.

Une instance SessionHandle inclut des informations sur la partition à laquelle la session en cours est liée. SessionHandle est particulièrement utile pour la stratégie de positionnement de partition par conteneur. La sérialisation standard Java peut lui être appliquée.

Si vous avez une instance SessionHandle, vous pouvez appliquer cet indicateur à une session avec la méthode setSessionHandle(SessionHandle target), en utilisant l'indicateur en tant que cible. Vous pouvez récupérer l'instance SessionHandle avec la méthode Session.getSessionHandle.

Etant donné que cela est applicable uniquement dans un scénario de positionnement par conteneur, SessionHandle émet une exception IllegalStateException si un ObjectGrid donné dispose de plusieurs ensembles de mappes par conteneur ou n'en a aucun. Si vous n'invoquez pas la méthode setSessionHandle avant la méthode getSessionHandle, l'instance SessionHandle appropriée sera sélectionnée en fonction de la configuration de ClientProperties.

Vous pouvez également utiliser la classe auxiliaire SessionHandleTransformer pour convertir l'indicateur en différents formats. Les méthodes de cette classe peuvent modifier la représentation d'un indicateur, d'un tableau d'octets en une instance, d'une chaîne en une instance, et vice versa dans les deux cas. Elles permettent d'écrire le contenu de l'indicateur dans le flux de sortie.

Pour un exemple d'utilisation de l'instance SessionHandle, reportez-vous à la section relative au routage par zone préférée dans la *Présentation du produit*.

Intégration de l'objet SessionHandle

Un objet SessionHandle englobe les informations relatives à la partition pour la session à laquelle il est lié et facilite le routage des demandes. Les objets SessionHandle s'appliquent uniquement au scénario de positionnement de partition par conteneur.

Objet SessionHandle pour le routage des demandes

Vous pouvez lier un objet SessionHandle à une session en procédant comme suit :

Conseil : Dans chaque appel de méthode suivant, une fois qu'un SessionHandle est lié à une session, l'objet SessionHandle peut être obtenu à partir de la méthode Session.getSessionHandle utilisée avec la méthode Session.setSessionHandle.

- Appelle la méthode Session.getSessionHandle : au moment où cette méthode est appelée, s'il n'existe aucun SessionHandle lié à la session, un SessionHandle est sélectionné de façon aléatoire et est lié à la session.
- Appelle les opérations CRUD (create, read, update, delete) transactionnelles : au moment où ces méthodes sont appelées, s'il n'existe aucun SessionHandle lié à la session, un SessionHandle est sélectionné de façon aléatoire et est lié à la session.

- Appelle la méthode ObjectMap.getNextKey : au moment où cette méthode est appelée, s'il n'existe aucun SessionHandle lié à la session, la demande d'opération est acheminée de façon aléatoire vers les partitions individuelles jusqu'à obtention d'une clé. Si une clé est renvoyée d'une partition, un SessionHandle correspondant à cette partition est lié à la session. Si aucune clé n'est trouvée, aucun SessionHandle n'est lié à la session.
- Appelle la méthode QueryQueue.getNextEntity ou QueryQueue.getNextEntities : au moment où cette méthode est appelée, s'il n'existe aucun SessionHandle lié à la session, la demande d'opération est acheminée de façon aléatoire vers les partitions individuelles jusqu'à obtention d'un objet. Si un objet est renvoyé d'une partition, un SessionHandle correspondant à cette partition est lié à la session. Si aucun objet n'est trouvé, aucun SessionHandle n'est lié à la session.
- Définit un SessionHandle à l'aide de la méthode Session.setSessionHandle(SessionHandle sh) : si un SessionHandle est obtenu de la méthode Session.getSessionHandle, le SessionHandle peut être lié à une session. La définition d'un SessionHandle influence le routage des demandes dans l'étendue de la session à laquelle il est lié.

La méthode Session.getSessionHandle renvoie toujours un SessionHandle et lie automatiquement un SessionHandle à la session si aucun SessionHandle n'est lié à la session. Si vous voulez uniquement vérifier si une session comporte un SessionHandle, appelez la méthode Session.isSessionHandleSet. Si cette méthode renvoie la valeur false, aucun SessionHandle n'est actuellement lié à la session.

nouvelle méthode ajoutée à l'API Session

```
/**
  * Détermine si un SessionHandle est actuellement défini sur cette session.
  *
  * @return true si un SessionHandle est actuellement défini sur cette session.
  *
  * @since 7.1
  */

 public boolean isSessionHandleSet();
```

Principaux types d'opérations dans le scénario de positionnement par conteneur

Les éléments ci-dessous résument le comportement de routage des principaux types d'opérations dans le scénario de positionnement de partition par conteneur eu égard aux objets SessionHandle.

- **Objet de session avec objet lié SessionHandle**
  - Index - API MapIndex et MapRangeIndex : SessionHandle
  - Query et ObjectQuery : SessionHandle
  - Agent - API MapGridAgent et ReduceGridAgent : SessionHandle
  - ObjectMap.Clear : SessionHandle
  - ObjectMap.getNextKey : SessionHandle
  - QueryQueue.getNextEntity, QueryQueue.getNextEntities : SessionHandle
  - Opérations CRUD transactionnelles (API ObjectMap et API EntityManager) : SessionHandle
- **Objet de session sans objet lié SessionHandle**
  - Index - API MapIndex et MapRangeIndex : toutes les partitions actives
  - Query et ObjectQuery : partition spécifiée via la méthode setPartition de Query et ObjectQuery

- Agent - MapGridAgent et ReduceGridAgent
  - Non pris en charge : méthodes ReduceGridAgent.reduce(Session s, ObjectMap map, Collection keys) et MapGridAgent.process(Session s, ObjectMap map, Object key).
  - Toutes les méthodes actives : méthodes ReduceGridAgent.reduce(Session s, ObjectMap map) et MapGridAgent.processAllEntries(Session s, ObjectMap map).
- ObjectMap.clear : toutes les partitions actives.
- ObjectMap.getNextKey : lie un SessionHandle à la session si une clé est renvoyée d'une des partitions sélectionnées de façon aléatoire. Si aucune clé n'est envoyée, la session n'est pas liée à un SessionHandle.
- QueryQueue : spécifie une partition avec la méthode QueryQueue.setPartition. Si aucune partition est définie, la méthode sélectionne une partition à renvoyer de façon aléatoire. Si un objet est renvoyé, la session actuelle est liée au SessionHandle lié à la partition renvoyant l'objet. Si aucun objet n'est renvoyé, la session n'est pas liée à un SessionHandle.
- Opérations CRUD transactionnelles (API ObjectMap et API EntityManager) : sélectionne une partition de façon aléatoire.

Dans la plupart des cas, il est conseillé d'utiliser l'objet SessionHandle pour contrôler le routage vers une partition particulière. Vous pouvez extraire et mettre en cache le SessionHandle de la session qui insère les données. Après la mise en cache du SessionHandle, vous pouvez le définir sur une autre session de façon à acheminer les demandes vers la partition spécifiée par le SessionHandle mis en cache. Pour effectuer des opérations telles que ObjectMap.clear sans SessionHandle, vous pouvez définir temporairement le SessionHandle sur la valeur null en appelant Session.setSessionHandle(null). Sans un SessionHandle spécifié, les opérations s'exécutent sur toutes les partitions actives.

- **Comportement de routage QueryQueue**

  Dans le scénario de positionnement de la partition par conteneur, vous pouvez utiliser SessionHandle pour contrôler le routage des méthodes getNextEntity et getNextEntities de l'API QueryQueue. Si la session est liée à un SessionHandle, les demandes sont acheminées vers la partition à laquelle le SessionHandle est lié. Si la session n'est pas liée à un SessionHandle, les demandes sont acheminées vers la partition définie à l'aide de la méthode QueryQueue.setPartition si une partition a été définie de cette manière. Si la session n'est associée à aucun SessionHandle ou aucune partition, une partition sélectionnée de façon aléatoire est renvoyée. Si aucune partition de ce type n'est trouvée, le processus s'interrompt et aucun SessionHandle n'est lié à la session actuelle.

Le fragment de code suivant montre comment utiliser l'objet SessionHandle.

Exemple de programmation de l'objet SessionHandle

```
Session ogSession = objectGrid.getSession();

// liaison du SessionHandle
SessionHandle sessionHandle = ogSession.getSessionHandle();

ogSession.begin();
ObjectMap map = ogSession.getMap("planet");
map.insert("planet1", "mercury");

// transaction est acheminée vers la répartition spécifiée par SessionHandle
ogSession.commit();
```

```
// met en cache SessionHandle qui insère des données
SessionHandle cachedSessionHandle = ogSession.getSessionHandle();

// vérifiez si SessionHandle est défini sur la session
boolean isSessionHandleSet = ogSession.isSessionHandleSet();

// déconnecte temporairement SessionHandle de la session
if(isSessionHandleSet) {
    ogSession.setSessionHandle(null);
}

// si la session n'est liée à aucun SessionHandle,
// l'opération clear s'exécute sur toutes les partitions actives
// et supprime toutes les données de la mappe dans l'ensemble de la grille
map.clear();

// après l'opération clear, réinitialisez SessionHandle,
// si la session doit utiliser le SessionHandle précédent.
// Eventuellement l'appel de getSessionHandle peut provoquer un nouveau
// SessionHandle
ogSession.setSessionHandle(cachedSessionHandle);
```

Considérations liées à la conception d'applications

Dans le scénario de stratégie de positionnement par conteneur, il est conseillé d'utiliser l'objet SessionHandle pour la plupart des opérations. Le SessionHandle contrôle le routage vers les partitions. Pour extraire des données, le SessionHandle lié à la session doit être identique à celui de la transaction de données d'insertion.

Lorsque vous voulez effectuer une opération sans un SessionHandle défini sur la session, vous pouvez déconnecter un SessionHandle d'une session en appelant la méthode Session.setSessionHandle(null).

Lorsqu'une session est liée à un SessionHandle, toutes les demandes d'opération sont acheminées vers la partition spécifiée par le SessionHandle. Sans la définition d'un SessionHandle, les opérations sont acheminées vers toutes les partitions ou une partition sélectionnée de manière aléatoire.

Mise en cache d'objets où aucune relation n'est impliquée (API ObjectMap)

Les mappes d'objet sont identiques aux mappes Java qui permettent le stockage de données en tant que paires clé-valeur. Les mappes d'objet offrent une approche simplifiée et intuitive de stockage des données par l'application. Une mappe d'objet se prête parfaitement à la mise en cache d'objets qui n'entretiennent aucune relation entre eux. Si les objets ont des relations entre eux, il est conseillé d'utiliser l'API EntityManager.

Pour plus d'informations sur l'API EntityManager, reportez-vous à la rubrique «Mise en cache d'objets et de leurs relations (API EntityManager)», à la page 61.

Les applications obtiennent généralement une référence WebSphere eXtreme Scale, puis un objet Session à partir de la référence de chaque unité d'exécution. Les sessions ne peuvent pas être partagées par plusieurs unités d'exécution. La méthode getMap de la session renvoie une référence à une mappe d'objet devant être utilisée pour cette unité d'exécution.

Introduction à l'interface ObjectMap

L'interface ObjectMap est utilisée pour l'interaction transactionnelle entre les applications et les mappes de sauvegarde.

Rôle

Une instance ObjectMap est extraite d'un objet de session qui correspond à l'unité d'exécution en cours. L'interface ObjectMap représente le principal moyen utilisé par les applications pour apporter des modifications aux entrées d'une mappe de sauvegarde.

Obtention d'une instance ObjectMap

Une application extrait une instance ObjectMap d'un objet de session à l'aide de la méthode Session.getMap(String). Le fragment de code suivant montre comment obtenir une instance ObjectMap :

```
ObjectGrid objectGrid = ...;
BackingMap backingMap = objectGrid.defineMap("mapA");
Session sess = objectGrid.getSession();
ObjectMap objectMap = sess.getMap("mapA");
```

Chaque instance ObjectMap correspond à un objet de session particulier. Le fait d'appeler plusieurs fois la méthode getMap sur un objet Session particulier avec le même nom de mappe de sauvegarde renvoie toujours la même instance ObjectMap.

Transactions de validation automatique

Les opérations sur les mappes de sauvegarde qui utilisent des instances ObjectMaps et JavaMaps sont effectuées de manière plus efficace dans une transaction de session. WebSphere eXtreme Scale prend en charge la validation automatique si les méthodes sur les interfaces ObjectMap et JavaMap sont appelées en dehors d'une transaction de session. Les méthodes démarrent une transaction implicite, effectuent l'opération demandée et valident la transaction implicite.

Sémantique des méthodes

Vous trouverez ci-après une explication de la sémantique de chaque méthode des interfaces ObjectMap et JavaMap. La méthode setDefaultKeyword, la méthode invalidateUsingKeyword et les méthodes qui contiennent un argument sérialisable sont abordées dans la rubrique sur les mots clés. La méthode setTimeToLive est abordée dans la rubrique sur les expulseurs. Pour plus d'informations sur ces méthodes, reportez-vous à la documentation de l'API.

Méthode containsKey
: La méthode containsKey détermine si une clé possède une valeur dans la mappe de sauvegarde ou le chargeur. Si les valeurs null sont prises en charge par une application, cette méthode peut être utilisée pour déterminer si une référence null renvoyée par une opération d'extraction fait référence à une valeur null ou indique que la mappe de sauvegarde et le chargeur ne contiennent pas la clé.

Méthode flush
: La sémantique de la méthode flush est similaire à celle de la méthode flush sur l'interface Session, à la différence près que la méthode flush de l'interface Session applique les modifications actuelles en attente de toutes

les mappes modifiées dans la session en cours. Avec cette méthode, seules les modifications apportées à cette instance ObjectMap sont vidées dans le chargeur.

Méthode get
La méthode get extrait l'entrée de l'instance BackingMap. Si l'entrée est introuvable dans l'instance BackingMap, mais qu'un chargeur est associé à l'instance BackingMap, cette dernière tente d'extraire l'entrée du chargeur. La méthode getAll est fournie pour permettre un traitement d'extraction par lots.

Méthode getForUpdate
La méthode getForUpdate est identique à la méthode get, mais l'utilisation de la méthode getForUpdate indique à la mappe de sauvegarde et au chargeur une intention de mise à jour de l'entrée. Un chargeur peut utiliser ce conseil pour lancer une requête SELECT for UPDATE sur un système dorsal de base de données. Si une stratégie de verrouillage pessimiste est définie pour la mappe de sauvegarde, le gestionnaire de verrouillage verrouille l'entrée. La méthode getAllForUpdate est fournie pour permettre un traitement d'extraction par lots.

Méthode insert
La méthode insert insère une entrée dans la mappe de sauvegarde et le chargeur. L'utilisation de cette méthode indique à la mappe de sauvegarde et au chargeur que vous souhaitez insérer une entrée qui n'existait pas précédemment. Si vous appelez cette méthode sur une entrée existante, une exception se produit lorsque la méthode est appelée ou que la transaction en cours est validée.

Méthode invalidate
La sémantique de la méthode invalidate dépend de la valeur du paramètre **isGlobal** transmise à la méthode. La méthode invalidateAll est fournie pour permettre un traitement d'invalidation par lots.

L'invalidation locale est spécifiée si la valeur false est transmise au paramètre **isGlobal** de la méthode invalidate. L'invalidation locale annule les modifications apportées à l'entrée dans le cache des transactions. Si l'application génère une méthode get, l'entrée est extraite de la dernière valeur validée dans la mappe de sauvegarde. Si aucune entrée n'est présente dans la mappe de sauvegarde, l'entrée est extraite de la dernière valeur vidée ou validée du chargeur. Si une transaction est validée, les entrées marquées comme invalidées en local n'ont aucun impact sur la mappe de sauvegarde. Les modifications vidées dans le chargeur sont quand même validées, même si l'entrée a été invalidée.

L'invalidation globale est spécifiée si la valeur true est transmise comme paramètre **isGlobal** de la méthode invalidate. L'invalidation globale supprime les modifications en attente apportées à l'entrée dans le cache des transactions et ignore la valeur BackingMap sur les opérations suivantes effectuées sur l'entrée. Si une transaction est validée, les entrées marquées comme invalidées globalement sont expulsées de la mappe de sauvegarde. Soit le scénario d'invalidation suivant : La mappe de sauvegarde est sauvegardée par une table de base de données qui contient une colonne d'incrémentation automatique. Les colonnes d'incrémentation sont utiles pour affecter des numéros uniques aux enregistrements. L'application insère une entrée. Après l'insertion, l'application doit connaître le numéro de séquence de la ligne insérée. Elle sait que sa copie

de l'objet est ancienne et elle utilise donc l'invalidation globale pour extraire la valeur du chargeur. Le code suivant illustre ce scénario d'utilisation :

```
Session sess = objectGrid.getSession();
ObjectMap map = sess.getMap("mymap");
sess.begin();
map.insert("Billy", new Person("Joe", "Bloggs", "Manhattan"));
sess.flush();
map.invalidate("Billy", true);
Person p = map.get("Billy");
System.out.println("Version column is: " + p.getVersion());
map.commit();
```

Cet exemple de code ajoute une entrée pour `Billy`. L'attribut de version de Person est défini à l'aide d'une colonne d'incrémentation automatique dans la base de données. L'application effectue d'abord une commande d'insertion. Elle lance ensuite une opération de vidage qui envoie l'insertion au chargeur et à la base de données. La base de données affecte à la colonne de version le numéro suivant dans la séquence, ce qui rend obsolète l'objet Person dans la transaction. Pour mettre à jour l'objet, l'application est invalidée globalement. La méthode get suivante qui est générée extrait l'entrée du chargeur et ignore la valeur de la transaction. L'entrée est extraite de la base de données avec la valeur de version mise à jour.

Méthode put

La sémantique de la méthode put varie suivant qu'une méthode get précédente ait été appelée dans la transaction pour la clé. Si l'application lance une opération get qui renvoie une entrée existant dans la mappe de sauvegarde ou le chargeur, l'appel de méthode put est interprété comme une mise à jour et renvoie la valeur précédente dans la transaction. Si un appel de méthode put a été exécuté sans appel de méthode get précédent ou qu'un appel de méthode get précédent n'a pas trouvé d'entrée, l'opération est interprétée comme une insertion. La sémantique des méthodes insert et update s'applique lorsque l'opération put est validée. La méthode putAll est fournie pour permettre un traitement d'insertion et de mise à jour par lots.

Méthode remove

La méthode remove supprime l'entrée de la mappe de sauvegarde et du chargeur, si un chargeur est connecté. La valeur de l'objet supprimé est renvoyée par cette méthode. Si l'objet n'existe pas, cette méthode renvoie une valeur null. La méthode removeAll est fournie pour permettre un traitement de suppression par lots sans les valeurs de retour.

Méthode setCopyMode

La méthode setCopyMode spécifie une valeur de mode de copie pour cet ObjectMap. Avec cette méthode, une application peut remplacer la valeur du mode de copie qui est spécifiée sur la mappe de sauvegarde. La valeur du mode de copie spécifiée est appliquée tant que la méthode clearCopyMode n'est pas appelée. Les deux méthodes sont appelées en dehors des liaisons transactionnelles. Une valeur de mode de copie ne peut pas être modifiée en cours de transaction.

Méthode touch

La méthode touch met à jour le dernier temps d'accès d'une entrée. Cette méthode n'extrait pas la valeur de la mappe de sauvegarde. Utilisez cette méthode dans sa propre transaction. Si la clé fournie n'existe pas dans la

mappe de sauvegarde car elle a été invalidée ou supprimée, une exception se produit lors du traitement de validation.

Méthode update
La méthode update met à jour de manière explicite une entrée de la mappe de sauvegarde et du chargeur. L'utilisation de cette méthode indique à la mappe de sauvegarde et au chargeur que vous souhaitez mettre à jour une entrée existante. Une exception se produit si vous appelez cette méthode sur une entrée qui n'existe pas lorsque la méthode est appelée ou lors du traitement de validation.

Méthode getIndex
La méthode getIndex tente d'obtenir un index nommé généré sur la mappe de sauvegarde. L'index ne peut pas être partagé entre les unités d'exécution et utilise les mêmes règles qu'une session. L'objet d'index renvoyé doit être transtypé dans l'interface d'index d'application appropriée (par exemple, l'interface MapIndex, l'interface MapRangeIndex ou une interface d'index personnalisée).

Méthode clear
La méthode clear supprime toutes les entrées de cache dans une mappe sur toutes les partitions. Cette opération étant une fonction d'auto-validation, aucune transaction active ne doit être présente lors de l'appel de la méthode clear.

Remarque : La méthode clear n'efface que le contenu de la mappe sur laquelle elle est appelée ; les mappes d'entités associées ne sont pas affectées. Cette méthode n'appelle pas le plug-in du chargeur.

Mappes dynamiques

La fonction de mappes dynamiques vous permet de créer des mappes après l'initialisation de la grille.

Dans les versions précédentes de eXtreme Scale, vous deviez définir les mappes avant d'initialiser la grille d'objets. Vous deviez donc créer toutes les mappes à utiliser avant d'exécuter des transactions dans celles-ci.

Avantages liés aux mappes dynamiques

L'introduction des mappes dynamiques permet de ne plus devoir définir toutes les mappes avant l'initialisation. Avec les modèles de mappe, vous pouvez désormais créer des mappes après l'initialisation de la grille d'objets.

Les modèles de mappe sont définis dans le fichier XML ObjectGrid. Des comparaisons de modèle sont exécutées lorsqu'une Session demande une mappe n'ayant pas été précédemment définie. Si le nom de la nouvelle mappe correspond à l'expression régulière d'un modèle de mappe, la mappe est créée dynamiquement et elle reçoit le nom de la mappe demandée. Elle hérite de tous les paramètres du modèle de mappe définis dans le fichier XML ObjectGrid.

Création de mappes dynamiques

La création de mappes dynamiques est étroitement associée à la méthode Session.getMap(String). Les appels de cette méthode renvoient une ObjectMap basée sur la mappe de sauvegarde qui a été configurée par le fichier XML ObjectGrid.

La transmission d'une chaîne qui correspond à l'expression régulière d'un modèle de mappe entraîne la création d'une ObjectMap et d'une mappe de sauvegarde associée.

Pour plus d'informations sur la méthode Session.getMap(String cacheName), voir la documentation relative aux API.

Définir un modèle de mappe dans XML est aussi simple que définir un attribut booléen de modèle pour l'élément backingMap. Lorsque le modèle est associé à la valeur true, le nom de la mappe de sauvegarde est interprété comme une expression régulière.

WebSphere eXtreme Scale utilise Java la correspondance de modèle entre expressions régulières. Pour plus d'informations sur le moteur d'expressions régulières dans Java, consultez la documentation relative aux API du package et des classes java.util.regex.

Un exemple de fichier XML ObjectGrid et de modèle de mappe défini est présenté ci-dessous.

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
 <objectGrids>
  <objectGrid name="accounting">
   <backingMap name="payroll" readOnly="false" />
   <backingMap name="templateMap.*" template="true"
    pluginCollectionRef="templatePlugins" lockStrategy="PESSIMISTIC" />
  </objectGrid>
 </objectGrids>

 <backingMapPluginCollections>
  <backingMapPluginCollection id="templatePlugins">
   <bean id="Evictor"
    className="com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor" />
  </backingMapPluginCollection>
 </backingMapPluginCollections>
</objectGridConfig>
```

Le fichier XML précédent définit un modèle de mappe et une mappe ne servant pas de modèle. Le nom du modèle de mappe est une expression régulière : `templateMap.*`. Lorsque la méthode Session.getMap(String) est appelée avec un nom de mappe correspondant à cette expression régulière, l'application crée une mappe.

Remarque : Si vous avez défini plusieurs modèles de mappe, vérifiez que le nom des arguments de la méthode Session.getMap(String) ne correspond pas à plusieurs modèles de mappe.

Exemple

Vous devez configurer un modèle de mappe pour créer une mappe dynamique. Ajoutez l'attribut booléen du modèle à une mappe de sauvegarde dans le fichier XML ObjectGrid.

```
<backingMap name="templateMap.*" template="true" />
```

Le nom du modèle de mappe est traité comme une expression régulière.

Lorsque vous appelez la méthode Session.getMap(String cacheName) avec un nom de cache correspondant à l'expression régulière, la mappe dynamique est créée. Un objet ObjectMap est renvoyé par cet appel de méthode et un objet BackingMap associé est créé.

```
Session session = og.getSession();
    ObjectMap map = session.getMap("templateMap1");
```

La mappe créée est configurée avec tous les attributs et plug-in définis dans la définition du modèle de mappe. Considérez à nouveau le fichier XML ObjectGrid précédent.

Supposons qu'un expulseur soit configuré dans une mappe dynamique créée d'après le modèle de mappe de ce fichier XML et que sa stratégie de verrouillage soit pessimiste.

Remarque : Un modèle n'est pas véritablement une mappe de sauvegarde. Autrement dit, l'ObjectGrid "accounting" ne contient aucune mappe "templateMap.*" réelle. Le modèle sert uniquement de base à la création d'une mappe dynamique. Vous devez cependant inclure la mappe dynamique dans l'élément mapRef du fichier XML de règle de déploiement portant exactement le même nom que dans le fichier XML ObjectGrid. Vous pouvez ainsi identifier dans quel mapSet les mappes dynamiques se trouvent.

Considérez le changement de comportement de la méthode Session.getMap(String cacheName) lorsque vous utilisez des modèles de mappe. Avant WebSphere eXtreme Scale version 7.0, tous les appels de la méthode Session.getMap(String cacheName) entraînaient une exception UndefinedMapException si la mappe demandée n'existait pas. Avec les mappes dynamiques, chaque nom correspondant à l'expression régulière d'un modèle de mappe entraîne une création de mappe. Vérifiez que vous notez bien le nombre de mappes créées par votre application, notamment si votre expression régulière est générique.

Par ailleurs, ObjectGridPermission.DYNAMIC_MAP est nécessaire à la création de mappes dynamiques lorsque la sécurité eXtreme Scale est activée. Cette permission est vérifiée lorsque la méthode Session.getMap(String) est appelée. Pour plus d'informations, consultez les informations relatives aux autorisations du client d'application dans la *Présentation du produit*.

Autres exemples

objectGrid.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
    xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>

   <objectGrid name="session.partition.info">
    <backingMap name="partition.info" readOnly="false" lockStrategy="PESSIMISTIC"
       ttlEvictorType="NONE" copyMode="NO_COPY" numberOfBuckets="107"/>
    <backingMap name="clone.info" readOnly="false" lockStrategy="PESSIMISTIC"
      ttlEvictorType="NONE" copyMode="NO_COPY" numberOfBuckets="107"
      lockTimeout="300"/>
   </objectGrid>

   <objectGrid name="session">
    <bean id="ObjectGridEventListener"
     className="com.ibm.ws.xs.sessionmanager.SessionHandleManager"/>
    <backingMap name="objectgrid.session.metadata"
```

```
     pluginCollectionRef="objectgrid.session.metadata.dynamicmap.*
     template= true " readOnly="false" lockStrategy="PESSIMISTIC"
     ttlEvictorType="LAST_ACCESS_TIME" copyMode="NO_COPY"/>
    <backingMap name="objectgrid.session.attribute"
     pluginCollectionRef="objectgrid.session.attribute.dynamicmap.*"
     template=true readOnly="false" lockStrategy="OPTIMISTIC"
     ttlEvictorType="NONE" copyMode="NO_COPY"/>
    <backingMap name="datagrid.session.global.ids" readOnly="false"
     lockStrategy="PESSIMISTIC" ttlEvictorType="NONE" copyMode="NO_COPY"/>
   </objectGrid>

  </objectGrids>
</objectGridConfig>
```

objectGridDeployment.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy
   ../deploymentPolicy.xsd"
  xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">

  <objectgridDeployment objectgridName="session.partition.info">
    <mapSet name="endPointMapSet" numberOfPartitions="5"
      minSyncReplicas="0" maxSyncReplicas="1" maxAsyncReplicas="0"
      developmentMode="false" placementStrategy="FIXED_PARTITIONS">
     <map ref="partition.info"/>
     <map ref="clone.info"/>
    </mapSet>
  </objectgridDeployment>

  <objectgridDeployment objectgridName="session">
    <mapSet name="mapSet2" numberOfPartitions="5" minSyncReplicas="0"
      maxSyncReplicas="0" maxAsyncReplicas="1" developmentMode="false"
      placementStrategy="PER_CONTAINER">
     <map ref="logical.name"/>
     <map ref="objectgrid.session.metadata.dynamicmap.*"/>
     <map ref="objectgrid.session.attribute.dynamicmap.*"/>
     <map ref="datagrid.session.global.ids"/>
    </mapSet>
  </objectgridDeployment>

</deploymentPolicy>
```

Limitations et considérations:

Limitations :

- Vous ne pouvez pas utiliser les mappes dynamiques avec Query.
- Le QuerySchema ne prend pas en charge le modèle de mapName.
- Vous ne pouvez pas utiliser les entités avec les mappes dynamiques.
- Une entité BackingMap est définie de façon implicite et mappée vers l'entité via le nom de classe.

Considérations :

- De nombreux plug-in n'ont aucun moyen de déterminer la mappe à laquelle ils sont associés.
- Les autres plug-in se différencient les uns des autres en utilisant un nom de mappe ou de mappe de sauvegarde en tant qu'argument.

ObjectMap et JavaMap

Une instance JavaMap est obtenue à partir d'un objet ObjectMap. L'interface JavaMap possède les mêmes signatures de méthode qu'ObjectMap, mais avec un traitement des exceptions différent. JavaMap étend l'interface java.util.Map, pour que toutes les exceptions soient des instances de la classe java.lang.RuntimeException. JavaMap étendant l'interface java.util.Map, il est facile d'utiliser rapidement WebSphere eXtreme Scale avec une application existante qui utilise une interface java.util.Map pour la mise en cache des objets.

Obtention d'une instance JavaMap

Une application extrait une instance JavaMap d'un objet ObjectMap à l'aide de la méthode ObjectMap.getJavaMap. Le fragment de code suivant montre comment obtenir une instance JavaMap.

```
ObjectGrid objectGrid = ...;
BackingMap backingMap = objectGrid.defineMap("mapA");
Session sess = objectGrid.getSession();
ObjectMap objectMap = sess.getMap("mapA");
java.util.Map map = objectMap.getJavaMap();
JavaMap javaMap = (JavaMap) javaMap;
```

Une instance JavaMap est sauvegardée par l'objet ObjectMap à partir duquel elle a été obtenue. Si vous appelez plusieurs fois la méthode getJavaMap à l'aide d'un objet ObjectMap particulier, la même instance JavaMap est renvoyée.

Méthodes

L'interface JavaMap ne prend en charge qu'un sous-ensemble des méthodes de l'interface java.util.Map. L'interface java.util.Map prend en charge les méthodes suivantes :

Méthode containsKey(java.lang.Object)

Méthode get(java.lang.Object)

Méthode put(java.lang.Object, java.lang.Object)

Méthode putAll(java.util.Map)

Méthode remove(java.lang.Object)

clear()

Toutes les autres méthodes héritées de l'interface java.util.Map génèrent une exception java.lang.UnsupportedOperationException.

Mappes comme files d'attente FIFO

Avec WebSphere eXtreme Scale, vous pouvez fournir une fonctionnalité FIFO (premier entré, premier sorti) similaire aux files d'attente pour toutes les mappes. WebSphere eXtreme Scale recherche l'ordre d'insertion de toutes les mappes. Un client peut demander à une mappe sa prochaine entrée déverrouillée, suivant l'ordre d'insertion, et verrouiller cette entrée. Ce processus permet à plusieurs clients d'utiliser les entrées de la mappe de manière efficace.

Exemple FIFO

Le fragment de code ci-après montre un client qui entre dans une boucle pour traiter les entrées de la mappe jusqu'à épuisement de cette dernière. La boucle démarre une transaction, puis appelle la méthode ObjectMap.getNextKey(5000).

Cette méthode renvoie la clé de la prochaine entrée déverrouillée disponible et la verrouille. Si la transaction est bloquée pour plus de 5000 millisecondes, la méthode renvoie la valeur null.

```
Session session = ...;
ObjectMap map = session.getMap("xxx");
// cela doit être défini quelque part pour arrêter cette boucle
boolean timeToStop = false;

while(!timeToStop)
{
  session.begin();
  Object msgKey = map.getNextKey(5000);
  if(msgKey == null)
  {
    // la partition actuelle est épuisée ; appelez-la de nouveau dans
    // une nouvelle transaction pour passer à la partition suivante
    session.rollback();
    continue;
  }
  Message m = (Message)map.get(msgKey);
  // consommez maintenant le message
  ...
  // doit être supprimé
  map.remove(msgKey);
  session.commit();
}
```

Mode local versus mode client

Si l'application utilise un coeur local (il ne s'agit pas d'un client), le mécanisme fonctionne comme décrit précédemment.

Pour le mode client, si la machine virtuelle Java est un client, ce dernier se connecte d'abord à un fragment primaire aléatoire de partition. S'il n'existe pas de travaux dans cette partition, le client passe à la partition suivante pour en rechercher. Le client trouve une partition contenant des entrées ou boucle sur la partition initiale aléatoire. Dans ce second cas, il renvoie une valeur null à l'application. Si le client trouve une partition avec une mappe qui contient des entrées, il utilise ces dernières jusqu'à ce qu'aucune entrée ne soit disponible pour le délai d'expiration. Une fois le délai d'expiration dépassé, la valeur null est renvoyée. Cette action signifie que si la valeur null est renvoyée et qu'une mappe partitionnée est utilisée, vous devez démarrer une nouvelle transaction et reprendre l'écoute. L'exemple de fragment de code précédent possède ce comportement.

Exemple

Si votre système fonctionne comme client et qu'une clé est renvoyée, cette transaction est maintenant associée à la partition avec l'entrée de cette clé. Si vous ne souhaitez pas mettre à jour d'autres mappes lors de cette transaction, cela ne pose aucun problème. Si vous souhaitez en mettre à jour, vous ne pouvez mettre à jour que les mappes de la même partition que la mappe dont vous avez extrait la clé. L'entrée renvoyée par la méthode getNextKey doit fournir à l'application un moyen de détecter les données appropriées dans cette partition. Soit par exemple deux mappes ; une pour les événements et l'autre pour les travaux sur lesquels ces événements ont un impact. Vous définissez les deux mappes avec les entités suivantes :

Job.java

```
package tutorial.fifo;
```

```
import com.ibm.websphere.projector.annotations.Entity;
import com.ibm.websphere.projector.annotations.Id;

@Entity
public class Job
{
 @Id String jobId;

 int jobState;
}
```

JobEvent.java

```
package tutorial.fifo;

import com.ibm.websphere.projector.annotations.Entity;
import com.ibm.websphere.projector.annotations.Id;
import com.ibm.websphere.projector.annotations.OneToOne;

@Entity
public class JobEvent
{
 @Id String eventId;
 @OneToOne Job job;
}
```

Le travail possède un ID et un état, qui est un entier. Supposons que vous souhaitez incrémenter l'état chaque fois qu'un événement arrive. Les événements sont stockés dans la mappe JobEvent. Chaque entrée possède une référence au travail concerné par l'événement. Le code permettant au programme d'écoute d'effectuer cette opération ressemble au suivant :

JobEventListener.java

```
package tutorial.fifo;

import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.em.EntityManager;

public class JobEventListener
{
 boolean stopListening;

 public synchronized void stopListening()
 {
  stopListening = true;
 }

 synchronized boolean isStopped()
 {
  return stopListening;
 }

 public void processJobEvents(Session session)
  throws ObjectGridException
 {
  EntityManager em = session.getEntityManager();
  ObjectMap jobEvents = session.getMap("JobEvent");
  while(!isStopped())
  {
   em.getTransaction().begin();

   Object jobEventKey = jobEvents.getNextKey(5000);
   if(jobEventKey == null)
   {
    em.getTransaction().rollback();
    continue;
```

```
    }
    JobEvent event = (JobEvent)em.find(JobEvent.class, jobEventKey);
    // traitez l'événement ; ici nous nous contentons d'incrémenter
    // l'état du travail
    event.job.jobState++;
    em.getTransaction().commit();
   }
  }
}
```

Le programme d'écoute est démarré sur une unité d'exécution par l'application. Il est exécuté jusqu'à ce que la méthode stopListening soit appelée. La méthode processJobEvents est exécutée sur l'unité d'exécution jusqu'à ce que la méthode stopListening soit appelée. La boucle se bloque en attendant une clé d'événement de la mappe JobEvent, puis utilise l'API EntityManager pour accéder à l'objet d'événement, supprimer la référence au travail et incrémenter l'état.

L'API EntityManager ne possède pas de méthode getNextKey, contrairement à l'ObjectMap. Par conséquent, le code utilise l'ObjectMap du JobEvent pour extraire la clé. Si une mappe est utilisée avec les entités, elle ne stocke plus d'objets. Elle stocke à la place des nuplets ; un objet nuplet pour la clé et un autre pour la valeur. La méthode EntityManager.find accepte un nupplet pour la clé.

Le code permettant de créer un événement ressemble à celui de l'exemple suivant :

```
em.getTransaction().begin();
Job job = em.find(Job.class, "Job Key");
JobEvent event = new JobEvent();
event.id = Random.toString();
event.job = job;
em.persist(event); // insérez-le
em.getTransaction().commit();
```

Vous recherchez le travail de l'événement, construisez un événement, pointez ce dernier vers le travail, l'insérez dans la mappe JobEvent et validez la transaction.

Chargeurs et mappes FIFO

Si vous souhaitez assister une mappe utilisée comme une file d'attente FIFO d'un chargeur, des tâches supplémentaires peuvent être requises. Si l'ordre des entées dans la mappe n'est pas important, aucune autre tâche n'est requise. Si l'ordre est important, vous devez ajouter un numéro de séquence à tous les enregistrements insérés lorsque vous stockez les enregistrements sur le système dorsal. Le mécanisme de préchargement doit être écrit pour insérer les enregistrements au démarrage suivant cet ordre.

Mise en cache d'objets et de leurs relations (API EntityManager)

La plupart des mémoires cache utilisent des API fondées sur les mappes pour stocker les données sous la forme de paires clé-valeur. L'API ObjectMap et le cache dynamique de WebSphere Application Server, entre autres, procèdent de cette manière. Les API fondées sur les mappes connaissent toutefois des limitations. L'API EntityManager simplifie l'interaction avec le cache eXtreme Scale en vous permettant de déclarer un graphe complexe d'objets connexes et d'interagir avec celui-ci en toute simplicité.

Limitations de l'API fondée sur les mappes

Si vous utilisez une API fondée sur les mappes, par exemple le cache dynamique de WebSphere Application Server ou l'API ObjectMap, vous devez prendre en compte les limitations suivantes :

- Le cache doit utiliser la réflexion pour extraire les données des objets présents dans le cache, ce qui a un impact sur les performances.
- Deux applications ne peuvent partager le même cache si elles utilisent différents objets pour les mêmes données.
- L'utilisation de l'évolution des données est impossible car vous ne pouvez pas facilement ajouter un attribut à un objet Java mis en cache.
- Il est difficile de manipuler des graphes d'objets. L'application doit stocker des références artificielles entre les objets et les joindre manuellement.

Utilisation de l'API EntityManager

L'API EntityManager utilise l'infrastructure fondée sur les mappes existante, mais elle convertit les objets entités en blocs de données (ou inversement) avant de les stocker ou de les lire dans la mappe. Un objet entité est transformé en un bloc de données clé et en un bloc de données valeur, qui sont ensuite stockés en tant que paire clé-valeur. Un bloc de données est une matrice d'attributs primitifs.

En respectant le style de programmation POJO (Plain Old Java Object) adopté par la plupart des structures, cet ensemble d'API facilite considérablement l'utilisation d'eXtreme Scale.

Définition d'un schéma d'entité

Un ObjectGrid peut posséder un nombre quelconque de schémas d'entité logiques. Les entités sont définies à l'aide de classes Java annotées, d'un XML ou d'une combinaison d'un XML et de classes Java. Les entités définies sont ensuite enregistrées sur un serveur eXtreme Scale et associées à BackingMaps, à des index et d'autres plug-in.

Lorsque vous concevez un schéma d'entité, vous devez effectuer les tâches suivantes :

1. Définissez les entités et leurs relations.
2. Configurez eXtreme Scale.
3. Enregistrez les entités.
4. Créez des applications basées sur des entités qui interagissent avec les API EntityManager d'eXtreme Scale.

Configuration d'un schéma d'entité

Un schéma d'entité est composé d'un ensemble d'entités et des relations entre ces entités. Dans une application eXtreme Scale comportant plusieurs partitions, les restrictions et les options suivantes s'appliquent aux schémas d'entité :

- Une seule racine doit être définie pour chaque schéma d'entité. Cette racine est appelée racine du schéma.
- Toutes les entités d'un schéma donné doivent se trouver dans le même groupe de mappes, ce qui signifie que toutes les entités accessibles à partir d'une racine de schéma à l'aide de relations de clé ou d'autres relations doivent être définies dans le même groupe de mappes que la racine de schéma.
- Chaque entité ne peut appartenir qu'à un seul schéma d'entité.

- Chaque application eXtreme Scale peut avoir plusieurs schémas.

Les entités sont enregistrées avec une instance ObjectGrid avant qu'elle ne soit initialisée. Chaque entité définie doit posséder un nom unique et est automatiquement associée à une mappe de sauvegarde ObjectGrid de même nom. La méthode d'initialisation varie en fonction de la configuration que vous utilisez :

Configuration eXtreme Scale locale

Si vous utilisez un ObjectGrid local, vous pouvez configurer le schéma d'entité à l'aide d'un programme. Dans ce mode, vous pouvez utiliser les méthodes ObjectGrid.registerEntities pour enregistrer les classes d'entité annotées ou un fichier de descripteur de métadonnées d'entité.

Configuration eXtreme Scale répartie

Si vous utilisez une configuration eXtreme Scale répartie, vous devez fournir un fichier de descripteur de métadonnées d'entité avec le schéma d'entité.

Pour plus d'informations, reportez-vous à la rubrique «EntityManager dans un environnement réparti», à la page 73.

Exigences des entités

Les métadonnées d'entité sont configurées à l'aide de fichiers de classe Java et/ou d'un fichier XML de descripteur d'entité. Le XML du descripteur d'entité au moins est requis pour identifier les mappes de sauvegarde eXtreme Scale associées à des entités. Les attributs persistants de l'entité et ses relations avec les autres entités sont décrits dans une classe Java annotée (classe de métadonnées d'entité) ou dans le fichier XML du descripteur d'entité. La classe de métadonnées d'entité, si elle est spécifiée, est également utilisée par l'API EntityManager pour interagir avec les données de la grille.

7.0.0.0 FIX 2+ Une grille eXtreme Scale peut être définie sans fournir de classes d'entité. Cela peut être avantageux si le serveur et le client interagissent directement avec les données de nuplet stockées dans les mappes sous-jacentes. De telles entités sont intégralement définies dans le fichier XML du descripteur d'entité et sont appelées entités sans classe.

Entités sans classe

Les entités sans classe sont utiles s'il n'est pas possible d'inclure des classes d'application dans le chemin d'accès aux classes du serveur ou du client. De telles entités sont définies dans le fichier XML du descripteur de métadonnées d'entité, où le nom de classe de l'entité est spécifié à l'aide d'un identificateur d'entité sans classe de la forme suivante : @<identificateur d'entité>. Le symbole @ identifie l'entité comme sans classe et est utilisé pour les associations de mappage entre les entités. Pour un exemple de fichier XML de descripteur de métadonnées d'entité avec deux entités sans classe définies, voir la figure "Métadonnées d'entité sans classe".

Si un serveur ou un client eXtreme Scale n'a pas accès aux classes, il peut quand même utiliser l'API EntityManager à l'aide d'entités sans classe. Les cas d'utilisation courants sont les suivants :

- Le conteneur eXtreme Scale est hébergé sur un serveur qui n'autorise pas les classes d'application dans le chemin d'accès aux classes. Dans ce cas, les clients peuvent toujours accéder à la grille à l'aide de l'API EntityManager à partir d'un client, où les classes sont autorisées.
- Le client eXtreme Scale ne nécessite pas un accès aux classes d'entité car il utilise un client non Java tel que le service de données REST d'eXtreme Scale ou il accède aux données de nuplet de la grille à l'aide de l'API ObjectMap.

Si les métadonnées d'entité sont compatibles entre le client et le serveur, elles peuvent être créées à l'aide de classes de métadonnées d'entité et/ou d'un fichier XML.

Par exemple, la "classe d'entité par programmation" de la figure ci-après est compatible avec le code des métadonnées sans classe de la section suivante.

Classe d'entité par programmation

```
@Entity
public class Employee {
    @Id long serialNumber;
    @Basic byte[] picture;
    @Version int ver;
    @ManyToOne(fetch=FetchType.EAGER, cascade=CascadeType.PERSIST)
    Department department;
}

@Entity
public static class Department {
    @Id int number;
    @Basic String name;
    @OneToMany(fetch=FetchType.LAZY, cascade=CascadeType.ALL, mappedBy="department")
    Collection<Employee> employees;
}
```

Zones sans classe, clés et versions

Comme indiqué précédemment, les entités sans classe sont configurées intégralement dans le fichier de descripteur XML d'entité. Les entités basées sur des classes définissent leurs attributs à l'aide d'annotations, de propriétés et de champs Java. Par conséquent, les entités sans classe doivent définir une structure de clés et d'attributs dans le descripteur XML d'entité à l'aide des balises `<basic>` et `<id>`.

Métadonnées d'entité sans classe

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://ibm.com/ws/projector/config/emd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ibm.com/ws/projector/config/emd ./emd.xsd">

<entity class-name="@Employee" name="Employee">
    <attributes>
        <id name="serialNumber" type="long"/>
        <basic name="firstName" type="java.lang.String"/>
        <basic name="picture" type="[B"/>
        <version name="ver" type="int"/>
        <many-to-one
            name="department"
            target-entity="@Department"
            fetch="EAGER"">
                <cascade><cascade-persist/></cascade>
        </many-to-one>
    </attributes>
</entity>
```

```
<entity class-name="@Department" name="Department" >
    <attributes>
        <id name="number" type="int"/>
        <basic name="name" type="java.lang.String"/>
        <version name="ver" type="int"/>
        <one-to-many
            name="employees"
            target-entity="@Employee"
            fetch="LAZY"
            mapped-by="department">
            <cascade><cascade-all/></cascade>
        </one-to-many>
    </attributes>
</entity>
```

Notez que chaque entité ci-dessus contient un élément <id>. Une entité sans classe doit posséder un ou plusieurs éléments <id> ou une association à valeur unique qui représente la clé de l'entité. Les champs de l'entité sont représentées par des éléments <basic>. Les éléments <id>, <version> et <basic> requièrent un nom et un type dans les entités sans classe. Pour des détails sur les types pris en charge, reportez-vous à la section sur les types d'attribut pris en charge.

Exigences des classes d'entité

Les entités basées sur des classes sont identifiées en associant diverses métadonnées à une classe Java. Les métadonnées peuvent être spécifiées à l'aide d'annotations Java Platform, Standard Edition 5 et/ou d'un fichier descripteur de métadonnées d'entité. Les classes d'entité doivent satisfaire les critères suivants :

- L'annotation `@Entity` est spécifiée dans le fichier du descripteur XML d'entité.
- La classe possède un constructeur sans argument public ou protégé.
- Il doit s'agir d'une classe de niveau supérieur. Les interfaces et les types énumérés ne sont pas des classes d'entité valides.
- Le mot clé final ne peut pas être utilisé.
- L'héritage ne peut pas être utilisé.
- Chaque instance ObjectGrid doit posséder un nom et un type uniques.

Les entités possèdent toutes un nom et un type uniques. Si des annotations sont utilisées, le nom correspond au nom simple (short) de la classe par défaut, mais il peut être remplacé à l'aide de l'attribut de nom de l'annotation `@Entity`.

Attributs persistants

L'état persistant d'une entité est accessible par les clients et le gestionnaire d'entités à l'aide d'accesseurs aux champs (variables d'instance) ou aux propriété (style EJB). Chaque entité doit définir un accès par champ ou par propriété. Les entités annotées sont accessibles par des champs si les champs de la classe sont annotés ou accessibles par des propriétés si la méthode d'accès get de la propriété est annotée. Il n'est pas possible de mélanger les accès par champ et les accès par propriété. Si le type ne peut pas être déterminé automatiquement, l'attribut **accessType** de l'annotation `@Entity` ou le XML équivalent peut être utilisé pour identifier le type d'accès.

Zones persistantes

Les variables d'instance d'entité accessibles par champ sont accessibles directement à partir du gestionnaire d'entités et des clients. Les champs

marqués avec le modificateur transitoire ou l'annotation transitoire sont ignorés. Les champs persistants ne doivent pas contenir de modificateurs final ou static.

Propriétés persistantes

Les entités d'accès aux propriétés doivent respecter les conventions de signature des JavaBeans pour les propriétés de lecture et d'écriture. Les méthodes qui ne respectent pas les conventions des JavaBeans ou pour lesquelles l'annotation Transitoire se trouve sur la méthode d'accès get sont ignorées. Pour une propriété de type T, il doit exister une méthode d'accès get getProperty qui renvoie une valeur de type T et une méthode d'accès set setProperty(T) nulle. Pour les types booléens, la méthode d'accès get peut être exprimé sous la forme isProperty et renvoyer la valeur true ou false. Les propriétés persistantes ne peuvent pas posséder le modificateur statique.

Types d'attribut pris en charge

Les types suivants de propriété et de champ persistant sont pris en charge :

- Les types de primitive Java incluent les encapsuleurs
- java.lang.String
- java.math.BigInteger
- java.math.BigDecimal
- java.util.Date
- java.util.Calendar
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp
- byte[]
- java.lang.Byte[]
- char[]
- java.lang.Character[]
- enum

Les types d'attribut sérialisables par l'utilisateur sont pris en charge mais connaissent des limitations en termes de performances, de requête et de détection des modifications. Les données persistantes qui ne peuvent pas avoir de proxys, tels que les tableaux et les objets sérialisables par l'utilisateur, doivent être réaffectées à l'entité si elles sont modifiées.

Les attributs sérialisables sont représentés dans le fichier XML du descripteur d'entité à l'aide du nom de classe de l'objet. Si l'objet correspond à un tableau, le type de données est représenté à l'aide du format Java interne. Par exemple, si un type de données d'attribut est java.lang.Byte[][], la représentation de la chaîne est la suivante : [[Ljava.lang.Byte;

Les types sérialisables par l'utilisateur doivent respecter les meilleures pratiques suivantes :

- Implémentez des méthodes de sérialisation à hautes performances. Implémentez l'interface java.lang.Cloneable et la méthode publique clone.
- Implémentez l'interface java.io.Externalizable.
- Implémentez les méthodes equals et hashCode

Associations d'entités

Les associations d'entités bidirectionnelles et unidirectionnelles, ou relations entre les entités, peuvent être définies comme des associations one-to-one, many-to-one, one-to-many et many-to-many. Le gestionnaire d'entités convertit automatiquement les relations d'entité dans les références de clé appropriées lorsqu'il stocke les entités.

La grille eXtreme Scale est un cache de données et n'applique pas l'intégrité référentielle comme le fait une base de données. Les relations permettent les opérations de stockage et de suppression en cascade pour les entités enfant, mais elles ne détectent pas les liens rompus et ne les appliquent pas aux objets. Lors de la suppression d'un objet enfant, la référence à cet objet doit être supprimée du parent.

Si vous définissez une association bidirectionnelle entre deux entités, vous devez identifier le propriétaire de la relation. Dans une association to-many, la partie "many" de la relation est toujours la partie propriétaire. Si la propriété ne peut pas être déterminée automatiquement, l'attribut **mappedBy** de l'annotation ou son équivalent XML doit être spécifié. L'attribut **mappedBy** identifie le champ dans l'entité cible propriétaire de la relation. Cet attribut permet également d'identifier les champs en rapport lorsqu'il existe plusieurs attributs de même type et de même cardinalité.

Associations à valeur unique

Les associations one-to-one et many-to-one sont dénotées à l'aide des annotations @OneToOne et @ManyToOne ou des attributs XML équivalents. Le type d'entité cible est déterminé par le type d'attribut. L'exemple ci-après définit une association unidirectionnelle entre Person et Address. L'entité Customer possède une référence à une entité Address. Dans ce cas, il peut également s'agir d'une association many-to-one car il n'existe pas de relation inverse.

```
@Entity
public class Customer {
  @Id id;
  @OneToOne Address homeAddress;
}

@Entity
public class Address{
  @Id id
  @Basic String city;
}
```

Pour spécifier une relation bidirectionnelle entre les classes Customer et Address, ajoutez une référence à la classe Customer à partir de la classe Address et ajoutez l'annotation appropriée pour marquer la partir inverse de la relation. Comme il s'agit d'une association one-to-one, vous devez spécifier un propriétaire de la relation en utilisant l'attribut mappedBy sur l'annotation @OneToOne.

```
@Entity
public class Address{
  @Id id
  @Basic String city;
  @OneToOne(mappedBy="homeAddress") Customer customer;
}
```

Associations évaluées par des collections

Les associations one-to-one et many-to-many sont dénotées à l'aide des annotations `@OneToMany` et `@ManyToMany` ou des attributs XML équivalents. Toutes les relations "many" sont représentées à l'aide des types suivants : java.util.Collection, java.util.List ou java.util.Set. Le type d'entité cible est déterminé par le type générique de la collection, de la liste ou de l'ensemble ou explicitement en utilisant l'attribut **targetEntity** sur l'annotation `@OneToMany` ou `@ManyToMany` (ou son équivalent XML).

Dans l'exemple précédent, il n'est pas pratique d'avoir un objet d'adresse par client car de nombreux clients peuvent partager une adresse ou posséder plusieurs adresses. Il est préférable d'utiliser une association "many" :

```
@Entity
public class Customer {
  @Id id;
  @ManyToOne Address homeAddress;
  @ManyToOne Address workAddress;
}

@Entity
public class Address{
  @Id id
  @Basic String city;
  @OneToMany(mappedBy="homeAddress") Collection<Customer> homeCustomers;


  @OneToMany(mappedBy="workAddress", targetEntity=Customer.class)
  Collection workCustomers;
}
```

Dans cet exemple, il existe deux relations entre les mêmes entités : une relation d'adresse entre Home et Work. Une collection non générique est utilisée pour l'attribut **workCustomers** afin d'illustrer l'utilisation de l'attribut **targetEntity** lorsqu'aucun générique n'est disponible.

Associations sans classe

Les associations d'entités sans classe sont définies dans le fichier XML du descripteur de métadonnées d'entité, de la même manière que les associations basées sur des classes. Toutefois, l'entité cible ne pointe pas vers une classe réelle, mais vers un identificateur d'entité sans classe utilisé pour le nom de classe de l'entité.

Voici un exemple :

```
<many-to-one name="department" target-entity="@Department" fetch="EAGER">
      <cascade><cascade-all/></cascade>
</many-to-one>
<one-to-many name="employees" target-entity="@Employee" fetch="LAZY">
      <cascade><cascade-all/></cascade>
</one-to-many>
```

Clés primaires

Toutes les entités doivent posséder une clé primaire, qui peut être une clé simple (un attribut) ou composée (plusieurs attributs). Les attributs de clé sont signalés à l'aide de l'annotation Id ou définis dans le fichier XML du descripteur d'entité. Les exigences suivantes s'appliquent à ces attributs de clé :

- La valeur d'une clé primaire ne peut pas être modifiée.

- Un attribut de clé primaire doit appartenir à l'un des types suivants : encapsuleurs et type de primitive Java, java.lang.String, java.util.Date ou java.sql.Date.
- Une clé primaire peut contenir un nombre quelconque d'associations à valeur unique. L'entité cible de l'association clé primaire ne doit pas posséder d'association inverse directe ou indirecte avec l'entité source.

Les clés primaires composées peuvent éventuellement définir une classe de clé primaire. Une entité est associée à une classe de clé primaire à l'aide de l'annotation `@IdClass` ou du fichier XML de descripteur d'entité. Une annotation `@IdClass` est utilise lorsqu'elle est utilisée conjointement avec la méthode EntityManager.find.

Les classes de clé primaire sont soumises aux exigences suivantes :

- Elles doivent être publiques avec un constructeur sans argument.
- Le type d'accès de la classe de clé primaire est déterminé par l'entité qui déclare la classe de clé primaire.
- Si elles sont accessibles par des propriétés, les propriétés de la classe de clé primaire doivent être publiques ou protégées.
- Les propriétés ou les champs des clés primaires doivent correspondre aux noms et aux types d'attribut de clé définis dans l'entité qui y font référence.
- Les classes de clé primaire doivent implémenter les méthodes equals et hashCode.

Voici un exemple :

```
@Entity
@IdClass(CustomerKey.class)
public class Customer {
    @Id @ManyToOne Zone zone;
    @Id int custId;
    String name;
    ...
}

@Entity
public class Zone{
    @Id String zoneCode;
    String name;
}

public class CustomerKey {
    Zone zone;
    int custId;

    public int hashCode() {...}
    public boolean equals(Object o) {...}
}
```

Clés primaires sans classe

Les entités sans classe doivent contenir au moins un élément <id> ou une association dans le fichier XML avec l'attribut `id=true`. Voici un exemple de ces deux cas de figure :

```
<id name="serialNumber" type="int"/>
<many-to-one name="department" target-entity="@Department" id="true">
<cascade><cascade-all/></cascade>
</many-to-one>
```

A faire :
La balise XML <id-class> n'est pas prise en charge pour les entités sans classe.

Interception des champs et des proxys d'entité

Les classes d'entité et les types d'attribut modifiables pris en charge sont étendus par des classes proxy pour les entités accessibles par des propriétés et leur bytecode est étendu pour les entités Java Development Kit (JDK) 5 accessibles par champ. Tous les accès à l'entité, même par des méthodes métier internes et par les méthodes equals, doivent utiliser les méthodes appropriées d'accès par propriété ou par champ.

Les proxys et les intercepteurs de champ sont utilisés pour permettre au gestionnaire d'entités de rechercher l'état de l'entité, de déterminer si l'entité a été modifiée et d'améliorer les performances. Les intercepteurs de champ ne sont disponibles que sur les plateformes Java SE 5 si l'agent d'instrumentation des entités est configuré.

Avertissement : Si des entités d'accès aux propriétés sont utilisées, la méthode equals doit utiliser l'opérateur instanceof pour comparer l'instance actuelle à l'objet d'entrée. Toute introspection de l'objet cible doit être effectuée via les propriétés de l'objet et non via les champs eux-mêmes, car l'instance d'objet correspondra au proxy.

Fichier `emd.xsd`

Utilisez la définition de schéma XML de métadonnées d'entité pour créer un fichier XML de descripteur et définir un schéma d'entité pour WebSphere eXtreme Scale.

Pour les descriptions de chaque élément et attribut du fichier `emd.xsd`, voir les informations sur le fichier du descripteur de métadonnées d'entité, dans le *Guide d'administration*.

Fichier `emd.xsd`

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:emd="http://ibm.com/ws/projector/config/emd"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://ibm.com/ws/projector/config/emd"
    elementFormDefault="qualified" attributeFormDefault="unqualified"
    version="1.0">

    <!-- ************************************************** -->
    <xsd:element name="entity-mappings">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="description" type="xsd:string" minOccurs="0"/>
                <xsd:element name="entity" type="emd:entity" minOccurs="1" maxOccurs="unbounded" />
            </xsd:sequence>
        </xsd:complexType>
        <xsd:unique name="uniqueEntityClassName">
            <xsd:selector xpath="emd:entity" />
            <xsd:field xpath="@class-name" />
        </xsd:unique>
    </xsd:element>

    <!-- ************************************************** -->
    <xsd:complexType name="entity">
        <xsd:sequence>
            <xsd:element name="description" type="xsd:string" minOccurs="0"/>
            <xsd:element name="id-class" type="emd:id-class" minOccurs="0" />
            <xsd:element name="attributes" type="emd:attributes" minOccurs="0" />
            <xsd:element name="entity-listeners" type="emd:entity-listeners" minOccurs="0" />
            <xsd:element name="pre-persist" type="emd:pre-persist" minOccurs="0" />
            <xsd:element name="post-persist" type="emd:post-persist" minOccurs="0" />
            <xsd:element name="pre-remove" type="emd:pre-remove" minOccurs="0" />
            <xsd:element name="post-remove" type="emd:post-remove" minOccurs="0" />
            <xsd:element name="pre-invalidate" type="emd:pre-invalidate" minOccurs="0" />
            <xsd:element name="post-invalidate" type="emd:post-invalidate" minOccurs="0" />
            <xsd:element name="pre-update" type="emd:pre-update" minOccurs="0" />
            <xsd:element name="post-update" type="emd:post-update" minOccurs="0" />
            <xsd:element name="post-load" type="emd:post-load" minOccurs="0" />
```

```
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:attribute name="class-name" type="xsd:string" use="required" />
    <xsd:attribute name="access" type="emd:access-type" />
    <xsd:attribute name="schemaRoot" type="xsd:boolean" />
</xsd:complexType>

<!-- ************************************************** -->
<xsd:complexType name="attributes">
    <xsd:sequence>
        <xsd:choice>
            <xsd:element name="id" type="emd:id" minOccurs="0" maxOccurs="unbounded" />
        </xsd:choice>
        <xsd:element name="basic" type="emd:basic" minOccurs="0" maxOccurs="unbounded" />
        <xsd:element name="version" type="emd:version" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="many-to-one" type="emd:many-to-one" minOccurs="0" maxOccurs="unbounded" />
        <xsd:element name="one-to-many" type="emd:one-to-many" minOccurs="0" maxOccurs="unbounded" />
        <xsd:element name="one-to-one" type="emd:one-to-one" minOccurs="0" maxOccurs="unbounded" />
        <xsd:element name="many-to-many" type="emd:many-to-many" minOccurs="0" maxOccurs="unbounded" />
        <xsd:element name="transient" type="emd:transient" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
</xsd:complexType>

<!-- ************************************************** -->
<xsd:simpleType name="access-type">
    <xsd:restriction base="xsd:token">
        <xsd:enumeration value="PROPERTY" />
        <xsd:enumeration value="FIELD" />
    </xsd:restriction>
</xsd:simpleType>

<!-- ************************************************** -->
<xsd:complexType name="id-class">
    <xsd:attribute name="class-name" type="xsd:string" use="required" />
</xsd:complexType>

<!-- ************************************************** -->
<xsd:complexType name="id">
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:attribute name="type" type="xsd:string" />
    <xsd:attribute name="alias" type="xsd:string" use="optional" />
</xsd:complexType>

<!-- ************************************************** -->
<xsd:complexType name="transient">
    <xsd:attribute name="name" type="xsd:string" use="required" />
</xsd:complexType>

<!-- ************************************************** -->
<xsd:complexType name="basic">
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:attribute name="alias" type="xsd:string" />
    <xsd:attribute name="type" type="xsd:string" />
    <xsd:attribute name="fetch" type="emd:fetch-type" />
</xsd:complexType>

<!-- ************************************************** -->
<xsd:simpleType name="fetch-type">
    <xsd:restriction base="xsd:token">
        <xsd:enumeration value="LAZY" />
        <xsd:enumeration value="EAGER" />
    </xsd:restriction>
</xsd:simpleType>

<!-- ************************************************** -->
<xsd:complexType name="many-to-one">
    <xsd:sequence>
        <xsd:element name="cascade" type="emd:cascade-type" minOccurs="0" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:attribute name="alias" type="xsd:string" />
    <xsd:attribute name="target-entity" type="xsd:string" />
    <xsd:attribute name="fetch" type="emd:fetch-type" />
    <xsd:attribute name="id" type="xsd:boolean" />
</xsd:complexType>
<!-- ************************************************** -->
<xsd:complexType name="one-to-one">
    <xsd:sequence>
        <xsd:element name="cascade" type="emd:cascade-type" minOccurs="0" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:attribute name="alias" type="xsd:string" />
    <xsd:attribute name="target-entity" type="xsd:string" />
    <xsd:attribute name="fetch" type="emd:fetch-type" />
    <xsd:attribute name="mapped-by" type="xsd:string" />
    <xsd:attribute name="id" type="xsd:boolean" />
</xsd:complexType>
<!-- ************************************************** -->
<xsd:complexType name="one-to-many">
    <xsd:sequence>
        <xsd:element name="order-by" type="emd:order-by" minOccurs="0" />
```

```
        <xsd:element name="cascade" type="emd:cascade-type" minOccurs="0" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:attribute name="alias" type="xsd:string" />
    <xsd:attribute name="target-entity" type="xsd:string" />
    <xsd:attribute name="fetch" type="emd:fetch-type" />
    <xsd:attribute name="mapped-by" type="xsd:string" />
</xsd:complexType>

<!-- ************************************************** -->
<xsd:complexType name="many-to-many">
    <xsd:sequence>
        <xsd:element name="order-by" type="emd:order-by" minOccurs="0" />
        <xsd:element name="cascade" type="emd:cascade-type" minOccurs="0" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:attribute name="alias" type="xsd:string" />
    <xsd:attribute name="target-entity" type="xsd:string" />
    <xsd:attribute name="fetch" type="emd:fetch-type" />
    <xsd:attribute name="mapped-by" type="xsd:string" />
</xsd:complexType>

<!-- ************************************************** -->
<xsd:simpleType name="order-by">
    <xsd:restriction base="xsd:string" />
</xsd:simpleType>

<!-- ************************************************** -->
<xsd:complexType name="cascade-type">
    <xsd:sequence>
        <xsd:element name="cascade-all" type="emd:emptyType" minOccurs="0" />
        <xsd:element name="cascade-persist" type="emd:emptyType" minOccurs="0" />
        <xsd:element name="cascade-remove" type="emd:emptyType" minOccurs="0" />
        <xsd:element name="cascade-invalidate" type="emd:emptyType" minOccurs="0" />
        <xsd:element name="cascade-merge" type="emd:emptyType" minOccurs="0" />
        <xsd:element name="cascade-refresh" type="emd:emptyType" minOccurs="0" />
    </xsd:sequence>
</xsd:complexType>

<!-- ************************************************** -->
<xsd:complexType name="emptyType" />

<!-- ************************************************** -->
<xsd:complexType name="version">
      <xsd:attribute name="name" type="xsd:string" use="required"/>
      <xsd:attribute name="alias" type="xsd:string" />
      <xsd:attribute name="type" type="xsd:string" />
</xsd:complexType>

<!-- ************************************************** -->

<xsd:complexType name="entity-listeners">
    <xsd:sequence>
        <xsd:element name="entity-listener" type="emd:entity-listener" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
</xsd:complexType>

<!-- ************************************************** -->
<xsd:complexType name="entity-listener">
    <xsd:sequence>
        <xsd:element name="pre-persist" type="emd:pre-persist" minOccurs="0" />
        <xsd:element name="post-persist" type="emd:post-persist" minOccurs="0" />
        <xsd:element name="pre-remove" type="emd:pre-remove" minOccurs="0" />
        <xsd:element name="post-remove" type="emd:post-remove" minOccurs="0" />
        <xsd:element name="pre-invalidate" type="emd:pre-invalidate" minOccurs="0" />
        <xsd:element name="post-invalidate" type="emd:post-invalidate" minOccurs="0" />
        <xsd:element name="pre-update" type="emd:pre-update" minOccurs="0" />
        <xsd:element name="post-update" type="emd:post-update" minOccurs="0" />
        <xsd:element name="post-load" type="emd:post-load" minOccurs="0" />
    </xsd:sequence>
    <xsd:attribute name="class-name" type="xsd:string" use="required" />
</xsd:complexType>

<!-- ************************************************** -->
<xsd:complexType name="pre-persist">
    <xsd:attribute name="method-name" type="xsd:string" use="required" />
</xsd:complexType>

<!-- ************************************************** -->
<xsd:complexType name="post-persist">
    <xsd:attribute name="method-name" type="xsd:string" use="required" />
</xsd:complexType>

<!-- ************************************************** -->
<xsd:complexType name="pre-remove">
    <xsd:attribute name="method-name" type="xsd:string" use="required" />
</xsd:complexType>

<!-- ************************************************** -->
<xsd:complexType name="post-remove">
    <xsd:attribute name="method-name" type="xsd:string" use="required" />
```

```
    </xsd:complexType>

    <!-- ************************************************** -->
    <xsd:complexType name="pre-invalidate">
        <xsd:attribute name="method-name" type="xsd:string" use="required" />
    </xsd:complexType>

    <!-- ************************************************** -->
    <xsd:complexType name="post-invalidate">
        <xsd:attribute name="method-name" type="xsd:string" use="required" />
    </xsd:complexType>

    <!-- ************************************************** -->
    <xsd:complexType name="pre-update">
        <xsd:attribute name="method-name" type="xsd:string" use="required" />
    </xsd:complexType>

    <!-- ************************************************** -->
    <xsd:complexType name="post-update">
        <xsd:attribute name="method-name" type="xsd:string" use="required" />
    </xsd:complexType>

    <!-- ************************************************** -->
    <xsd:complexType name="post-load">
        <xsd:attribute name="method-name" type="xsd:string" use="required" />
    </xsd:complexType>

</xsd:schema>
```

EntityManager dans un environnement réparti

Vous pouvez utiliser EntityManager avec une grille locale ou dans un environnement eXtreme Scale réparti. La principale différence réside dans le mode choisi pour se connecter à cet environnement. Une fois la connexion établie, il n'existe aucune différence entre l'utilisation d'un objet Session et l'utilisation de l'API EntityManager.

Fichiers de configuration requis

Les fichiers de configuration XML suivants sont requis :

- Fichier XML descripteur d'ObjectGrid
- Fichier XML descripteur d'entité
- Fichier XML descripteur de grille ou de déploiement

Ces fichiers définissent les entités et les mappes de sauvegarde qui seront hébergées par un serveur.

Le fichier descripteur des métadonnées d'entité contient une description des entités utilisées. Vous devez au minimum définir la classe et le nom de l'entité. Si votre environnement d'exécution est un environnement Java Platform, Standard Edition 5, eXtreme Scale lit automatiquement la classe d'entités et ses annotations. Vous pouvez définir des attributs XML supplémentaires si la classe d'entités n'est associée à aucune annotation ou si vous devez remplacer les attributs de classe. Si vous enregistrez les entités sans classe, entrez toutes les informations relatives à l'entité uniquement dans le fichier XML.

Vous pouvez utiliser le fragment de code XML suivant pour définir une grille de données et ses entités. Dans ce fragment, le serveur crée une grille d'objets nommée `bookstore` et une mappe de sauvegarde associée nommée `order`. Vous avez noté que le fragment de code `objectgrid.xml` fait référence au fichier `entity.xml`. Dans ce cas, le fichier `entity.xml` contient une entité, l'entité Order.

objectgrid.xml

```
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">

 <objectGrids>
```

```
    <objectGrid name="bookstore" entityMetadataXMLFile="entity.xml">
     <backingMap name="Order"/>
    </objectGrid>
</objectGrids>

</objectGridConfig>
```

Ce fichier `objectgrid.xml` fait référence au fichier `entity.xml` avec l'attribut **entityMetadataXMLFile**. L'emplacement de ce fichier est relatif à l'emplacement du fichier `objectgrid.xml`. Voici un exemple de fichier `entity.xml` :

entity.xml

```
<entity-mappings xmlns="http://ibm.com/ws/projector/config/emd"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/projector/config/emd ./emd.xsd">
 <entity class-name="com.ibm.websphere.tutorials.objectgrid.em.
    distributed.step1.Order" name="Order"/>
</entity-mappings>
```

Cet exemple suppose que l'attribut orderNumber et le champ desc de la classe Order sont annotés de manière similaire.

Un fichier `entity.xml` sans classe équivalent se présenterait ainsi :

classless entity.xml

```
<entity-mappings xmlns="http://ibm.com/ws/projector/config/emd"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/projector/config/emd ./emd.xsd">
 <entity class-name="@Order " name="Order">
     <description>"Entity named: Order"</description>
        <attributes>
            <id name="orderNumber" type="int"/>
            <basic name="desc" type="java.lang.String"/>
        </attributes>
 </entity>
</entity-mappings>
```

Pour obtenir des informations sur le démarrage d'un serveur eXtreme Scale, consultez la section *Démarrage des processus serveur WebSphere eXtreme Scale* du manuel *Guide d'administration*, où les fichiers `deployment.xml` et `objectgrid.xml` sont tous deux utilisés pour démarrer le serveur de catalogues.

Connexion à un serveur eXtreme Scale réparti

Le code suivant active le mécanisme de connexion pour un client et un serveur installés sur le même ordinateur :

```
String catalogEndpoints="localhost:2809";
URL clientOverrideURL= new URL("file:etc/emtutorial/distributed/step1/objectgrid.xml");
ClientClusterContext clusterCtx = ogMgr.connect(catalogEndpoints, null, clientOverrideURL);
ObjectGrid objectGrid=ogMgr.getObjectGrid(clusterCtx, "bookstore");
```

Notez la référence au serveur eXtreme Scale distant. Une fois la connexion établie, vous pouvez appeler les méthodes de l'API EntityManager telles que persist, update, remove et find.

Avertissement : Si vous utilisez des entités, transmettez le fichier XML descripteur d'ObjectGrid de remplacement par les clients à la méthode connect. Si la valeur null est transmise à la propriété clientOverrideURL et que la structure des répertoires du client soit différente de celle du serveur, le client risque de ne pas parvenir à localiser le fichier XML descripteur d'ObjectGrid ou le fichier XML descripteur d'entité. Vous pouvez au moins copier ces fichiers du serveur vers le client.

Pour utiliser des entités sur le client ObjectGrid, vous deviez auparavant mettre le fichier XML ObjectGrid et le fichier XML d'entité à la disposition du client de l'une des deux façons suivantes :

1. En transmettant un fichier XML ObjectGrid de remplacement à la méthode ObjectGridManager.connect(String catalogServerAddresses, ClientSecurityConfiguration securityProps, URL overRideObjectGridXml).

```
String catalogEndpoints="myHost:2809";
URL clientOverrideURL= new URL("file:etc/emtutorial/distributed/step1/objectgrid.xml");
ClientClusterContext clusterCtx = ogMgr.connect(catalogEndpoints, null, clientOverrideURL);
ObjectGrid objectGrid=ogMgr.getObjectGrid(clusterCtx, "bookstore");
```

2. En transmettant la valeur null pour le fichier de remplacement et en vérifiant que le fichier XML ObjectGrid et le fichier XML d'entité se trouvent dans le même chemin sur le client et sur le serveur.

```
String catalogEndpoints="myHost:2809";
ClientClusterContext clusterCtx = ogMgr.connect(catalogEndpoints, null, null);
ObjectGrid objectGrid=ogMgr.getObjectGrid(clusterCtx, "bookstore");
```

7.0.0.0 FIX 2+ Que vous souhaitiez ou non utiliser des entités de sous-ensemble côté client, les fichiers XML étaient jusqu'alors nécessaires. Ils ne le sont désormais plus pour utiliser les entités telles qu'elles sont définies par le serveur. Vous pouvez simplement transmettre la valeur null comme paramètre pour overRideObjectGridXml, comme dans l'option 2 de la précédente section. Si le fichier XML ne se trouve pas sur le même chemin que sur le serveur, le client utilise la configuration d'entité du serveur.

Toutefois, si vous utilisez des entités de sous-ensemble sur le client, vous devez fournir un fichier XML ObjectGrid de remplacement, comme dans l'option 1.

Schéma côté client et côté serveur

Le schéma côté serveur définit le type de données stockées dans les mappes du serveur. Le schéma côté client est un mappage du schéma du serveur vers les objets application. Voici un exemple de schéma côté serveur :

```
@Entity
class ServerPerson
{
  @Id String ssn;
  String firstName;
  String surname;
  int age;
  int salary;
}
```

Un objet du client peut être annoté de la façon suivante :

```
@Entity(name="ServerPerson")
class ClientPerson
{
  @Id @Basic(alias="ssn") String socialSecurityNumber;
  String surname;
}
```

Ce client prend alors une entité côté serveur et projette le sous-ensemble de l'entité vers l'objet client. Cette projection permet d'économiser de la bande passante et de la mémoire sur le client car celui-ci obtient uniquement les informations dont il a besoin, et non la totalité des informations se trouvant dans l'entité côté serveur. Des applications différentes peuvent utiliser leurs propres objets plutôt que contraindre toutes les applications à partager un ensemble de classes en vue de l'accès aux données.

Le fichier XML descripteur d'entité côté client est requis dans les cas suivants : si le serveur s'exécute avec des entités basées sur les classes alors que le côté client s'exécute sans classe, ou l'inverse. Le mode client sans classe permet au client

d'exécuter les requêtes d'entité sans avoir accès aux classes physiques. Supposons que le serveur ait enregistré l'entité ServerPerson ci-dessus ; le client utilise alors à la place un fichier `entity.xml` :

```
<entity-mappings xmlns="http://ibm.com/ws/projector/config/emd"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/projector/config/emd ./emd.xsd">
 <entity class-name="@ServerPerson " name="Order">
     <description>"Entity named: Order"</description>
        <attributes>
            <id name="socialSecurityNumber" type="java.lang.String"/>
            <basic name="surname" type="java.lang.String"/>
        </attributes>
 </entity>
</entity-mappings>
```

Ce fichier a pour effet de créer une entité de sous-ensemble équivalente dans le client, ce qui dispense ce dernier de fournir la classe annotée réelle. Si le serveur est sans classe alors que le client ne l'est pas, le client fournit un fichier XML descripteur d'entités dans lequel la référence au fichier classe est remplacée. Ce fichier XML contient un remplacement de la référence au fichier de classes.

Référencer le schéma

Si votre application s'exécute dans Java SE 5, vous pouvez ajouter l'application aux objets à l'aide d'annotations. EntityManager peut lire le schéma à partir de celles-ci. L'application fournit à l'environnement d'exécution eXtreme Scale les références à ces objets via le fichier `entity.xml` référencé par le fichier `objectgrid.xml`. Le fichier `entity.xml` répertorie toutes les entités, chacune de celles-ci étant associée à une classe ou à un schéma. Si un nom de classe correct est indiqué, l'application tente de lire les annotations Java SE 5 dans ces classes pour identifier le schéma. Si vous n'annotez pas le fichier classe ou que vous indiquiez un identificateur sans classe en tant que nom de classe, le schéma est extrait du fichier XML. Ce fichier permet de définir les attributs, clés et relations pour chaque entité.

Les fichiers XML ne sont pas nécessaires pour une grille locale. Le programme peut obtenir une référence ObjectGrid et appeler la méthode ObjectGrid.registerEntities pour définir une liste de classes annotées Java SE 5 ou un fichier XML.

L'environnement d'exécution utilise le fichier XML ou une liste des classes annotées pour rechercher les noms d'entité, les noms et les types des attributs, les champs et les types de clés, ainsi que les relations entre les entités. Si eXtreme Scale s'exécute sur un serveur ou en mode autonome, il crée immédiatement des mappes nommées d'après le nom de chaque entité. Vous pouvez personnaliser ces mappes à l'aide du fichier `objectgrid.xml` ou des API définies par l'application ou par les structures d'injection telles que Spring.

Fichier descripteur des métadonnées d'entité

Pour plus d'informations sur le fichier descripteur des métadonnées, consultez la section «Fichier `emd.xsd`», à la page 70.

Interactions avec EntityManager

En général, les applications obtiennent d'abord une référence ObjectGrid, puis, pour chaque unité d'exécution, une session à partir de cette référence. Plusieurs unités d'exécution ne peuvent pas partager une même session. Une autre méthode, getEntityManage, peut être utilisée dans la session. Elle permet de renvoyer une référence à un gestionnaire d'entités en vue de son utilisation pour cette unité d'exécution. L'interface EntityManager peut remplacer les interfaces Session et

ObjectMap pour toutes les applications. Vous pouvez utiliser ces API EntityManager si le client a accès aux classes d'entités définies.

Obtention d'une instance EntityManager à partir d'une session

La méthode getEntityManager est disponible pour un objet Session. L'exemple de code suivante présente comment créer une instance ObjectGrid locale et comment accéder à EntityManager. Pour plus de détails sur les méthodes prises en charge, consultez la partie consacrée à l'interface EntityManager dans la documentation concernant l'API.

```
ObjectGrid og =
ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("intro-grid");
Session s = og.getSession();
EntityManager em = s.getEntityManager();
```

Une relation one-to-one existe entre l'objet Session et l'objet EntityManager. Vous pouvez utiliser l'objet EntityManager plusieurs fois.

Stockage d'une entité

Stocker une entité signifie sauvegarder l'état d'une nouvelle entité dans le cache d'une grille d'objets. Une fois la méthode persist appelée, l'entité passe à l'état géré. Persist est une opération transactionnelle et la nouvelle entité est stockée dans le cache de la grille d'objets après la validation de la transaction.

Chaque entité est associée à une mappe de sauvegarde correspondante dans laquelle des blocs de données sont stockés. La mappe de sauvegarde porte le même nom que l'entité. Elle est créée lors de l'enregistrement de la classe. L'exemple de code suivant montre comment créer un objet Order à l'aide de l'opération persist.

```
Order order = new Order(123);
em.persist(order);
order.setX();
...
```

L'objet Order est créé avec la clé 123, puis il est transmis à la méthode persist. Vous pouvez continuer à modifier l'état de l'objet tant que la transaction n'est pas validée.

Important : L'exemple précédent ne contient aucune limite transactionnelle telle que begin ou commit. Pour plus d'informations, consultez le tutoriel du gestionnaire d'entités du manuel *Présentation du produit*.

Recherche d'une entité

Pour localiser une entité dans le cache de la grille d'objets, utilisez la méthode find et fournissez une clé une fois que l'entité est stockée dans le cache. Cette méthode ne requiert aucune limite transactionnelle, ce qui est utile pour les sémantiques en lecture seule. L'exemple suivant montre qu'une seule ligne suffit pour rechercher l'entité.

```
Order foundOrder = (Order)em.find(Order.class, new Integer(123));
```

Suppression d'une entité

De même que la méthode persist, la méthode remove est une opération transactionnelle. L'exemple suivant présente les limites de la transaction appelées par les méthodes begin et commit.

```
em.getTransaction().begin();
Order foundOrder = (Order)em.find(Order.class, new Integer(123));
em.remove(foundOrder );
em.getTransaction().commit();
```

L'entité doit être gérée avant d'être supprimée. Pour cela, appelez la méthode find à l'intérieur des limites de la transaction. Appelez ensuite la méthode remove dans l'interface EntityManager.

Invalidation d'une entité

La méthode invalidate se comporte comme la méthode remove, mais elle n'appelle pas les plug-in Loader. Utilisez cette méthode pour supprimer des entités de la grille d'objets tout en les conservant dans le magasin de données dorsal.

```
em.getTransaction().begin();
Order foundOrder = (Order)em.find(Order.class, new Integer(123));
em.invalidate(foundOrder );
em.getTransaction().commit();
```

L'entité doit être gérée avant d'être invalidée. Pour cela, appelez la méthode find à l'intérieur des limites de la transaction. Une fois cette méthode appelée, vous pouvez appeler la méthode invalidate dans l'interface EntityManager.

Mise à jour d'une entité

La méthode update est également une opération transactionnelle. L'entité doit être gérée avant d'être mise à jour.

```
em.getTransaction().begin();
Order foundOrder = (Order)em.find(Order.class, new Integer(123));
foundOrder.date = new Date(); // update the date of the order
em.getTransaction().commit();
```

Dans cet exemple, la méthode persist n'est pas appelée après la mise à jour de l'entité. L'entité est mise à jour dans le cache de l'entité lors de la validation de la transaction.

Requêtes et files d'attente de requêtes

Grâce au moteur de requête très souple, vous pouvez extraire de entités à l'aide de l'API EntityManager. Créez des requêtes de type SELECT sur une entité ou un schéma basé sur un objet à l'aide du langage ObjectGrid Query. L'interface de requête explique en détail comment exécuter les requêtes à l'aide de l'API EntityManager. Pour plus d'informations sur l'utilisation des requêtes, reportez-vous à l'API Query.

Une file d'attente de requête d'entité est une structure de données semblable à une file d'attente associée à une requête d'entité. Elle sélectionne toutes les entités correspondant à la condition WHERE du le filtre de la requête et met les entités résultantes en file d'attente. Les clients peuvent alors extraire les entités de cette file d'attente de manière itérative. Pour plus d'informations, voir «Files d'attente des requêtes d'entité», à la page 90.

Programmes d'écoute d'entité et méthodes de rappel

Les applications peuvent être averties lorsqu'une entité passe d'un état à un autre. Il existe deux mécanismes de rappel pour les événements de modification d'état : les méthodes de rappel de cycle de vie définies sur une classe d'entité et appelées chaque fois que l'état de l'entité est modifié et les programmes d'écoute d'entité, qui sont plus généraux car le programme d'écoute d'entité peut être enregistré sur plusieurs entités.

Cycle de vie d'une instance d'entité

Une instance d'entité possède les états suivants :

- **New** : Une instance d'entité nouvellement créée qui n'existe pas dans le cache d'eXtreme Scale.
- **Managed** : L'instance d'entité existe dans le cache d'eXtreme Scale et est extraite ou stockée à l'aide du gestionnaire d'entités. Une entité doit être associée à une transaction active pour être à l'état Managed.
- **Detached** : L'instance d'entité existe dans le cache d'eXtreme Scale, mais elle n'est plus associée à une transaction active.
- **Removed** : L'instance d'entité est supprimée ou sa suppression du cache d'eXtreme Scale est planifiée pour lorsque la transaction sera vidée ou validée.
- **Invalidated** : L'instance d'entité est invalidée ou planifiée pour être invalidée dans le cache d'eXtreme Scale lorsque la transaction est vidée ou validée.

Si les entités passent d'un état à un autre, vous pouvez appeler des méthodes de rappel de cycle de vie.

Les sections ci-après décrivent davantage la signification des états New, Managed, Detached, Removed et Invalidated lorsque ces états s'appliquent à une entité.

Méthodes de rappel de cycle de vie

Les méthodes de rappel de cycle de vie d'entité peuvent être définies sur la classe d'entité et sont appelées lorsque l'état de l'entité est modifié. Ces méthodes sont utiles pour valider les champs d'entité et mettre à jour l'état transitoire qui n'est généralement pas stocké avec l'entité. Les méthodes de rappel de cycle de vie d'entité peuvent également être définies sur des classes qui n'utilisent pas d'entités. De telles classes sont des classes de programme d'écoute d'entité, qui peuvent être associées à plusieurs types d'entité. Les méthodes de rappel de cycle de vie d'entité peuvent être définies à l'aide d'annotations de métadonnées et d'un fichier descripteur XML de métadonnées d'entité :

- **Annotations** : Les méthodes de rappel du cycle de vie peuvent être dénotées à l'aide des annotations PrePersist, PostPersist, PreRemove, PostRemove, PreUpdate, PostUpdate et PostLoad dans une classe d'entité.
- **Descripteur XML d'entité** : Les méthodes de rappel de cycle de vie d'entité peuvent être décrites à l'aide d'un fichier XML lorsque les annotations ne sont pas disponibles.

Programmes d'écoute d'entité

Une classe de programme d'écoute d'entité est une classe qui n'utilise pas d'entités qui définissent une ou plusieurs méthodes de rappel de cycle de vie d'entité. Les programmes d'écoute d'entité sont utiles pour les applications générales d'audit ou de consignation. Ils peuvent être définis à l'aide d'annotations de métadonnées et d'un fichier descripteur XML de métadonnées d'entité :

- **Annotation** : L'annotation EntityListeners peut être utilisée pour dénoter une ou plusieurs classes de programme d'écoute d'entité sur une classe d'entité. Si plusieurs programmes d'écoute d'entité sont définis, l'ordre suivant lequel ils sont appelés est déterminé par l'ordre suivant lequel ils sont spécifiés dans l'annotation EntityListeners.
- **Descripteur XML d'entité** : Le descripteur XML peut également être utilisé pour spécifier l'ordre d'appel des programmes d'écoute d'entité ou pour remplacer l'ordre spécifié dans les annotations de métadonnées.

Exigences des méthodes de rappel

Tout sous-ensemble ou combinaison d'annotations peut être spécifié sur une classe d'entité ou une classe de programme d'écoute d'entité. Une même classe ne peut pas contenir plusieurs méthodes de rappel de cycle de vie pour un même événement de cycle de vie. Toutefois, la même méthode peut être utilisée pour plusieurs événements de rappel. La classe de programme d'écoute d'entité doit contenir un constructeur sans argument public. Les programmes d'écoute d'entité ne possèdent pas d'état. Le cycle de vie d'un programme d'écoute d'entité n'est pas spécifié. eXtreme Scale ne prenant pas en charge l'héritage des entités, les méthodes de rappel ne peuvent être définies que dans la classe d'entité, mais non dans la superclasse.

Signature de méthode de rappel

Les méthodes de rappel de cycle de vie d'entité peuvent être définies sur une classe de programme d'écoute d'entité et/ou directement sur une classe d'entité. Les méthodes de rappel de cycle de vie d'entité peuvent être définies à l'aide d'annotations de métadonnées et du descripteur XML d'entité. Les annotations utilisées pour les méthodes de rappel sur la classe entité et sur la classe de programme d'écoute d'entité sont les mêmes. Les signatures sur les méthodes de rappel sont différentes lorsqu'elles sont définies sur une classe entité et sur une classe de programme d'écoute d'entité. Les méthodes de rappel définies sur une classe d'entité ou une superclasse mappée possèdent la signature suivante :

```
void <METHOD>()
```

Les méthodes de rappel définies sur une classe de programme d'écoute d'entité possèdent la signature suivante :

```
void <METHOD>(Object)
```

L'argument Object correspond à l'instance d'entité pour laquelle la méthode de rappel est appelée. L'argument Object peut être déclaré comme objet java.lang.Object ou type d'entité réel.

Les méthodes de rappel peuvent posséder un accès de niveau public, privé, protégé ou package, mais ne doivent pas être statiques ou finales.

Les annotations suivantes sont définies pour désigner les méthodes de rappel des événements du cycle de vie des types correspondants :

- com.ibm.websphere.projector.annotations.PrePersist
- com.ibm.websphere.projector.annotations.PostPersist
- com.ibm.websphere.projector.annotations.PreRemove
- com.ibm.websphere.projector.annotations.PostRemove
- com.ibm.websphere.projector.annotations.PreUpdate
- com.ibm.websphere.projector.annotations.PostUpdate

- com.ibm.websphere.projector.annotations.PostLoad

Pour plus de détails, reportez-vous à la documentation de l'API. Pour chaque annotation un attribut XML équivalent est défini dans le fichier du descripteur XML des métadonnées d'entité.

Sémantique des méthodes de rappel du cycle de vie

Chacune des différentes méthodes de rappel du cycle de vie possède un rôle différent et est appelée dans différentes phases du cycle de vie :

PrePersist
Appelée pour une entité avant que cette dernière n'ait été rangée dans le stockage de persistance. Inclut les entités conservées en raison d'une opération de cascade. Cette méthode est appelée sur l'unité d'exécution de l'opération EntityManager.persist.

PostPersist
Appelée pour une entité après que cette dernière a été rangée dans le stockage de persistance. Inclut les entités conservées en raison d'une opération de cascade. Cette méthode est appelée sur l'unité d'exécution de l'opération EntityManager.persist. Elle est appelée une fois que l'opération EntityManager.flush ou EntityManager.commit est appelée.

PreRemove
Appelée pour une entité avant que cette dernière n'ait été supprimée. Inclut les entitées supprimées en raison d'une opération de cascade. Cette méthode est appelée sur l'unité d'exécution de l'opération EntityManager.remove.

PostRemove
Appelée pour une entité après que cette dernière n'ait été supprimée. Inclut les entitées supprimées en raison d'une opération de cascade. Cette méthode est appelée sur l'unité d'exécution de l'opération EntityManager.remove. Elle est appelée une fois que l'opération EntityManager.flush ou EntityManager.commit est appelée.

PreUpdate
Appelée pour une entité avant que cette dernière n'ait été mise à jour dans le stockage de persistance. Cette méthode est appelée sur l'unité d'exécution de l'opération de vidage des transactions ou de validation.

PostUpdate
Appelée pour une entité après que cette dernière a été mise à jour dans le stockage de persistance. Cette méthode est appelée sur l'unité d'exécution de l'opération de vidage des transactions ou de validation.

PostLoad
Appelée pour une entité après que cette dernière a été chargée à partir du stockage de persistance qui comprend les entités chargées via une association. Cette méthode est appelée sur l'unité d'exécution de l'opération de chargement, telle qu'EntityManager.find ou une requête.

Méthodes de rappel de cycle de vie en double

Si plusieurs méthodes de rappel sont définies pour un événement de cycle de vie d'entité, l'appel de ces méthodes s'effectue selon l'ordre suivant :

1. **Méthodes de rappel de cycle de vie définies dans les programmes d'écoute d'entité** : Les méthodes de rappel de cycle de vie définies dans les classes de programme d'écoute d'entité d'une classe d'entité sont appelées dans le même

ordre que la spécification des classes du programme d'écoute d'entité dans l'annotation EntityListeners ou le descripteur XML.

2. **Superclasse de programme d'écoute** : Les méthodes de rappel définies dans la superclasse du programme d'écoute d'entité sont appelées avant les enfants.
3. **Méthodes de cycle de vie d'entité** : WebSphere eXtreme Scale ne prenant pas en charge l'héritage des entités, les méthodes de cycle de vie d'entité ne peuvent être définies que dans la classe d'entités.

Exceptions

Les méthodes de rappel de cycle de vie peuvent générer des exceptions d'exécution. Si une méthode de rappel de cycle de vie génère une exception d'exécution dans une transaction, la transaction est annulée. Aucune autre méthode de rappel de cycle de vie n'est appelée après une exception d'exécution.

Exemple de programme d'écoute d'entité

Vous pouvez écrire des programmes d'écoute d'entité en fonction de vos exigences. Vous trouverez ci-après plusieurs exemples de script.

Exemple de programme d'écoute d'entité utilisant des annotations

L'exemple ci-après illustre les appels de méthode de rappel de cycle de vie et l'ordre de ces appels. Soit la classe d'entité Employee et les deux programmes d'écoute d'entité suivants : EmployeeListener et EmployeeListener2.

```
@Entity
@EntityListeners(EmployeeListener.class, EmployeeListener2.class)
public class Employee {
    @PrePersist
    public void checkEmployeeID() {
        ....
    }
}

public class EmployeeListener {
    @PrePersist
    public void onEmployeePrePersist(Employee e) {
        ....
    }
}

public class PersonListener {
    @PrePersist
    public void onPersonPrePersist(Object person) {
        ....
    }
}

public class EmployeeListener2 {
    @PrePersist
    public void onEmployeePrePersist2(Object employee) {
        ....
    }
}
```

Si un événement PrePersist se produit sur une instance Employee, les méthodes suivantes sont appelées dans l'ordre :

1. Méthode onEmployeePrePersist
2. Méthode onPersonPrePersist
3. Méthode onEmployeePrePersist2

4. Méthode checkEmployeeID

Exemple de programme d'écoute d'entité utilisant XML

L'exemple suivant explique comment définir un programme d'écoute d'entité sur une entité à l'aide du fichier XML du descripteur d'entité :

```
<entity
    class-name="com.ibm.websphere.objectgrid.sample.Employee"
    name="Employee" access="FIELD">
    <attributes>
        <id name="id" />
        <basic name="value" />
    </attributes>
    <entity-listeners>
        <entity-listener
            class-name="com.ibm.websphere.objectgrid.sample.EmployeeListener">
            <pre-persist method-name="onListenerPrePersist" />
            <post-persist method-name="onListenerPostPersist" />
        </entity-listener>
    </entity-listeners>
    <pre-persist method-name="checkEmployeeID" />
</entity>
```

L'entité Employee est configurée avec une classe de programme d'écoute d'entité com.ibm.websphere.objectgrid.sample.EmployeeListener, pour laquelle deux méthodes de rappel de cycle de vie sont définies. La méthode onListenerPrePersist est destinée à l'événement PrePersist event et la méthode onListenerPostPersist, à l'événement PostPersist. En outre, la méthode checkEmployeeID de la classe Employee est configurée pour écouter l'événement PrePersist.

Stratégie FetchPlan de l'EntityManager

Un plan d'extraction (FetchPlan) est la stratégie utilisée par EntityManager pour extraire des objets associés si l'application doit accéder aux relations.

Exemple

Supposons que votre application comporte deux entités : Service et Employé. La relation entre l'entité Service et l'entité Employé est une relation un à plusieurs bidirectionnelle : un service comporte plusieurs employés et un employé appartient à un seul service. Etant donné que dans la plupart des cas, lorsque l'entité Service est extraite, les employés correspondants doivent être extraits, le type d'extraction de cette relation un à plusieurs sera défini comme EAGER.

Voici un fragment de code de la classe Service (Department).

```
 @Entity
public class Department {

    @Id
    private String deptId;

    @Basic
    String deptName;

  @OneToMany(fetch = FetchType.EAGER, mappedBy="department", cascade = {CascadeType.PERSIST})
  public Collection<Employee> employees;

}
```

Dans un environnement réparti, lorsqu'une application appelle `em.find(Department.class, "dept1")` pour trouver une entité Service avec la clé

"dept1", cette opération find retourne l'entité Service et toutes les relations d'extraction hâtive. Dans le cas du fragment de code précédent, il s'agit de tous les employés du service "dept1".

Dans les versions antérieures à WebSphere eXtreme Scale 6.1.0.5, l'extraction d'une entité Service et de N entités Employé provoquait N+1 transferts client-serveur car le client extrayait une entité pour un transfert client-serveur. Vous pouvez améliorer les performances avec l'extraction de ces N+1 entités en un seul transfert.

Plan d'extraction

Un plan d'extraction peut être utilisé pour personnaliser le mode d'extraction des relations hâtives en adaptant la profondeur maximale des relations. La profondeur d'extraction se substitue davantage aux relations hâtives que la profondeur spécifiée pour les relations lentes. Par défaut, la profondeur d'extraction correspond à la profondeur d'extraction maximale, ce qui implique l'extraction des relations hâtives de tous les niveaux pouvant être parcourues hâtivement depuis l'entité racine. Une relation EAGER peut être parcourue hâtivement depuis une entité racine uniquement si toutes les relations en partant de l'entité racine sont configurées comme extraites hâtivement.

Dans l'exemple précédent, l'entité Employé peut être parcourue hâtivement depuis l'entité Service car la relation Service-Employé est configurée comme étant extraite hâtivement.

Si l'entité Employé présente une autre relation hâtive avec une entité Adresse par exemple, cette dernière peut également être parcourue hâtivement depuis l'entité Service. Toutefois, si les relations Service-Employé ont été configurées en mode d'extraction lente, l'entité Adresse ne peut pas être parcourue hâtivement depuis l'entité Service car la relation Service-Employé rompt la chaîne d'extraction hâtive.

Un objet FetchPlan peut être extrait de l'instance EntityManager. L'application peut utiliser la méthode setMaxFetchDepth pour modifier la profondeur d'extraction maximale.

Un plan d'extraction est associé à une instance EntityManager. Le plan d'extraction s'applique à toutes les opérations fetch, tel que décrit ci-dessous de manière plus spécifique.

- Opérations EntityManager `find(Class class, Object key)` et `findForUpdate(Class class, Object key)`
- Opération query
- Opérations QueryQueue

L'objet FetchPlan est modifiable. Une fois modifiée, la valeur est appliquée aux opérations fetch exécutées ultérieurement.

Un plan d'extraction est important dans un environnement réparti car il décide si les entités de relation d'extraction hâtive sont extraites avec l'entité racine dans un ou plusieurs transferts client-serveur.

En reprenant l'exemple précédent, considérez que la profondeur maximale du plan d'extraction est infinie. Dans ce cas, lorsqu'une application appelle `em.find(Department.class, "dept1")` pour trouver une entité Service, cette opération find retourne l'entité Service et N entités Employé dans un transfert client-serveur. Toutefois, pour un plan d'extraction doté d'une profondeur

d'extraction maximale égale à zéro, seul l'objet Service est extrait du serveur alors que les entités Employé sont extraites du serveur uniquement lors de l'accès à la collection d'employés de l'objet Service.

Plans d'extraction différents

Plusieurs plans d'extraction sont disponibles en fonction de vos besoins ; ils sont détaillés dans les sections suivantes.

Impact sur une grille répartie

- *Plan d'extraction à profondeur infinie :* la profondeur d'extraction maximale d'un plan d'extraction à profondeur infinie est définie sur `FetchPlan.DEPTH_INFINITE`.

  Dans un environnement client-serveur, si un plan d'extraction à profondeur infinie est utilisé, toutes les relations pouvant être parcourues hâtivement depuis l'entité racine sont extraites dans un transfert client-serveur.

  Exemple : si l'application s'intéresse à toutes les entités Adresse de tous les employés d'un Service particulier, elle utilise un plan d'extraction à profondeur infinie pour extraire toutes les entités Adresse associées. Le code suivant implique uniquement un transfert client-serveur.

  ```
  em.getFetchPlan().setMaxFetchDepth(FetchPlan.DEPTH_INFINITE);

  tran.begin();
  Department dept = (Department) em.find(Department.class, "dept1");
  // Effectuez une action avec l'objet Address.
  for (Employee e: dept.employees) {
      for (Address addr: e.addresses) {
      // effectuez une action avec les adresses.
      }
  }
  tran.commit();
  ```

- *Plan d'extraction à profondeur égale à zéro :* la profondeur d'extraction maximale d'un plan d'extraction à profondeur égale à zéro est définie sur 0.

  Dans un environnement client-serveur, si un plan d'extraction à profondeur égale à zéro est utilisé, seule l'entité racine est extraite dans le premier transfert client-serveur. Toutes les relations hâtives sont traitées si elles ont été lentes.

  Exemple : dans cet exemple, l'application s'intéresse uniquement à l'attribut entité Service. Elle n'a pas besoin d'accéder aux employés correspondants et définit donc la profondeur du plan d'extraction sur 0.

  ```
  Session session = objectGrid.getSession();
  EntityManager em = session.getEntityManager();
  EntityTransaction tran = em.getTransaction();
  em.getFetchPlan().setMaxFetchDepth(0);

  tran.begin();
  Department dept = (Department) em.find(Department.class, "dept1");
  // effectuez une action avec l'objet dept.
  tran.commit();
  ```

- *Plan d'extraction à profondeur k :*

  Parler de plan d'extraction à profondeur *k*- signifie que la profondeur maximale des extractions réalisées par ce plan est de *k*.

  Dans un environnement client-serveur eXtreme Scale, si un plan d'extraction à profondeur *k*-est utilisé, toutes les relations pouvant être parcourues hâtivement depuis l'entité racine en *k* étapes sont extraites dans le premier transfert client-serveur.

  Le plan d'extraction à profondeur infinie (*k* = infini) et le plan d'extraction à profondeur égale à zéro (*k* = 0) sont seulement deux exemples de plan d'extraction à profondeur *k*-.

  Pour développer l'exemple précédent, supposons qu'une autre relation hâtive existe entre l'entité Employé et l'entité Adresse. Si la profondeur d'extraction

maximale du plan d'extraction est égale à 1, l'opération `em.find(Department.class, "dept1")` extrait l'entité Service et toutes les entités Employé dans un transfert client-serveur. Toutefois, les entités Adresse ne sont pas extraites car elles ne peuvent pas être parcourues hâtivement dans l'entité Service en une seule étape ; elles requièrent deux étapes.

Si la profondeur d'extraction maximale d'un plan d'extraction est égale à 2, l'opération `em.find(Department.class, "dept1")` extrait l'entité Service, toutes les entités Employé et toutes les entités Adresse associées aux entités Employés dans un transfert client-serveur.

Conseil : La profondeur d'extraction maximale du plan d'extraction par défaut est infinie de sorte que le comportement par défaut d'une opération fetch peut changer. Toutes les relations pouvant être parcourues hâtivement depuis l'entité racine sont extraites. Avec le plan d'extraction par défaut, l'opération fetch peut uniquement s'effectuer en un seul transfert client-serveur et non en plusieurs. Pour conserver les paramètres de produits de la version précédente, définissez la profondeur d'extraction sur 0.

- *Plan d'extraction utilisé sur la requête :*

  Si vous exécutez une requête d'entité, vous pouvez également utiliser un plan d'extraction pour personnaliser l'extraction des relations.

  Par exemple, le résultat de la requête `SELECT d FROM Department d WHERE "d.deptName='Department'"` présente une relationà l'entité Service. Notez que la profondeur du plan d'extraction commence avec l'association du résultat de la requête, dans ce cas, l'entité Service et non le résultat de la requête lui-même. L'entité Service est au niveau de profondeur d'extraction 0. Par conséquent, un plan d'extraction doté d'une profondeur d'extraction maximale de 1 extrait l'entité Service et les entités Employé correspondantes dans un seul transfert client-serveur.

  Exemple : dans cet exemple, la profondeur du plan d'extraction est définie sur 1, de sorte que l'entité Service et les entités Employé sont extraites dans un transfert client-serveur mais les entités Adresse ne sont pas extraites dans le même transfert.

Important : Si une relation est demandée, à l'aide de l'annotation ou de la configuration OrderBy, elle est considérée comme une relation hâtive même si elle est configurée comme une extraction lente.

Remarques sur les performances dans un environnement réparti

Par défaut, toutes les relations pouvant être parcourues hâtivement depuis l'entité racine sont extraites en un seul transfert client-serveur. Ce mode de transfert peut améliorer les performances si toutes les relations sont amenées à être utilisées. Toutefois, dans certains scénarios d'utilisation, les relations pouvant être parcourues hâtivement depuis l'entité racine ne sont pas toutes utilisées, ce qui entraîne des frais d'exécution et une sollicitation de la bande passante en raison de l'extraction de ces entités superflues.

Dans ces cas de figure, l'application peut définir la profondeur d'extraction maximale sur une valeur faible afin de réduire la profondeur des entités à extraire en rendant lentes toutes les relations hâtives au-delà de ce niveau de profondeur. Ce paramètre peut améliorer les performances.

En reprenant l'exemple Service-Employé-Adresse précédent, par défaut, toutes les entités Adresse associées aux employés du Service "dept1" sont extraites lorsque `em.find(Department.class, "dept1")` est appelée. Si l'application n'utilise pas les

entités Adresse, elle peut définir la profondeur d'extraction maximale sur 1, de sorte que les entités Adresse ne sont pas extraites avec l'entité Service.

Impact sur les performances de l'interface EntityManager

Un environnement dans lequel toutes les applications doivent utiliser les mêmes objets d'accès aux données pour un fichier de données particulier est irréaliste. Par contre, l'interface EntityManager fournie avec WebSphere eXtreme Scale sépare les applications de l'état conservé dans son fichier de données de grille de serveurs.

Le coût d'utilisation de l'interface EntityManager n'est pas élevé et dépend du type de travail effectué. Utilisez toujours l'interface EntityManager et optimisez la logique métier primordiale une fois l'application terminée. Vous pouvez modifier tout code qui utilise des interfaces EntityManager pour qu'il utilise des mappes et des nuplets. Généralement cette modification du code peut être nécessaire pour dix pourcent du code.

Si vous utilisez des relations entre les objets, l'impact sur les performances est plus faible car une application qui utilise des mappes doit gérer ces relations de la même manière que l'interface EntityManager.

Les applications qui utilisent l'interface EntityManager n'ont pas besoin de fournir d'ObjectTransformer car l'optimisation est automatique.

Modification du code de l'interface EntityManager pour les mappes

Voici un exemple d'entité :

```
@Entity
public class Person {
 @Id
 String ssn;
 String firstName;
 @Index
 String middleName;
 String surname;
}
```

Voici un code permettant de rechercher l'entité et de la mettre à jour :

```
Person p = null;
s.begin();
p = (Person)em.find(Person.class, "1234567890");
p.middleName = String.valueOf(inner);
s.commit();
```

Voici le même code, qui utilise des mappes et des nuplets :

```
Tuple key = null;
key = map.getEntityMetadata().getKeyMetadata().createTuple();
key.setAttribute(0, "1234567890");

// Le mode de copie est toujours NO_COPY pour les mappes d'entités si
// COPY_TO_BYTES n'est pas utilisé.
// Nous devons copier le nuplet ou demander à ObjectGrid de le faire
// à notre place :
map.setCopyMode(CopyMode.COPY_ON_READ);
s.begin();
Tuple value = (Tuple)map.get(key);
```

```
value.setAttribute(1, String.valueOf(inner));
map.update(key, value);
value = null;
s.commit();
```

Ces deux fragments de code génèrent le même résultat et une application peut utiliser l'un ou l'autre.

Le second fragment de code montre comment utiliser directement les mappes et utiliser les nuplets (paires de clé/valeur). Le nuplet value possède trois attributs : firstName, middlename et surname, indexé respectivement 0, 1 et 2. Le nuplet key possède un seul attribut ; le numéro d'ID, indexé zéro. Vous pouvez voir comment les nuplets sont créés à l'aide des méthodes EntityMetadata#getKeyMetaData ou EntityMetadata#getValueMetaData. Vous devez utiliser ces méthodes pour créer des nuplets pour une entité. Vous ne pouvez pas implémenter l'interface Tuple et transmettre une instance de votre implémentation de Tuple.

Agent d'instrumentation

Vous pouvez améliorer les performances des entités accessibles par champ en activant l'agent d'instrumentation de WebSphere eXtreme Scale lorsque vous utilisez Java Development Kit (JDK) Version 1.5 ou une version ultérieure.

Activation de l'agent eXtreme Scale sur JDK Version 1.5 ou ultérieure

L'agent ObjectGrid peut être activé à l'aide d'une option de ligne de commande Java, avec la syntaxe suivante :

```
-javaagent:jarpath[=options]
```

La valeur *jarpath* correspond au chemin d'accès d'un fichier JAR (archive Java) de l'environnement d'exécution d'eXtreme Scale qui contient une classe d'agent eXtreme Scale et les classes de support, telles que les fichiers `objectgrid.jar`, `wsobjectgrid.jar`, `ogclient.jar`, `wsogclient.jar` et `ogagent.jar`. Généralement, dans un programme Java autonome ou un environnement Java Platform, Enterprise Edition qui n'exécute pas WebSphere Application Server, utilisez le fichier `objectgrid.jar` ou `ogclient.jar`. Dans un environnement WebSphere Application Server ou à plusieurs chargeurs de classes, vous devez utiliser le fichier `ogagent.jar`dans l'option d'agent de la ligne de commande Java. Spécifiez le fichier `ogagent.config` dans le chemin d'accès aux classes ou utilisez les options d'agent pour spécifier des informations supplémentaires.

Options de l'agent eXtreme Scale

config
: Remplace le nom du fichier de configuration.

include
: Spécifie ou remplace la définition du domaine d'instrumentation qui correspond à la première partie du fichier de configuration.

exclude
: Spécifie ou remplace la définition @Exclude.

fieldAccessEntity
: Spécifie ou remplace la définition @FieldAccessEntity.

trace
: Spécifie un niveau de trace. Ces niveaux peuvent être ALL, CONFIG, FINE, FINER, FINEST, SEVERE, WARNING, INFO et OFF.

trace.file
Indique l'emplacement du fichier de trace.

Le point-virgule (**;**) sert de délimiteur entre chaque option. La virgule (**,**) sert de délimiteur entre chaque élément d'une option. L'exemple suivant illustre l'option de l'agent eXtreme Scale pour un programme Java :

```
-javaagent:objectgridRoot/lib/objectgrid.jar=config=myConfigFile;
include=includedPackage;exclude=excludedPackage;
fieldAccessEntity=package1,package2
```

Fichier `ogagent.config`

Le fichier `ogagent.config` correspond au fichier de configuration d'agent eXtreme Scale désigné. Si le nom de ce fichier se trouve dans le chemin d'accès aux classes, l'agent eXtreme Scale recherche et analyse le fichier. Vous pouvez remplacer le nom de fichier désigné via l'option config de l'agent eXtreme Scale. L'exemple suivant indique comment spécifier le fichier de configuration :

```
-javaagent:objectgridRoot/lib/objectgrid.jar=config=myOverrideConfigFile
```

Un fichier de configuration d'agent eXtreme Scale se compose de deux parties :

- **Domaine de transformation :** Le domaine de transformation correspond à la première partie du fichier de configuration. Le domaine de transformation est une liste de modules et de classes inclus dans le processus de transformation des classes. Ce domaine de transformation doit inclure toutes les classes d'entités accessibles par champ et les autres classes qui y font référence. Les classes d'entités accessibles par champ et celles qui y font référence constituent le domaine de transformation. Si vous avez l'intention de spécifier des classes d'entités accessibles par champ dans le composant @FieldAccessEntity, vous n'avez pas besoin d'inclure ici de classe d'entités accessibles par champ. Le domaine de transformation doit être complet. Si ce n'est pas le cas, une exception FieldAccessEntityNotInstrumentedException risque d'être générée.
- **@Exclude :** Le marqueur @Exclude indique que les modules et les classes répertoriés après sont exclus du domaine de transformation.
- **@FieldAccessEntity :** Le marqueur @FieldAccessEntity indique que les modules et les classes répertoriés après sont des modules et des classes d'entité accessible par champ. S'il n'existe aucune ligne après le marqueur @FieldAccessEntity, son équivalent est "Aucun marqueur @FieldAccessEntity spécifié". L'agent eXtreme Scale détermine qu'aucun module et classe d'entité accessible par champ ne sont définis. S'il existe des lignes après le marqueur @FieldAccessEntity, elles représentent les packages et les classes d'entités accessibles par champ spécifiés par l'utilisateur. Par exemple, "domaine d'entités accessibles par champ". Le domaine d'entités accessibles par champ est un sous-domaine du domaine de transformation. Les packages et les classes répertoriés dans le domaine d'entités accessibles par champ font partie du domaine de transformation, même s'ils n'y sont pas répertoriés. Le marqueur @Exclude, qui répertorie les packages et les classes exclus de la transformation, n'a pas d'impact sur le domaine d'entités accessibles par champ. Si le marqueur @FieldAccessEntity est spécifié, toutes les entités accessibles par champ doivent se trouver dans ce domaine d'entités accessibles par champ. Si ce n'est pas le cas, une exception FieldAccessEntityNotInstrumentedException peut être générée.

Exemple de fichier de configuration d'agent (ogagent.config)

```
##############################
# Le symbole # indique une ligne de commentaire
##############################
# Il s'agit d'un fichier de configuration d'agent ObjectGrid (le nom de fichier désigné est ogagent.config) qui peut être détecté et analysé par
# l'agent ObjectGrid
# s'il se trouve dans le chemin d'accès aux classes.
# Si le nom de fichier est "ogagent.config" et qu'il se trouve dans le chemin
# d'accès aux classes, le programme Java est exécuté avec -javaagent:objectgridRoot/ogagent.jar
# et l'agent ObjectGrid est activé pour ce programme.
# Si le nom de fichier n'est pas "ogagent.config", mais qu'il se trouve dans le chemin d'accès aux classes, vous pouvez le spécifier dans
# l'option de configuration de l'agent ObjectGrid
#     -javaagent:objectgridRoot/lib/objectgrid.jar=config=myOverrideConfigFile
# Voir les commentaires ci-après pour plus d'informations sur le remplacement des paramètres d'instrumentation.

# La première partie de la configuration correspond à la liste des modules et des classes à inclure dans le domaine de transformation.
# Les inclusions (packages/classes, qui constituent le domaine d'instrumentation) doivent se trouver au début du fichier.
com.testpackage
com.testClass

# Domaine de transformation : Les lignes ci-dessus correspondent aux modules/classes qui construisent le domaine de transformation.
# Le système traite les classes dont le nom commence par les modules/classes ci-dessus pour la transformation.
#
# Marqueur @Exclude : Exclusion du domaine de transformation.
# Le marqueur @Exclude indique que les modules/classes qui se trouvent après cette ligne doivent être exclus du domaine de transformation.
# Il est utilisé lorsque l'utilisateur souhaite exclure certains modules/classes des modules inclus spécifiés ci-dessus
#
# Marqueur @FieldAccessEntity : Domaine d'entités accessibles par champ.
# Le marqueur @FieldAccessEntity indique que les modules/classes qui se trouvent après cette ligne correspondent à des modules/classes d'entités
# accessibles par champ.
# S'il n'existe aucune ligne après le marqueur @FieldAccessEntity, son équivalent est "Aucun marqueur @FieldAccessEntity spécifié".
# L'environnement d'exécution considère que l'utilisateur ne spécifie pas de modules/classes d'entité accessibles par champ.
# Le "domaine d'entités accessibles par champ" est un sous-domaine du domaine de transformation.
#
# Les modules/classes répertoriés dans le "domaine d'entités accessibles par champ" font toujours partie du domaine de transformation,
# même s'ils n'y sont pas répertoriés.
# Le marqueur @Exclude, qui répertorie les modules/classes exclus de la transformation, n'a pas d'impact sur le "domaine d'entités accessibles par champ".
# Remarque : Si le marqueur @FieldAccessEntity est spécifié, toutes les entités accessibles par champ doivent se trouver dans ce domaine d'entités
# accessibles par champ ;
# sinon, une erreur FieldAccessEntityNotInstrumentedException risque de se produire.
#
# Le nom de fichier de configuration par défaut de l'agent ObjectGrid est ogagent.config
# L'environnement d'exécution recherche ce fichier comme ressource dans le chemin d'accès aux classes et le traite.
# Les utilisateurs peuvent remplacer ce nom de fichier de configuration d'agent ObjectGrid désigné via l'option config de l'agent.
#
# Par exemple :
# javaagent:objectgridRoot/lib/objectgrid.jar=config=myOverrideConfigFile
#
# La définition d'instrumentation, y compris le domaine de transformation, le marqueur @Exclude et le marqueur @FieldAccessEntity peuvent être
# remplacés de manière individuelle
# par les options d'agent désignées correspondantes.
# Les options d'agent désignées incluent :
#    include              -> utilisé pour remplacer la définition du domaine d'instrumentation qui correspond à la première partie du fichier de configuration
#    exclude              -> utilisé pour remplacer la définition @Exclude
#    fieldAccessEntity    -> utilisé pour remplacer la définition @FieldAccessEntity
#
# Chaque option d'agent doit être séparée par ";"
# Dans l'option d'agent, le package ou la classe doit être séparé par ","
#
# Voici un exemple ne remplaçant pas le nom du fichier de configuration :
#    -javaagent:objectgridRoot/lib/objectgrid.jar=include=includedPackage;exclude=excludedPackage;fieldAccessEntity=package1,package2
#
##############################

@Exclude
com.excludedPackage
com.excludedClass

@FieldAccessEntity
```

Considérations sur les performances

Pour de meilleures performances, spécifiez le domaine de transformation et le domaine d'entités accessibles par champ.

Files d'attente des requêtes d'entité

Les applications peuvent créer des files d'attente qualifiées par des requêtes sur une entité dans l'environnement eXtreme Scale local ou côté serveur. Les entités du résultat de la requête sont stockées dans cette file d'attente. Les files d'attente sont actuellement prises en charge uniquement dans les mappes utilisant la stratégie de verrouillage pessimiste.

Une même file d'attente de requêtes est partagée par plusieurs transactions et plusieurs clients. Une fois la file d'attente vide, la requête d'entité associée à cette file d'attente est exécutée à nouveau et de nouveaux résultats sont ajoutés à la file.

La file d'attente est identifiée uniquement par la chaîne et les paramètres de la requête d'entité. Une instance ObjectGrid contient une seule instance de chaque file d'attente de requête unique. Pour plus d'informations, consultez la documentation relative à l'API EntityManager.

Exemple de requête de file d'attente

L'exemple suivant présente comment utiliser une file d'attente de requêtes.

```
/**
 * Get a unassigned question type task
 */
private void getUnassignedQuestionTask() throws Exception {
    EntityManager em = og.getSession().getEntityManager();
    EntityTransaction tran = em.getTransaction();

    QueryQueue queue = em.createQueryQueue("SELECT t FROM Task t
    WHERE t.type=?1 AND t.status=?2", Task.class);
    queue.setParameter(1, new Integer(Task.TYPE_QUESTION));
    queue.setParameter(2, new Integer(Task.STATUS_UNASSIGNED));

    tran.begin();
    Task nextTask = (Task) queue.getNextEntity(10000);
    System.out.println("next task is " + nextTask);
    if (nextTask != null) {
        assignTask(em, nextTask);
    }
    tran.commit();
}
```

Dans cet exemple, une file d'attente est créée avec une chaîne de requête d'entité : "SELECT t FROM Task t WHERE t.type=?1 AND t.status=?2". Les paramètres de l'objet QueryQueue sont ensuite définis. Cette file d'attente de requêtes représente toutes les tâches "non affectées" de type "question". L'objet QueryQueue est très semblable à un objet Query d'entité.

Une fois la file d'attente créée, la transaction d'entité démarre et la méthode getNextEntity est appelée. Cette méthode extrait l'entité disponible suivante dans un délai de 10 secondes. Une fois l'entité extraite, elle est traitée par la méthode assignTask. Celle-ci modifie l'instance d'entité de tâche et lui donne le statut "affecté". Comme elle ne correspond plus au filtre de la file d'attente, elle est supprimée. Une fois affectée, la transaction est validée.

Cet exemple simple vous montre qu'une file d'attente de requête est semblable à une requête d'entité. Des différences sont toutefois à noter :

1. Il est possible d'extraire des entités de la file d'attente de manière itérative. L'application utilisateur décide du nombre d'entités à extraire. Par exemple, si vous utilisez QueryQueue.getNextEntity(timeout), une seule entité est extraite alors que si vous utilisez QueryQueue.getNextEntities(5, timeout), 5 entités le sont. Dans un environnement réparti, le nombre d'entités décide directement du nombre d'octets à transférer du serveur au client.
2. Lorsqu'une entité est extraite d'une file d'attente de requête, un verrou U est placé sur l'entité afin qu'aucune autre transaction ne puisse y accéder.

Extraction d'entités dans une boucle

Vous pouvez extraire des entités dans une boucle. L'exemple qui suit illustre comment obtenir toutes les tâches de type question non affectées.

```
/**
 * Get all unassigned question type tasks
 */
private void getAllUnassignedQuestionTask() throws Exception {
    EntityManager em = og.getSession().getEntityManager();
    EntityTransaction tran = em.getTransaction();

    QueryQueue queue = em.createQueryQueue("SELECT t FROM Task t WHERE
   t.type=?1 AND t.status=?2", Task.class);
    queue.setParameter(1, new Integer(Task.TYPE_QUESTION));
    queue.setParameter(2, new Integer(Task.STATUS_UNASSIGNED));

    Task nextTask = null;

    do {
        tran.begin();
        nextTask = (Task) queue.getNextEntity(10000);
        if (nextTask != null) {
            System.out.println("next task is " + nextTask);
        }
        tran.commit();
    } while (nextTask != null);
}
```

Si la mappe d'entités contient 10 tâches de type question non affectées, vous vous attendez à ce que ces 10 entités vont apparaître sur la console. Toutefois, lors de l'exécution de la requête, vous constatez, contrairement à vos suppositions, que le programme ne se termine jamais.

Lorsqu'une file d'attente de requête est créée et que la méthode getNextEntity est appelée, la requête d'entité associée à la file d'attente est exécutée et les 10 résultats sont ajoutés à la file d'attente. Lors de l'appel de cette méthode, une entité est retirée de la file d'attente. Après 10 appels, la file d'attente est vide. La requête d'entité est automatiquement réexécutée. Comme ces 10 entités existent toujours et qu'elles correspondent toujours aux critères de filtre de la file d'attente de la requête, elles sont à nouveau ajoutées à la file d'attente.

Si la ligne suivante est ajoutée après l'instruction println(), 10 entités seulement apparaissent.

```
em.remove(nextTask);
```

Pour plus d'informations sur l'utilisation de SessionHandle avec QueryQueue dans un déploiement de positionnement par conteneur, consultez la section Intégration de l'objet SessionHandle.

Déploiement des files d'attente de requêtes à toutes les partitions

Dans un environnement eXtreme Scale réparti, vous pouvez créer une file d'attente pour une partition ou pour toutes les partitions. Si la file d'attente est créée pour toutes les partitions, chaque partition sera associée à une instance de celle-ci.

Lorsque le client tente d'obtenir l'entité suivante à l'aide de la méthode QueryQueue.getNextEntity ou QueryQueue.getNextEntities, il envoie une requête à l'une des partitions. Un client envoie une demande d'exécution immédiate et une demande d'exécution différée au serveur :

- Avec une demande d'exécution immédiate, le client envoie une demande à une partition et le serveur répond immédiatement. Si la file d'attente contient une

entité, le serveur envoie une réponse avec l'entité. Dans le cas contraire, le serveur envoie une réponse sans entité. Dans les deux cas, le serveur répond immédiatement.

- Avec une demande d'exécution différée, le client envoie une demande à une partition et le serveur attend qu'une entité devienne disponible. Si la file d'attente contient une entité, le serveur envoie immédiatement une réponse avec l'entité. Dans le cas contraire, il attend qu'une entité devienne disponible, ou le délai d'attente de la requête est dépassé.

L'exemple suivant montre comment récupérer une entité pour une file d'attente de requête déployée sur toutes les partitions (n) :

1. Lorsque la méthode QueryQueue.getNextEntity ou QueryQueue.getNextEntities est appelée, le client sélectionne au hasard un numéro de partition compris entre 0 et n-1.
2. Le client envoie une demande d'exécution immédiate à la partition sélectionnée.
   - Si une entité est disponible, la méthode QueryQueue.getNextEntity ou QueryQueue.getNextEntities se termine en renvoyant l'entité.
   - Si une entité n'est pas disponible et que d'autres partitions non visitées existent, le client envoie une demande d'exécution immédiate à la partition suivante.
   - Si une entité n'est pas disponible et qu'aucune autre partition non visitée n'existe, le client envoie une demande d'exécution différée.
   - Si la demande d'exécution différée envoyée à la dernière partition arrive à expiration et qu'aucune donnée disponible n'existe, le client fait une dernière tentative et envoie une demande d'exécution immédiate à toutes les partitions en série. Si une entité est disponible dans les partitions précédentes, le client pourra ainsi l'obtenir.

Prise en charge des entités de sous-ensemble et des non-entités

Voici la méthode permettant de créer un objet QueryQueue dans le gestionnaire d'entités :

```
public QueryQueue createQueryQueue(String qlString, Class entityClass);
```

Le résultat en file d'attente doit être projeté vers l'objet défini par le second paramètre de la méthode, Class entityClass.

Si ce paramètre est indiqué, le nom d'entité associé à la classe doit être identique à celui indiqué dans la chaîne de requête. Cela se révèle utile si vous souhaitez projeter une entité dans une entité de sous-ensemble. Si une valeur null est associée à la classe d'entités, le résultat n'est pas projeté. La valeur stockée dans la mappe aura le format d'un bloc de format d'entité.

Collision de clé côté client

Dans un environnement eXtreme Scale réparti, les files d'attente de requête sont uniquement prises en charge pour les mappes eXtreme Scale dont le mode de verrouillage est pessimiste. Aucun cache local n'existe côté client. La mappe transactionnelle peut toutefois contenir les données (clé et valeur) d'un client. Une collision de clé peut en résulter lorsqu'une entité extraite du serveur et une entrée déjà présente dans la mappe transactionnelle partagent la même clé.

En cas de collision de clé, le client eXtreme Scale utilise la règle suivante pour émettre une exception ou pour remplacer les données en mode silencieux.

1. Si la clé concernée est la clé de l'entité indiquée dans la requête d'entité associée à la file d'attente, une exception est émise. Dans ce cas, la transaction est annulée et le verrou U de l'entité est libéré côté serveur.
2. Si la clé concernée est la clé de l'association d'entité, les données de la mappe transactionnelle sont remplacées sans avertissement.

La collision de clé produit uniquement lorsque la mappe transactionnelle contient des données. En d'autres termes, elle se produit uniquement lorsqu'un appel getNextEntity ou getNextEntities est émis dans une transaction ayant déjà été modifiée (une nouvelle donnée a été insérée ou une donnée a été mise à jour). Lorsqu'une application ne souhaite pas qu'une collision de clé se produise, elle doit appeler getNextEntity ou getNextEntities dans une transaction n'ayant pas encore été modifiée.

Echecs du client

Un échec du client peut se produire après l'envoi d'une demande getNextEntity ou getNextEntities du client au serveur, par exemple :

1. Le client envoie une demande au serveur, puis l'échec se produit.
2. Le client obtient une ou plusieurs entités du serveur, puis l'échec se produit.

Dans le premier cas, le serveur découvre que l'échec du client se produit lorsqu'il tente de renvoyer la réponse à ce dernier. Dans le second cas, lorsque le client obtient une ou plusieurs entités du serveur, un verrou X est placé sur ces entités. En cas d'échec du client, la transaction expire et le verrou X est libéré.

Requête contenant une clause ORDER BY

Les files d'attente de requête n'honorent généralement pas la clause ORDER BY. Si vous appelez getNextEntity ou getNextEntities à partir de la file d'attente, rien ne garantit que les entités seront renvoyées dans l'ordre indiqué. Il est en effet impossible de classer les entités de plusieurs partitions. Au cas où la file d'attente serait déployée sur toutes les partitions, une partition est sélectionnée au hasard en vue du traitement de la demande lorsqu'un appel getNextEntity ou getNextEntities est émis. L'ordre des entités n'est donc pas garanti.

La clause ORDER BY est honorée si une file d'attente est déployée sur une seule partition.

Pour plus d'informations, voir «API de requête EntityManager», à la page 106.

Un appel par transaction

Chaque appel à QueryQueue.getNextEntity ou à QueryQueue.getNextEntities extrait les entités correspondantes d'une partition prise au hasard. Les applications doivent n'appeler par transaction qu'un seule QueryQueue.getNextEntity ou qu'un seul QueryQueue.getNextEntities. Sinon, eXtreme Scale se retrouverait avec des entités provenant d'une multiplicité de partitions, ce qui provoquerait une exception au moment de valider la transaction.

Interface EntityTransaction

Vous pouvez utiliser l'interface EntityTransaction pour démarquer les transactions.

Rôle

Pour démarquer une transaction, vous pouvez utiliser l'interface EntityTransaction, qui est associée à une instance de gestionnaire d'entités. Utilisez la méthode EntityManager.getTransaction pour extraire l'instance EntityTransaction du gestionnaire d'entités. Chacune des instances EntityManager et EntityTransaction est associée à la session. Vus pouvez démarquer les transactions à l'aide de l'interface EntityTransaction ou de la session. Les méthodes de l'interface EntityTransaction ne contiennent pas d'exceptions vérifiées. Il ne résulte que des exceptions d'exécution de type PersistenceException ou de ses sous-classes.

Pour plus d'informations sur l'interface EntityTransaction, voir la documentation de l'APIl'interface EntityTransaction dans la documentation de l'API.

Tutoriel du gestionnaire d'entités : présentation

Le tutoriel sur le gestionnaire d'entités présente l'utilisation de WebSphere eXtreme Scale pour stocker des informations de commande sur un site Web. Vous pouvez créer une application Java Platform, Standard Edition 5 qui utilise une version d'eXtreme Scale locale en mémoire. Les entités utilisent les annotations et les valeurs génériques de Java SE 5.

Avant de commencer

Assurez-vous de respecter les exigences suivantes avant de commencer le tutoriel :

- Vous devez disposer de Java SE 5.
- Vous devez disposer du fichier `objectgrid.jar` dans le chemin d'accès aux classes.

Extraction d'entités et d'objets (API Query)

WebSphere eXtreme Scale fournit un moteur de requête flexible permettant d'extraire des entités à l'aide de l'API EntityManager et les objets Java utilisant l'API ObjectQuery.

Fonctions de requête WebSphere eXtreme Scale

Avec le moteur de requête eXtreme Scale, vous pouvez exécuter des requêtes de type SELECT sur une entité ou un schéma basé sur un objet à l'aide du langage de requête de l'eXtreme Scale.

Ce langage de requête inclut les fonctions suivantes :

- Résultats à valeur unique et valeurs multiples ;
- Fonctions d'agrégation ;
- Tri et regroupement ;
- Jointures ;
- Expressions conditionnelles avec sous-requêtes ;
- Paramètres nommés et positionnels ;
- Utilisation de l'index eXtreme Scale ;
- Syntaxe d'expression de chemin pour la navigation dans les objets ;
- Pagination.

Interface de requête

Utilisez l'interface de requête pour contrôler l'exécution des requêtes d'entité.

La méthode EntityManager.createQuery(String) permet de créer une requête. Vous pouvez faire appel à chaque instance de requête plusieurs fois à l'aide de l'instance EntityManager dans laquelle elle a été extraite.

Chaque résultat de requête produit une entité où la clé d'entité correspond à l'ID de ligne (de type long) et la valeur d'entité contient les résultats de champ de la clause SELECT. Vous pouvez utiliser chaque résultat de requête dans les requêtes ultérieures.

Les méthodes suivantes sont disponibles dans l'interface com.ibm.websphere.objectgrid.em.Query.

public ObjectMap getResultMap()

La méthode getResultMap exécute une requête SELECT et renvoie les résultats dans un objet ObjectMap avec les résultats classés dans l'ordre spécifié dans la requête. L'ObjectMap résultante est valide uniquement pour la transaction en cours.

La clé de mappe correspond au chiffe de résultat, de type long, débutant à 1. La valeur de mappe est de type com.ibm.websphere.projector.Tuple où chaque attribut et association est nommée en fonction de sa position ordinale dans la clause select de la requête. Utilisez la méthode pour extraire l'EntityMetadata pour l'objet Tuple stocké dans la mappe.

La méthode getResultMap est la méthode la plus rapide pour extraire les données de résultat de requête incluant plusieurs résultats. Vous pouvez extraire le nom de l'entité résultante à l'aide des méthodes ObjectMap.getEntityMetadata() et EntityMetadata.getName().

Exemple : la requête suivante renvoie deux lignes.

```
String ql = SELECT e.name, e.id, d from Employee e join e.dept d WHERE d.number=5
 Query q = em.createQuery(ql);
 ObjectMap resultMap = q.getResultMap();
 long rowID = 1; // démarre avec l'index 1
 Tuple tResult = (Tuple) resultMap.get(new Long(rowID));
 while(tResult != null) {
     // Le premier attribut est nom et possède un nom d'attribut 1
     // mais présente une position ordinale de 0.
     String name = (String)tResult.getAttribute(0);
     Integer id = (String)tResult.getAttribute(1);

     // Dept est une association avec un nom 3 mais
     // présente une position ordinale de 0 car il s'agit de la première association.
     // L'association est toujours une relation un à un,
     // il y existe donc uniquement une clé.
     Tuple deptKey = tResult.getAssociation(0,0);
     ...
     ++rowID;
     tResult = (Tuple) resultMap.get(new Long(rowID));

 }
```

public Iterator getResultIterator

La méthode getResultIterator exécute une requête SELECT et renvoie les résultats de la requête à l'aide d'un itérateur où chaque résultat est un objet pour une requête à valeur unique ou un tableau Objet pour une requête à plusieurs valeurs. Les valeurs du résultat Object[] sont stockées dans l'ordre de la requête. L'itérateur de résultat est valide pour la transaction en cours uniquement.

Cette méthode est privilégiée pour extraire les résultats de requête dans le contexte EntityManager. Vous pouvez utiliser la méthode facultative setResultEntityName(String) pour nommer l'entité résultant de façon à ce qu'elle soit utilisée dans des requêtes ultérieures.

Exemple : La requête suivante renvoie deux lignes.

```
String ql = SELECT e.name, e.id, e.dept from Employee e WHERE e.dept.number=5
 Query q = em.createQuery(ql);
 Iterator results = q.getResultIterator();
 while(results.hasNext()) {
     Object[] curEmp = (Object[]) results.next();
     String name = (String) curEmp[0];
     Integer id = (Integer) curEmp[1];
     Dept d = (Dept) curEmp[2];
     ...
 }
```

public Iterator getResultIterator(Class resultType)

La méthode getResultIterator(Class resultType) exécute une requête SELECT et renvoie les résultats de la requête à l'aide d'un itérateur d'entité. Le type d'entité est déterminé par le paramètre resultType. L'itérateur de résultat est valide uniquement pour la transaction en cours.

Faites appel à la méthode lorsque vous voulez utiliser les API EntityManager pour accéder aux entités résultantes.

Exemple : la requête suivante renvoie tous les employés et le service auquel ils appartiennent pour une division donnée, dans l'ordre de leur salaire. Pour imprimer les cinq employés qui ont le salaire le plus élevé, puis sélectionner le travail des employés d'un seul service dans le même ensemble de travail, utilisez le code suivant.

```
String string_ql = "SELECT e.name, e.id, e.dept from Employee e WHERE
  e.dept.division='Manufacturing' ORDER BY e.salary DESC";
Query query1 = em.createQuery(string_ql);
query1.setResultEntityName("AllEmployees");
Iterator results1 = query1.getResultIterator(EmployeeResult.class);
int curEmployee = 0;
System.out.println("Highest paid employees");
while (results1.hasNext() && curEmployee++ < 5) {
 EmployeeResult curEmp = (EmployeeResult) results1.next();
 System.out.println(curEmp);
 // Supprimez l'employé de l'ensemble de résultats.
 em.remove(curEmp);
}

// Videz les modifications dans la mappe de résultats.
em.flush();

// Exécutez une requête avec le jeu de documents local sans les employés qui ont été
// supprimés
String string_q2 = "SELECT e.name, e.id, e.dept from AllEmployees e
  WHERE e.dept.name='Hardware'";
Query query2 = em.createQuery(string_q2);
Iterator results2 = query2.getResultIterator(EmployeeResult.class);
System.out.println("Subset list of Employees");
while (results2.hasNext()) {
 EmployeeResult curEmp = (EmployeeResult) results2.next();
 System.out.println(curEmp);
}
```

public Object getSingleResult

La méthode getSingleResult exécute une requête SELECT qui renvoie un seul résultat.

Si plusieurs champs ont été définis pour la clause SELECT, il en résulte un tableau d'objets où chaque élément du tableau repose sur sa position ordinale au sein de la clause SELECT.

```
String ql = SELECT e from Employee e WHERE e.id=100"
 Employee e = em.createQuery(ql).getSingleResult();

 String ql = SELECT e.name, e.dept from Employee e WHERE e.id=100"
 Object[] empData = em.createQuery(ql).getSingleResult();
 String empName= (String) empData[0];
 Department empDept = (Department) empData[1];
```

public Query setResultEntityName(String entityName)

La méthode setResultEntityName(String entityName) spécifie le nom de l'entité de résultats de la requête.

A chaque appel des méthodes getResultIterator ou getResultMap, une entité associée à une ObjectMap est dynamiquement créée pour contenir les résultats de la requête. Si l'entité n'est pas spécifiée ou égale à null, le nom de l'entité et le nom de l'ObjectMap sont automatiquement générés.

Etant donné que tous les résultats de requête sont disponibles pour la durée d'une transaction, un nom de requête ne peut pas être réutilisé au sein d'une même transaction.

public Query setPartition(int partitionId)

Définition la partition vers laquelle la requête s'achemine.

Cette méthode est requise si les mappes de la requête sont partitionnées et si le gestionnaire d'entité n'a pas d'affinité avec une seule partition d'entité racine de schéma.

Utilisez l'interface PartitionManager pour déterminer le nombre de partitions pour la mappe de sauvegarde d'une entité donnée.

Le tableau ci-dessous décrit les autres méthodes disponibles dans l'interface de requête.

Tableau 2. Autres méthodes.

Méthode	Résultat
public Query setMaxResults(int maxResult)	Définit le nombre maximal de résultats à extraire.
public Query setFirstResult(int startPosition)	Définit la position du premier résultat à extraire.
public Query setParameter(String name, Object value)	Lie un argument à un paramètre nommé.
public Query setParameter(int position, Object value)	Lie un argument à un paramètre positionnel.

Tableau 2. Autres méthodes. (suite)

Méthode	Résultat
public Query setFlushMode(FlushModeType flushMode)	Définit le type Mode de vidage à utiliser lors de l'exécution de la requête, remplaçant le type Mode de vidage défini sur l'EntityManager.

Eléments de requête eXtreme Scale

Avec le moteur de requêteeXtreme Scale, vous pouvez utiliser un langage de requête unique pour effectuer des recherches dans le cache eXtreme Scale. Ce langage de requête peut interroger les objets Java stockés dans les objets ObjectMap et les objets Entity. Utilisez la syntaxe suivante pour créer une chaîne de requête.

Une requête eXtreme Scale consiste en une chaîne qui contient les éléments suivants :

- une clause SELECT qui indique les objets ou les valeurs à renvoyer ;
- une clause FROM qui nomme les collections d'objets ;
- une clause WHERE facultative qui contient des prédicats de recherche sur les collections ;
- une clause GROUP BY et HAVING facultative (voir la rubrique eXtreme Scale Fonctions d'agrégation de requête).
- une clause ORDER BY facultative qui indique l'ordre de la collecte des résultats.

Les collections d'objets Java sont identifiées dans des requêtes via leur nom dans la clause FROM de la requête.

Les éléments du langage de requête sont présentés plus en détail dans les rubriques connexes suivantes :

- «Formulaire BNF (Backus-Naur Form) de requête ObjectGrid», à la page 118 syntaxe
- «Référence pour les requêtes eXtreme Scale», à la page 110

Les rubriques ci-dessous décrivent les modes d'utilisation de l'API de requête :

- «API de requête EntityManager», à la page 106
- «Utilisation de l'API ObjectQuery», à la page 101

Requêtes sur des données situées dans plusieurs fuseaux horaires

Dans un scénario réparti, les requêtes s'exécutent en fait sur des serveurs. Les requêtes utilisant des prédicats de type calendar, java.util.Date et timestamp spécifient une valeur de date et/ou d'heure à partir du fuseau horaire local du serveur.

Dans un système où tous les clients et tous les serveurs tournent dans le même fuseau horaire, les types de prédicats calendar, java.util.Date et timestamp ne posent pas de problèmes particuliers. Il n'en va évidemment pas de même lorsque clients et serveurs sont situés dans des fuseaux horaires différents. La date et l'heure étant spécifiées à partir du fuseau horaire du serveur, les données retournées au client risquent de ne pas être celles qu'il faudrait. Faute de connaître

le fuseau horaire du serveur, ces dates et heures sont inexploitables. C'est pourquoi les dates et heures spécifiées doivent prendre en compte le décalage horaire entre le fuseau de la cible et celui du serveur.

Décalage horaire

Supposons, par exemple, qu'un client se trouve en zone [GMT-0] avec un serveur en zone [GMT-6]. Autrement dit, le serveur est à 6 heures de moins que le client. Le client souhaite exécuter la requête suivante :

```
SELECT e FROM Employee e WHERE e.birthDate='1999-12-31 06:00:00'
```

En supposant que l'entité Employee a un attribut birthDate de type java.util.Date, le client est en zone [GMT-0] et il veut extraire les Employees dont birthDate a une valeur '1999-12-31 06:00:00 [GMT-0]' pour son propre fuseau horaire.

La requête va s'exécuter sur le serveur et la valeur de birthDate utilisée par le moteur de requête sera '1999-12-31 06:00:00 [GMT-6]', qui est égale à '1999-12-31 12:00:00 [GMT-0]'. Les Employees dont la valeur de birthDate égale '1999-12-31 12:00:00 [GMT-0]' seront retournés au client. De ce fait, le client ne récupérera pas les Employees qu'ils souhaitent vraiment, c'est-à-dire ceux dont la valeur de birthDate est '1999-12-31 06:00:00 [GMT-0]'.

Le problème est dû à la différence de fuseaux horaires entre le client et le serveur. Pour résoudre ce problème, une possibilité est de calculer le décalage horaire entre le client et le serveur et d'appliquer ce décalage à la valeur de date et d'heure dans la requête. Dans notre exemple, le décalage horaire est de -6 heures et le prédicat birthDate, après ajustement, devra être “birthDate='1999-12-31 00:00:00'” si le client souhaite extraire les Employees dont la valeur de birthDate est '12-31 06:00:00 [GMT-0]'. Avec cette valeur ajustée, le serveur utilisera '1999-12-31 00:00:00 [GMT-6]' qui est égal à la valeur cible '12-31 06:00:00 [GMT-0]', et le client récupérera les bons Employees.

Déploiement réparti sur plusieurs fuseaux horaires

Si la grille répartie eXtreme Scale est déployée sur plusieurs serveurs ObjectGrid situés dans divers fuseaux horaires, l'approche qui consiste à ajuster le décalage horaire ne fonctionnera pas car le client sera incapable de savoir quel serveur va exécuter la requête et donc il ne pourra déterminer le décalage à utiliser. La seule solution est d'utiliser le suffixe ‘Z' (en majuscules ou en minuscules) dans le format d'échappement des dates et heures JDBC pour indiquer d'utiliser une valeur basée sur le seul fuseau horaire GMT. Le suffixe ‘Z' (en majuscules ou en minuscules) indique d'utiliser une valeur basée sur le seul fuseau horaire GMT. Sans ce suffixe, c'est la valeur basée sur le fuseau horaire local qui serait utilisée dans la requête.

La requête qui suit équivaut à l'exemple précédent, mais avec le suffixe ‘Z' :

```
SELECT e FROM Employee e WHERE e.birthDate='1999-12-31 06:00:00Z'
```

La requête va rechercher les Employees dont la valeur de birthDate est ‘1999-12-31 06:00:00'. Le suffixe ‘Z' indique que la valeur spécifiée pour birthDate est basée sur GMT, et donc que c'est la valeur de birthDate GMT ‘1999-12-31 06:00:00 [GMT-0]' qui sera utilisée par le moteur de requête comme critères de la recherche. Les Employees dont l'attribut birthDate a la valeur égale à cette valeur GMT ‘1999-12-31 06:00:00 [GMT-0]' figureront dans les résultats de la requête. L'utilisation dans les requêtes du suffixe ‘Z' dans le format d'échappement des dates et heures JDBC est capitale pour permettre aux applications de traiter en toute sécurité des problèmes de fuseaux horaires. Sans cette approche, le date et

l'heure resterait celle du fuseau horaire du serveur et n'aurait aucune signification pour le client dès lors que celui-ci se trouverait dans un autre fuseau horaire que le serveur.

Pour plus d'informations, voir dans dans la *Présentation du produit* la rubrique consacrée à l'insertion de données pour des fuseaux horaires différents.

Insertion de données pour différents fuseaux horaires

Lors de l'insertion de données associées à des attributs de calendrier, java.util.Date et d'horodatage dans un objet ObjectGrid, vous devez vous assurer que ces attributs de date et heure sont créés en fonction du même fuseau horaire, surtout en cas de déploiement sur des serveurs correspondant à des fuseaux horaires différents. L'utilisation d'objets de date et heure basés sur le même fuseau horaire permet d'éviter les problèmes liés aux fuseaux horaires divergents. En outre, les données peuvent être interrogées à l'aide de prédicats de calendrier, java.util.Date et d'horodatage.

Si un fuseau horaire n'est pas explicitement indiqué lors de la création d'objets de date et heure, Java utilise le fuseau horaire local, ce qui peut entraîner l'utilisation de valeurs de date et heure incohérentes sur les clients et les serveurs.

Prenons l'exemple d'un déploiement réparti dans lequel client1 correspond au fuseau horaire [GMT-0] et client2 au fuseau horaire [GMT-6]. Ces derniers veulent créer un objet java.util.Date avec la valeur "1999-12-31 06:00:00". client1 crée l'objet java.util.Date avec la valeur "1999-12-31 06:00:00 [GMT-0]" et client2 l'objet java.util.Date avec la valeur "1999-12-31 06:00:00 [GMT-6]". Les deux objets java.util.Date ne correspondent pas car leurs fuseaux horaires sont divergents. Un problème similaire survient lors du pré-chargement de données dans des partitions résidant sur des serveurs correspondants à des fuseaux horaires différents, si le fuseau horaire local est utilisé pour créer les objets de date et heure.

Pour éviter ce problème, l'application peut sélectionner un fuseau horaire de base, tel que [GMT-0], pour la création des objets de calendrier, java.util.Date et d'horodatage.

Pour de plus amples informations, voir la rubrique sur l'interrogation des données situées dans plusieurs fuseaux horaires le *Guide de programmation*

Utilisation de l'API ObjectQuery

L'API ObjectQuery propose des méthodes permettant d'interroger les données de l'ObjectGrid stockées à l'aide de cette même API ObjectMap. Lorsqu'un schéma est défini dans l'instance ObjectGrid, l'API ObjectQuery permet de créer et exécuter des requêtes sur les objets hétérogènes stockés dans les mappes d'objet.

Requête et mappes d'objet

Vous pouvez utiliser une fonction de requête étendue pour les objets stockés à l'aide de l'API ObjectMap. Ce type de requête permet d'extraire des objets à l'aide d'attributs non clés et d'exécuter des agrégations simples telles que des additions, des moyennes, des minima et des maxima pour toutes les données correspondant à une requête. Les applications construisent des requêtes à l'aide de la méthode Session.createObjectQuery. Cette méthode renvoie un objet ObjectQuery que vous pouvez également interroger pour obtenir les résultats de la requête. Il est aussi possible de personnaliser la requête avant de l'exécuter. La requête est exécutée

automatiquement lorsqu'une méthode renvoyant un résultat est appelée.

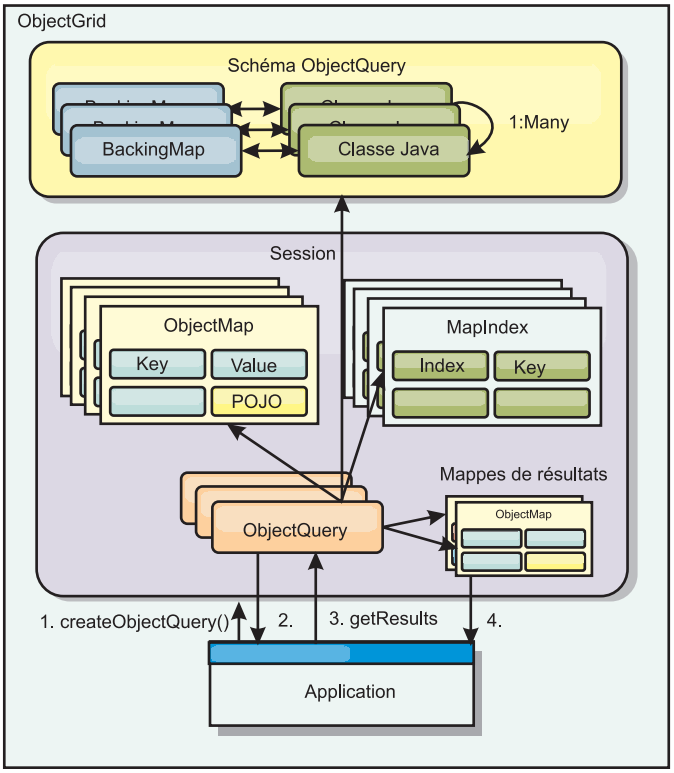


Figure 1. Interaction de la requête avec les mappes de l'objet ObjectGrid, définition d'un schéma pour les classes et association de celui-ci à une mappe ObjectGrid.

Définition d'un schéma ObjectMap

Les mappes d'objet permettent de stocker des objets sous différentes formes et ne tiennent pas compte du format. Un schéma doit être défini dans l'ObjectGrid pour déterminer le format des données. Un schéma est composé des éléments suivants :

- le type d'objet stocké dans l'ObjectMap
- les relations entre les ObjectMaps
- la méthode pour laquelle chaque requête doit accéder aux attributs de données dans les objets (méthodes par champ ou par propriété)
- nom d'attribut de la clé primaire dans l'objet

Pour plus d'informations, reportez-vous à la rubriqueConfiguration d'une ObjectQuery.

Pour obtenir un exemple de création d'un schéma à l'aide d'un programme ou du fichier XML du descripteur d'ObjectGrid, consultez le didacticiel de l'ObjectQuery dans le *Présentation du produit*.

Interrogation des objets à l'aide de l'API ObjectQuery

L'interface ObjectQuery permet l'interrogation d'objets non entité, à savoir des objets hétérogènes stockés directement dans les mappes d'objet de l'ObjectGrid. L'API ObjectQuery constitue un moyen pratique de rechercher des objets ObjectMap sans recourir directement aux mécanismes de mot clé et d'index.

Il existe deux méthodes d'extraction des résultats à partir d'une ObjectQuery : getResultIterator et getResultMap.

Extraction des résultats d'une requête à l'aide de la méthode getResultIterator

Les résultats d'une requête représentent en fait une liste d'attributs. Imaginons la requête suivante : select a,b,c from X where y=z. Une liste de lignes contenant a, b et c est renvoyée. Cette liste est stockée dans une mappe de portée transaction, ce qui signifie que vous devez associer une clé artificielle à chaque ligne et utiliser un entier qui augmente à chaque ligne. Cette mappe est obtenue à l'aide de la méthode ObjectQuery.getResultMap(). Vous pouvez accéder aux éléments de chaque ligne à l'aide de lignes de code semblables à l'exemple qui suit :

```
ObjectQuery q = session.createQuery(
    "select c.id, c.firstName, c.surname from Customer c where c.surname=?1");

  q.setParameter(1, "Claus");

  Iterator iter = q.getResultIterator();
  while(iter.hasNext())
  {
    Object[] row = (Object[])iter.next();
    System.out.println("Found a Claus with id "
      + row[objectgrid: 0 ] + ", firstName: "
      + row[objectgrid: 1 ] + ", surname: "
      + row[objectgrid: 2 ]);
  }
```

Extraction des résultats d'une requête à l'aide de getResultMap

Les résultats d'une requête peuvent également être extraits directement à l'aide de la mappe de résultats. L'exemple suivant présente comment extraire certaines parties des clients correspondant à la requête et montre comment accéder aux lignes de résultats. Notez que si vous utilisez l'objet ObjectQuery pour accéder aux données, l'identificateur généré pour la ligne de la valeur de type long est masqué. Il est visible uniquement lorsque l'ObjectMap est utilisée pour accéder au résultat.

Une fois la transaction terminée, la mappe disparaît. Celle-ci est visible uniquement par la session utilisée, c'est-à-dire, normalement, par l'unité d'exécution qui l'a créée. La mappe utilise une clé de type Long qui représente l'identificateur de la ligne. Les valeurs stockées dans la mappe sont de type Object ou Object[], où chaque élément correspond au type d'élément dans la clause select de la requête.

```
ObjectQuery q = em.createQuery(
      "select c.id, c.firstName, c.surname from Customer c where c.surname=?1");
  q.setParameter(1, "Claus");
  ObjectMap qmap = q.getResultMap();
  for(long rowId = 0; true; ++rowId)
  {
    Object[] row = (Object[]) qmap.get(new Long(rowId));
    if(row == null) break;
    System.out.println(" I Found a Claus with id " + row[0]
      + ", firstName: " + row[1]
      + ", surname: " + row[2]);
  }
```

Pour consulter des exemples d'utilisation de l'ObjectQuery, reportez-vous au didacticiel de l'API ObjectQuery dans le *Présentation du produit*.

Configuration d'un schéma ObjectQuery

ObjectQuery utilise des informations de schéma ou de forme pour effectuer une vérification sémantique et évaluer des expressions de chemin. Cette section décrit comment définir le schéma au langage XML ou à l'aide d'un programme.

Définition du schéma

Le schéma ObjectMap est défini dans le descripteur de déploiement ObjectGrid au langage XML ou à l'aide d'un programme en faisant appel aux techniques de configuration classiques d'eXtreme Scale. Pour obtenir un exemple de création de schéma, reportez-vous à la section «Configuration d'un schéma ObjectQuery»

Les informations de schéma décrivent les objets Java simples : quels sont leurs attributs, quels sont les types d'attributs, si les attributs sont des champs de clé primaire, des relations à valeur unique ou à valeurs multiples ou des relations bidirectionnelles. Les informations de schéma demandent à ObjectQuery d'utiliser l'accès par champ ou par propriété.

Attributs pouvant être interrogés

Lorsque le schéma est défini dans le descripteur d'ObjectGrid, les objets du schéma sont examinés en profondeur pour déterminer les attributs disponibles pour l'interrogation. Vous pouvez interroger les types d'attributs suivants :

- les types primitifs Java, notamment les encapsuleurs
- java.lang.String
- java.math.BigInteger
- java.math.BigDecimal
- java.util.Date
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp
- java.util.Calendar
- byte[]
- java.lang.Byte[]
- char[]
- java.lang.Character[]
- J2SE enum

Les types sérialisables imbriqués autres que ceux susmentionnés peuvent également être inclus dans un résultat de requête et non dans la clause WHERE ou FROM de la requête. Les attributs sérialisables ne peuvent pas être parcourus.

Les types d'attributs peuvent être exclus du schéma si le type n'est pas sérialisable, si le champ ou la propriété sont statiques ou si le champ est transitoire. Etant donné que tous les objets de mappe doivent être sérialisables, le descripteur d'ObjectGrid inclut uniquement les attributs qui peuvent être conservés dans l'objet. Les autres objets sont ignorés.

Attributs de zone

Lorsque le schéma est configuré pour accéder à l'objet par champ, tous les champs sérialisables non transitoires sont automatiquement incorporés au schéma. Pour

sélectionner un attribut de champ dans une requête, utilisez le nom de l'identificateur du champ tel qu'il existe dans la définition de la classe.

Tous les champs publics, privés, protégés et protégés par package sont inclus dans le schéma.

Attributs de propriété

Lorsque le schéma est configuré pour accéder à l'objet à l'aide de propriétés, toutes les méthodes sérialisables qui suivent les conventions de dénomination des propriétés JavaBeans sont automatiquement incorporées au schéma. Pour sélectionner un attribut de propriété pour la requête, utilisez les conventions de dénomination des propriétés de style JavaBeans.

Toutes les propriétés publiques, privées, protégées et protégées par package sont incluses dans le schéma.

Dans la classe ci-dessous, les attributs suivants sont ajoutés au schéma : name, birthday, valid.

```
public class Person {
  public String getName(){}
  private java.util.Date getBirthday(){}
  boolean isValid(){}
  public NonSerializableObject getData(){}
}
```

Lors de l'utilisation du mode de copie COPY_ON_WRITE de l'attribut CopyMode, le schéma de la requête doit toujours utiliser l'accès à l'aide des propriétés. Le mode COPY_ON_WRITE crée des objets proxy dès que des objets sont extraits de la mappe et peut uniquement y accéder à l'aide des méthodes de propriété. Si ces consignes ne sont pas respectées, les résultats de la requête seront nuls.

Relations

Chaque relation doit être explicitement définie dans la configuration du schéma. La cardinalité de la relation est automatiquement déterminée par le type de l'attribut. Si l'attribut implémente l'interface java.util.Collection, la relation est une relation un-à-plusieurs ou une relation plusieurs-à-plusieurs.

Contrairement aux requêtes d'entité, les attributs qui se réfèrent à d'autres objets cache ne doivent pas stocker les références directes à l'objet. Les références à d'autres objets sont sérialisées comme faisant partie intégrante des données de l'objet. Stockez plutôt la clé permettant d'accéder à l'objet lié.

Par exemple, dans le cas d'une relation plusieurs-à-un entre un client et une commande :

Incorrect. Enregistrement d'une référence à l'objet.

```
public class Customer {
  String customerId;
  Collection<Order> orders;
}

public class Order {
  String orderId;
  Customer customer;
}
```

```
Correct. Clé vers l'objet lié.

public class Customer {
  String customerId;
  Collection<String> orders;
}

public class Order {
  String orderId;
  String customer;
}
```

Lorsqu'une requête établissant une jointure entre deux objets de mappe est exécutée, la taille de la clé est automatiquement augmenté. Par exemple, la requête suivante devrait retourner les objets Customer :

```
SELECT c FROM Order o JOIN Customer c WHERE orderId=5
```

Utilisation des index

Le descripteur d'ObjectGrid utilise des plug-in d'index pour ajouter des index aux mappes. Le moteur de requête incorpore automatiquement tous les index définis pour un élément de mappage de schéma de type : com.ibm.websphere.objectgrid.plugins.index.HashIndex et la propriété rangeIndex est définie sur true. Si le type d'index n'est pas HashIndex et si la propriété rangeIndex n'est pas définie sur true, l'index n'est pas pris en compte par la requête. Pour consulter un exemple d'ajout d'un index au schéma, reportez-vous au didacticiel de requête sur les objets (Object Query tutorial) dans la *Présentation du produit*.

API de requête EntityManager

L'API EntityManager API propose des méthodes permettant d'interroger les données de la grille qui sont stockées à l'aide de cette même API. Elle permet de créer et d'exécuter des requêtes sur une ou plusieurs entités définies dans eXtreme Scale.

Requête et ObjectMaps pour les entités

WebSphere Extended Deployment v6.1 a introduit une fonction de requête étendue pour les entités stockées dans eXtreme Scale. Ce type de requête permet d'extraire des objets à l'aide d'attributs non clés et d'exécuter des agrégations simples telles que des additions, des moyennes, des minima et des maxima pour toutes les données correspondant à une requête. Les applications construisent des requêtes à l'aide de l'API EntityManager.createQuery. Un objet Query est renvoyé. Vous pouvez également l'interroger pour obtenir les résultats de la requête. Il est aussi possible de personnaliser la requête avant de l'exécuter. La requête est exécutée automatiquement lorsqu'une méthode renvoyant un résultat est appelée.

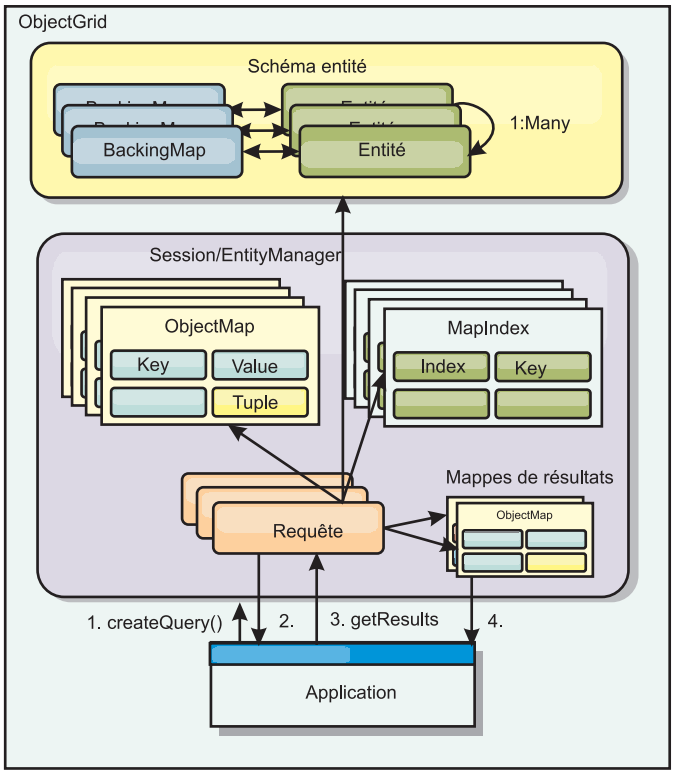


Figure 2. Interaction de la requête avec les mappes de l'objet ObjectGrid, définition du schéma d'entité et association de celui-ci à une mappe ObjectGrid.

Extraction des résultats d'une requête à l'aide de la méthode getResultIterator

Les résultats d'une requête consistent en une liste d'attributs. Imaginons la requête suivante : select a,b,c from X where y=z. Une liste de lignes contenant a, b et c est renvoyée. Cette liste est stockée dans une mappe de portée transaction, ce qui signifie que vous devez associer une clé artificielle à chaque ligne et utiliser un entier qui augmente à chaque ligne. Cette mappe est obtenue à l'aide de la méthode Query.getResultMap. Elle est associée à EntityMetaData, qui décrit chaque ligne de la mappe associée à celle-ci. Vous pouvez accéder aux éléments de chaque ligne à l'aide de lignes de code semblables à l'exemple qui suit :

```
Query q = em.createQuery("select c.id, c.firstName, c.surname from Customer c where c.surname=?1");

  q.setParameter(1, "Claus");

  Iterator iter = q.getResultIterator();
  while(iter.hasNext())
  {
    Object[] row = (Object[])iter.next();
    System.out.println("Found a Claus with id "  + row[objectgrid: 0 ]
      + ", firstName: " + row[objectgrid: 1 ]
      + ", surname: " + row[objectgrid: 2 ]);
  }
```

Extraction des résultats d'une requête à l'aide de getResultMap

Le code suivant présente comment extraire certaines parties des clients correspondant à la requête et montre comment accéder aux lignes de résultats. Si vous utilisez l'objet Query pour accéder aux données, l'identificateur généré pour la ligne de la valeur de type long est masqué. Il est visible uniquement lorsqu'ObjectMap est utilisé pour accéder au résultat. Une fois la transaction terminée, la mappe disparaît. Celle-ci est visible uniquement par la session utilisée,

c'est-à-dire, normalement, par l'unité d'exécution qui l'a créée. Pour la clé, elle utilise un bloc de données avec un seul attribut ou une valeur de type long avec l'ID de la ligne. La valeur est un autre bloc de données associé à un attribut pour chaque colonne de l'ensemble de résultats.

Cet exemple de code est présenté ci-dessous :

```
Query q = em.createQuery("select c.id, c.firstName, c.surname from
Customer c where c.surname=?1");
q.setParameter(1, "Claus");
ObjectMap qmap = q.getResultMap();
Tuple keyTuple = qmap.getEntityMetadata().getKeyMetadata().createTuple();
for(long i = 0; true; ++i)
{
  keyTuple.setAttribute(0, new Long(i));
  Tuple row = (Tuple)qmap.get(keyTuple);
  if(row == null) break;
  System.out.println(" I Found a Claus with id "  + row.getAttribute(0)
    + ", firstName: " + row.getAttribute(1)
    + ", surname: " + row.getAttribute(2));
}
```

Extraction des résultats d'une requête à l'aide d'un itérateur de résultat d'entité

L'exemple de code suivant présente la requête et la boucle permettant d'extraire chaque ligne de résultat à l'aide des API Map normales. La clé correspondant à la mappe est un bloc de données. Vous devez donc construire l'un des types corrects à l'aide de la méthode createTuple dans keyTuple. Essayez d'extraire toutes les lignes dont l'ID est égal ou supérieur à 0. Lorsque des valeurs null sont renvoyées (indiquant qu'aucune clé n'a été trouvée), la boucle se termine. Définissez le premier attribut de keyTuple en tant que valeur de type long que vous souhaitez rechercher. La valeur renvoyée par get est aussi un bloc de données associé à un attribut pour chaque colonne du résultat de la requête. Extrayez ensuite chaque attribut de la valeur Tuple à l'aide de getAttribute.

Voici un exemple du fragment de code suivant :

```
Query q2 = em.createQuery("select c.id, c.firstName, c.surname from Customer c where c.surname=?1");
q2.setResultEntityName("CustomerQueryResult");
q2.setParameter(1, "Claus");

Iterator iter2 = q2.getResultIterator(CustomerQueryResult.class);
while(iter2.hasNext())
{
  CustomerQueryResult row = (CustomerQueryResult)iter2.next();
  // firstName is the id not the firstName.
  System.out.println("Found a Claus with id " + row.id
    + ", firstName: " + row.firstName
    + ", surname: " + row.surname);
}

em.getTransaction().commit();
```

Une valeur ResultEntityName est indiquée pour la requête. Cette valeur indique au moteur de requête que vous souhaitez projeter chaque ligne vers un objet donné, dans ce cas CustomerQueryResult. La classe suit :

```
@Entity
public class CustomerQueryResult {
 @Id long rowId;
 String id;
 String firstName;
 String surname;
};
```

Dans le premier fragment, remarquez que chaque ligne de la requête est renvoyée en tant qu'objet CustomerQueryResult plutôt qu'en tant qu'Object[]. Les colonnes de résultat de la requête sont projetées vers l'objet CustomerQueryResult. La projection du résultat est légèrement plus lente lors de l'exécution, mais plus lisible. Les entités du résultat de la requête ne doivent pas être enregistrés avec eXtreme Scale au démarrage. Si elles le sont, une mappe globale de même nom est créée et la requête échoue avec une erreur indiquant que le nom de mappe est en double.

Requêtes simples avec EntityManager

WebSphere eXtreme Scale est fourni avec l'API de requête EntityManager.

L'API de requête EntityManager est très semblable aux autres moteurs de requête SQL qui exécutent des requêtes sur des objets. Une requête est définie, puis le résultat est extrait de la requête à l'aide de diverses méthodes getResult.

Les exemples suivants font référence aux entités utilisées dans le tutoriel EntityManager de la Présentation du produit.

Exécution d'une requête simple

Dans cet exemple, vous recherchez les clients dont le nom de famille est Claus :

```
em.getTransaction().begin();

  Query q = em.createQuery("select c from Customer c where c.surname='Claus'");

  Iterator iter = q.getResultIterator();
  while(iter.hasNext())
  {
    Customer c = (Customer)iter.next();
    System.out.println("Found a claus with id " + c.id);
  }

  em.getTransaction().commit();
```

Utilisation des paramètres

Comme vous recherchez tous les clients dont le nom de famille est Claus, vous utilisez un paramètre destiné à définir le nom de famille, car vous souhaiterez peut-être réutiliser cette requête.

Exemple de paramètre positionnel

```
Query q = em.createQuery("select c from Customer c where c.surname=?1");
  q.setParameter(1, "Claus");
```

L'utilisation de paramètres est très importante lorsqu'une requête est utilisée plusieurs fois. EntityManager doit analyser la chaîne de requête et générer un plan pour la requête, ce qui consomme une grande quantité de ressources. L'utilisation d'un paramètre permet à EntityManager de mettre en cache le plan de la requête et de réduire le temps nécessaire à son exécution.

Des paramètres positionnels et des paramètres de nom sont utilisés :

Exemple de paramètre de nom

```
Query q = em.createQuery("select c from Customer c where c.surname=:name");
  q.setParameter("name", "Claus");
```

Utilisation d'un index en vue d'améliorer les performances

Si vous avez des millions de clients, la requête précédente doit analyser toutes les lignes de la mappe Customer. Cette opération risque de ne pas être efficace. C'est pourquoi eXtreme Scale propose un mécanisme permettant de définir des index pour les attributs contenus dans une entité. Lorsque c'est nécessaire, la requête utilise automatiquement cet index, ce qui permet d'accélérer considérablement son traitement.

Pour définir les attributs à indexer, il vous suffit d'utiliser l'annotation @Index sur l'attribut d'entité :

```
@Entity
public class Customer
{
  @Id String id;
  String firstName;
  @Index String surname;
  String address;
  String phoneNumber;
}
```

EntityManager crée un index ObjectGrid approprié pour l'attribut du nom de famille dans l'entité Customer et le moteur de requête utilise automatiquement l'index, ce qui réduit fortement la durée de la requête.

Utilisation de la pagination en vue d'améliorer les performances

Si un million de clients se nomment Claus, il est peu probable que vous souhaitiez afficher une page les contenant tous. Vous souhaiterez sans doute afficher 10 ou 25 clients à la fois.

Les méthodes Query setFirstResult et setMaxResults sont utiles car elle renvoient uniquement un sous-ensemble de résultats.

Exemple de pagination

```
Query q = em.createQuery("select c from Customer c where c.surname=:name");
  q.setParameter("name", "Claus");
  // Display the first page
  q.setFirstResult=1;
  q.setMaxResults=25;
  displayPage(q.getResultIterator());

  // Display the second page
  q.setFirstResult=26;
  displayPage(q.getResultIterator());
```

Référence pour les requêtes eXtreme Scale

WebSphere eXtreme Scale possède son propre langage à l'aide duquel l'utilisateur peut interroger les données.

Clause FROM de requête ObjectGrid

La *clause FROM indique les collections d'objets auxquels la requête doit être appliquée. Chaque collection est identifiée par un nom de schéma abstrait et une variable d'identification, appelée variable de plage, ou par une déclaration de membre de collection qui identifie une relation à valeur unique ou à plusieurs valeurs et une variable d'identification.

D'une manière conceptuelle, la sémantique de la requête consiste à former d'abord une collection temporaire de nuplets, appelée R. Les nuplets sont composés d'éléments des collections identifiées dans la clause FROM. Chaque nuplet contient un élément de chacune des collections de la clause FROM. Toutes les combinaisons possibles sont constituées en fonction des contraintes imposées par les déclarations de membre de collection. Si un nom de schéma identifie une collection pour laquelle il n'existe pas d'enregistrements dans le stockage de persistance, la collection temporaire R est vide.

Exemples d'utilisation de la clause FROM

L'objet DeptBean contient les enregistrements 10, 20 et 30. L'objet EmpBean contient les enregistrements 1, 2 et 3 relatifs à la division 10 et les enregistrements 4 et 5 relatifs à la division 20. La division 30 n'est associée à aucun employé.

```
FROM DeptBean d, EmpBean e
```

Cette clause constitue une collection temporaire R qui contient 15 nuplets.

```
FROM  DeptBean d,  DeptBean d1
```

Cette clause constitue une collection temporaire R qui contient 9 nuplets.

```
FROM  DeptBean d,  IN (d.emps) AS e
```

Cette clause constitue une collection temporaire R qui contient 5 nuplets. La division 30 ne se trouve pas dans la collecte temporaire R car elle ne contient aucun employé. La division 10 se trouve trois fois dans la collection temporaire R et la division 20 s'y trouve deux fois.

Au lieu d'utiliser IN(d.emps) as e, vous pouvez utiliser un prédicat JOIN :

```
FROM DeptBean d JOIN d.emps as e
```

Une fois la collection temporaire formée, les conditions de recherche de la clause WHERE sont appliquées à la collection temporaire R et renvoient une nouvelle collection temporaire R1. Les clauses ORDER BY et SELECT sont appliquées à R1 pour renvoyer l'ensemble de résultats final.

Une variable d'identification est une variable déclarée dans la clause FROM à l'aide de l'opérateur IN ou de l'opérateur facultatif AS.

```
FROM DeptBean AS d,  IN (d.emps) AS e
```

équivaut à :

```
FROM DeptBean d, IN (d.emps) e
```

Une variable d'identification déclarée comme nom de schéma abstrait est appelée variable de plage. Dans la requête précédente, "d" est une variable de plage. Une variable d'identification déclarée comme expression de chemin à plusieurs valeurs est appelée déclaration de membre de collection. Les variables "d" et "e" de l'exemple précédent sont des déclarations de membre de collection.

Voici un exemple d'utilisation d'une expression de chemin à valeur unique dans la clause FROM :

```
FROM EmpBean e, IN(e.dept.mgr) as m
```

Clause SELECT de requête ObjectGrid

La syntaxe de la clause SELECT est illustrée dans l'exemple suivant :

```
SELECT { ALL | DISTINCT }  [ selection , ]*  selection
selection  ::= {single_valued_path_expression |
                variable_identification |  OBJECT ( variable_identification) |
        aggregate_functions } [[ AS ] id ]
```

La clause SELECT se compose d'un ou plusieurs des éléments suivants : une variable d'identification définie dans la clause FROM, une expression de chemin d'accès à valeur unique ayant pour résultat des références d'objet ou des valeurs et une fonction d'agrégation. Vous pouvez utiliser le mot clé DISTINCT pour éliminer les références en double.

Une sous-requête scalaire est une sous-requête qui ne renvoie qu'une valeur.

Exemples d'utilisation de la clause SELECT

Pour trouver tous les employés qui touchent plus que l'employé Jean :

```
SELECT OBJECT(e) FROM EmpBean ej, EmpBean eWHERE  ej.name = 'John'  and
e.salary > ej.salary
```

Trouver toutes les divisions ayant un ou plusieurs employés dont le salaire est inférieur à 20000 :

```
SELECT DISTINCT e.dept  FROM EmpBean e where e.salary < 20000
```

Une requête peut avoir une expression de chemin d'accès qui a pour résultat une valeur arbitraire :

```
SELECT e.dept.name  FROM EmpBean e where e.salary < 20000
```

La requête précédente renvoie une collection de noms des divisions ayant des employés dont le salaire est inférieur à 20000.

Une requête peut renvoyer une valeur agrégée :

```
SELECT  avg(e.salary)  FROM EmpBean e
```

Voici une requête qui extrait les noms et les références d'objet pour les employés sous-payés :

```
SELECT e.name as name, object(e) as emp from EmpBean e where e.salary <
50000
```

Clause WHERE de requête ObjectGrid

La clause WHERE contient des conditions de recherche composées des éléments présentés ci-après. Si une condition de recherche a pour résultat TRUE, le bloc de données est ajouté à l'ensemble de résultats.

Littéraux de requête ObjectGrid

Un littéral chaîne est indiqué entre apostrophe. Une apostrophe apparaissant dans un littéral chaîne doit être doublée. Par exemple : 'Tom''s'.

Un littéral numérique peut être de l'une des valeurs suivantes :

- Une valeur exacte, telle que 57, -957 ou +66
- Toute valeur prise en charge par le type Java long
- Un littéral décimal, tel que 57,5 ou -47,02
- Une valeur numérique approchée, telle que 7E3 ou -57,4E-2
- Les types float doivent inclure le qualificateur "F". Par exemple : 1.0F
- Les types long doivent inclure le qualificateur "L". Par exemple : 123L

Les littéraux booléens sont TRUE et FALSE.

Les littéraux temporaux respectent la syntaxe d'échappement de JDBC en fonction du type d'attribut :

- java.util.Date: aaaa-mm-ss
- java.sql.Date: aaaa-mm-ss
- java.sql.Time: hh-mm-ss
- java.sql.Timestamp: aaaa-mm-jj hh:mm:ss.f...
- java.util.Calendar: aaaa-mm-jj hh:mm:ss.f...

Les littéraux d'énumération sont exprimés à l'aide de la syntaxe des littéraux d'énumération Java et du nom complet de la classe enum.

Paramètres d'entrée de requête ObjectGrid

Vous pouvez spécifier des paramètres d'entrée à l'aide d'une position ordinale ou d'un nom de variable. La génération de requêtes qui utilisent des paramètres d'entrée est fortement recommandée, car l'utilisation de paramètres d'entrée augmente les performances en permettant à l'ObjectGrid d'intercepter le plan de requête entre les actions d'exécution.

Un paramètre d'entrée peut être de l'un des types suivants : Byte, Short, Integer, Long, Float, Double, BigDecimal, BigInteger, String, Boolean, Char, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.util.Calendar, une énumération Java SE 5, une entité ou un objet Java simple ou une chaîne de données binaires au format Java byte[].

Un paramètre d'entrée ne doit pas avoir une valeur NULL. Pour rechercher une valeur NULL, utilisez le prédicat NULL.

Paramètres positionnels

Les paramètres d'entrée positionnels sont définis à l'aide d'un point d'interrogation suivi d'un nombre positif :

```
?[positive integer].
```

Les paramètres d'entrée positionnels sont numérotés à partir de 1 et correspondent aux arguments de la requête ; en conséquence, une requête ne doit pas contenir de paramètre d'entrée dont le numéro est supérieur au nombre d'arguments d'entrée.

Exemple : `SELECT e FROM Employee e WHERE e.city = ?1 and e.salary >= ?2`

Paramètres de nom

Les paramètres d'entrée de nom sont définis à l'aide d'un nom de variable au format : :[nom du paramètre].

Exemple : `SELECT e FROM Employee e WHERE e.city = :city and e.salary >= :salary`

Prédicat BETWEEN de requête ObjectGrid

Le prédicat BETWEEN détermine si une valeur donnée est comprise entre deux autres valeurs données.

```
expression [NOT]  BETWEEN expression-2  AND expression-3
```

Exemple 1

```
e.salary BETWEEN 50000 AND 60000
```

équivaut à :

```
e.salary >= 50000  AND e.salary <= 60000
```

Exemple 2

```
e.name NOT BETWEEN 'A' AND 'B'
```

équivaut à :

```
e.name < 'A'  OR e.name > 'B'
```

Prédicat IN de requête ObjectGrid

Le prédicat IN compare une valeur à une série de valeurs. Vous pouvez utiliser le prédicat IN sous deux formes :

```
expression [NOT] IN ( subselect )expression [NOT] IN  ( value1, value2,
.... )
```

La valeur ValueN peut être une valeur littérale ou un paramètre d'entrée. L'expression ne peut pas avoir une référence pour résultat.

Exemple 1

```
e.salary IN ( 10000, 15000 )
```

équivaut à :

```
( e.salary = 10000  OR  e.salary = 15000 )
```

Exemple 2

```
e.salary IN ( select  e1.salary from EmpBean e1 where e1.dept.deptno = 10)
```

équivaut à :

```
e.salary = ANY   ( select  e1.salary from EmpBean e1 where e1.dept.deptno =
10)
```

Exemple 3

```
e.salary NOT IN ( select  e1.salary from EmpBean e1 where e1.dept.deptno =
10)
```

équivaut à :

```
e.salary <> ALL    ( select  e1.salary from EmpBean e1 where e1.dept.deptno
= 10)
```

Prédicat LIKE de requête ObjectGrid

Le prédicat LIKE recherche une valeur de chaîne pour un modèle particulier.

```
expression-chaîne   [NOT]  LIKE pattern [ ESCAPE caractère-échappement ]
```

La valeur de modèle est un littéral chaîne ou un marqueur de paramètre de chaîne de type dans lequel le soulignement (_) remplace tout caractère et le signe pourcentage (%) toute séquence de caractères, y compris une séquence vide). Tout autre caractère correspond à lui même. Le caractère d'échappement permet de rechercher les caractères _ et %. Il peut être indiqué en tant que littéral chaîne ou paramètre d'entrée.

Si l'expression de chaîne est nulle, le résultat est inconnu.

Si l'expression de chaîne et le modèle nuls, le résultat est vrai.

Exemple

```
'' LIKE '' is true
'' LIKE '%' is true
e.name LIKE '12%3' is true for '123' '12993' and false for '1234'
e.name LIKE 's_me' is true for 'some' and 'same', false for 'soome'
e.name LIKE '/_foo' escape '/' is true for '_foo', false for 'afoo'
e.name LIKE '//_foo' escape '/' is true for '/afoo' and for '/bfoo'
e.name LIKE '///_foo' escape '/' is true for '/_foo' but false for '/afoo'
```

Prédicat NULL de requête ObjectGrid

Le prédicat NULL recherche les valeurs nulles (NULL).

```
{single-valued-path-expression | input_parameter} IS [NOT] NULL
```

Exemple

```
e.name IS NULL
e.dept.name IS NOT NULL
e.dept IS NOT NULL
```

Prédicat de collection EMPTY de requête ObjectGrid

Utilisez le prédicat de collection EMPTY pour vérifier si une collection est vide.

Pour vérifier si une relation à plusieurs valeurs est vide, utilisez la syntaxe suivante :

```
expression-chemin-valorisé-collection  IS [NOT]  EMPTY
```

Exemple

Prédicat de collection empty Permet de rechercher toutes les divisions ne possédant pas d'employés :

```
SELECT OBJECT(d) FROM DeptBean d  WHERE  d.emps IS EMPTY
```

Prédicat MEMBER OF de requête ObjectGrid

L'expression ci-après vérifie si la référence d'objet indiquée par l'expression de chemin d'accès à valeur unique ou le paramètre d'entrée fait partie de la collection désignée. Si l'expression de chemin d'accès valorisée de la collection désigne une collection vide, la valeur de l'expression MEMBER OF est FALSE.

```
{ expression-chemin-valeur-unique | paramètre_entrée } [ NOT ] MEMBER [ OF
] expression-chemin-valorisée-collection
```

Exemple

Trouver les employés qui n'appartiennent pas à une division donnée :

```
SELECT OBJECT(e) FROM
EmpBean e , DeptBean d
WHERE e NOT MEMBER OF d.emps AND d.deptno = ?1
```

Trouver les employés dont le responsable appartient à une division donnée :

```
SELECT OBJECT(e) FROM EmpBean e,
DeptBean d
WHERE e.dept.mgr MEMBER  OF d.emps  and d.deptno=?1
```

Prédicat EXISTS de requête ObjectGrid

Le prédicat EXISTS vérifie la présence ou l'absence d'une condition spécifiée par une sous-requête.

```
EXISTS ( sous-requête )
```

Le résultat d'EXISTS est true si la sous-requête renvoie au moins une valeur ; sinon le résultat est false.

Pour inverser un prédicat EXISTS, faites-le précéder de l'opérateur logique NOT.

Exemple

Pour renvoyer les divisions dont l'un des employés au moins a un salaire supérieur à 1000000 :

```
SELECT OBJECT(d) FROM DeptBean d
WHERE EXISTS ( SELECT  e  FROM IN (d.emps) e WHERE  e.salary > 1000000 )
```

Pour renvoyer les divisions sans employés :

```
SELECT OBJECT(d) FROM DeptBean d
WHERE NOT EXISTS  ( SELECT e FROM IN (d.emps) e)
```

Vous pouvez également modifier la requête précédente conformément à l'exemple suivant :

```
SELECT OBJECT(d) FROM DeptBean d WHERE SIZE(d.emps)=0
```

Clause ORDER BY de requête ObjectGrid

La clause ORDER BY spécifie l'ordre de classement des objets dans la collection résultante. Voici un exemple :

ORDER BY [order_element ,]* order_element order_element ::={ path-expression }[ASC | DESC]

L'expression de chemin d'accès doit indiquer un champ à valeur unique de type primitif byte, short, int, long, float, double, char ou de type encapsuleur Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Character, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp ou java.util.Calendar. L'élément de classement ASC indique que les résultats sont affichés dans l'ordre croissant, qui correspond à la valeur par défaut. Un élément de classement DESC indique que les résultats sont affichés dans l'ordre décroissant.

Exemple

Renvoyez les objets division. Affichez les numéros des divisions par ordre décroissant :

```
SELECT  OBJECT(d) FROM DeptBean d ORDER BY  d.deptno DESC
```

Pour renvoyer les objets employés, triés par numéro de division et par nom :

```
SELECT  OBJECT(e) FROM EmpBean e ORDER BY e.dept.deptno ASC,  e.name DESC
```

Fonctions d'agrégation de requête ObjectGrid

Les fonctions d'agrégation opèrent sur une série de valeurs pour renvoyer une valeur scalaire. Vous pouvez utiliser ces fonctions dans les méthodes de sélection et de sous-requête. Voici un exemple d'agrégation :

```
SELECT  SUM (e.salary) FROM EmpBean e WHERE e.dept.deptno =20
```

Cette agrégation calcule le total des salaires de la division 20.

Les fonctions d'agrégation sont AVG, COUNT, MAX, MIN et SUM. La syntaxe d'une fonction d'agrégation est illustrée dans l'exemple suivant :

```
fonction-agrégation  (    [ ALL |  DISTINCT ]  expression )
```

ou :

```
COUNT( [ ALL | DISTINCT ] identification-variable )
```

L'option DISTINCT supprime les valeurs en double avant d'exécuter la fonction. L'option ALL est l'option par défaut qui ne supprime pas les valeurs en double. Les valeurs NULL sont ignorées lors du traitement de la fonction d'agrégation sauf si vous utilisez la fonction COUNT(identification-variable), qui renvoie le total de tous les éléments que contient la série.

Définition du type de retour

Les fonctions MAX et MIN peuvent s'appliquer à tout type de données numérique, chaîne ou de date et d'heure et renvoient le type de données correspondant. Les fonctions SUM et AVG acceptent un type numérique en entrée. La fonction AVG renvoie un type double. La fonction SUM renvoie un type long si l'entrée correspond à un entier, sauf s'il s'agit d'un type Java BigInteger. Dans ce cas, la fonction renvoie un type Java BigInteger. La fonction SUM renvoie un type double si l'entrée ne correspond pas à un entier, sauf s'il s'agit d'un type Java BigDecimal. Dans ce cas, la fonction renvoie un type Java BigDecimal. La fonction COUNT peut accepter n'importe quel type de données, à part les collections, et renvoie un type long.

Lorsqu'elles s'appliquent à un ensemble vide, les fonctions SUM, AVG, MAX et MIN peuvent renvoyer une valeur null. La fonction COUNT renvoie zéro (0) lorsqu'elle est appliquée à un ensemble vide.

Utilisation des clauses GROUP BY et HAVING

La série de valeurs utilisée pour la fonction d'agrégation est déterminée par la collection résultant de la clause FROM et WHERE de la requête. Vous pouvez diviser la série en groupes et appliquer la fonction d'agrégation à chaque groupe. Pour exécuter cette action, utilisez une clause GROUP BY dans la requête. Cette clause définit les membres des groupes, ce qui comprend une liste d'expressions de chemin d'accès. Chaque expression de chemin d'accès désigne un champ de type primitif byte, short, int, long, float, double, boolean ou char, ou de type encapsuleur Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Boolean, Character, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.util.Calendar ou enum Java SE 5.

L'exemple suivant illustre l'utilisation de la clause GROUP BY dans une requête qui calcule le salaire moyen pour chaque division :

```
SELECT e.dept.deptno,  AVG ( e.salary) FROM EmpBean e GROUP BY
e.dept.deptno
```

Lorsque vous divisez une série en groupes, une valeur NULL est considérée comme étant égale à une autre valeur NULL.

Les groupes peuvent être filtrés à l'aide d'une clause HAVING qui teste les propriétés des groupes avant de faire appel à des fonctions d'agrégation ou de regrouper les membres. Ce filtrage est similaire au filtrage des nuplets (c'est-à-dire, des enregistrements des valeurs des collections renvoyées) de la clause FROM par la clause WHERE. Voici un exemple de clause HAVING :

```
SELECT e.dept.deptno,  AVG ( e.salary) FROM EmpBean e
GROUP BY e.dept.deptno
HAVING COUNT(e) > 3 AND e.dept.deptno > 5
```

Cette requête renvoie le salaire moyen des divisions ayant plus de trois employés et dont le numéro de division est supérieur à cinq.

Vous pouvez utiliser une clause HAVING sans clause GROUP BY. Dans ce cas, la totalité de la série est traitée comme un seul et même groupe auquel est appliquée la clause HAVING.

Formulaire BNF (Backus-Naur Form) de requête ObjectGrid

Vous trouverez ci-après un récapitulatif de la notation du formulaire BNF (Backus-Naur Form) de requête ObjectGrid.

Tableau 3. Description du récapitulatif BNF

Représentation	Description
{...}	Regroupement
[...]	Constructions facultatives
bold	Mots clés
*	Zéro ou plus
|	Autres

```
ObjectGrid QL ::=select_clause from_clause [where_clause] [group_by_clause]
  [having_clause] [order_by_clause]

from_clause ::=FROM identification_variable_declaration
  [,identification_variable_declaration]*

déclaration_variable_identification ::=déclaration_membre_collection |
  range_variable_declaration

collection_member_declaration ::=IN ( collection_valued_path_expression |
  single_valued_navigation) [AS] identifier | [LEFT [OUTER]
  | INNER] JOIN collection_valued_path_expression |
  single_valued_navigation [AS] identifier

range_variable_declaration ::=abstract_schema_name [AS] identifier

single_valued_path_expression ::={single_valued_navigation | identification_variable}.
  { state_field | state_field.value_object_attribute } | single_valued_navigation

single_valued_navigation ::=identification_variable.[ single_valued_association_field. ]*
  single_valued_association_field

collection_valued_path_expression ::=identification_variable.[
  single_valued_association_field. ]* collection_valued_association_field

select_clause ::= SELECT [DISTINCT] [ selection , ]* selection

selection ::= {single_valued_path_expression |identification_variable | OBJECT
  ( identification_variable) |aggregate_functions } [[ AS ] id ]

order_by_clause ::= ORDER BY [ {identification_variable.[ single_valued_association_field.
  ]*state_field} [ASC|DESC],]* {identification_variable.[
  single_valued_association_field. ]*state_field}[ASC|DESC]

where_clause ::= WHERE conditional_expression

conditional_expression ::= conditional_term | conditional_expression OR conditional_term

conditional_term ::= conditional_factor | conditional_term AND conditional_factor

conditional_factor ::= [NOT] conditional_primary

primaire_conditionnel ::=expression_cond_simple | (expression_conditionnelle)

simple_cond_expression ::= comparison_expression | between_expression | like_expression |
  in_expression | null_comparison_expression | empty_collection_comparison_expression |
  exists_expression | collection_member_expression

between_expression ::= numeric_expression [NOT] BETWEEN numeric_expression
  AND numeric_expression | string_expression [NOT] BETWEEN
  string_expression AND string_expression | datetime_expression [NOT]
  BETWEEN datetime_expression AND datetime_expression

in_expression ::= identification_variable.[ single_valued_association_field. ]state_field
  [*NOT] IN { (subselect) | ( atom ,]* atom) }

atom ::= { string_literal | numeric_literal | input_parameter }

like_expression ::=string_expression [NOT] LIKE {string_literal | input_parameter}
  [ESCAPE {string_literal | input_parameter}]

null_comparison_expression ::= {single_valued_path_expression | input_parameter} IS
  [ NOT ] NULL

empty_collection_comparison_expression ::= collection_valued_path_expression IS
  [NOT] EMPTY

collection_member_expression ::={ ssingle_valued_path_expression | input_parameter }[
  NOT ] MEMBER [ OF ]collection_valued_path_expression

exists_expression ::= EXISTS {(subselect)}

subselect ::= SELECT [{ ALL | DISTINCT }] subselection from_clause
  [where_clause] [group_by_clause] [having_clause]

subselection ::= {single_valued_path_expression |identification_variable |
  aggregate_functions }
```

```
group_by_clause ::= GROUP BY[single_valued_path_expression,]*
  single_valued_path_expression

having_clause ::= HAVING conditional_expression

comparison_expression ::= numeric_ expression comparison_operator { numeric_expression
  | {SOME | ANY | ALL} (subselect) } | string_expression
  comparison_operator {

string_expression | {SOME | ANY | ALL}(subselect) } |

datetime_expression comparison_operator {

datetime_expression {SOME | ANY | ALL}(subselect) } |

boolean_expression {=|<>} {

boolean_expression {SOME | ANY | ALL}(subselect) } |

entity_expression {=|<>} {

entity_expression {SOME| ANY | ALL}(subselect) }

comparison_operator ::= = | > | >= | < | <= | <>

string_expression ::= string_primary | (subselect)

string_primary ::=state_field_path_expression |string_literal | input_parameter |
  functions_returning_strings

datetime_expression ::= datetime_primary |(subselect)

datetime_primary ::=state_field_path_expression | string_literal | long_literal
  | input_parameter | functions_returning_datetime

boolean_expression ::= boolean_primary |(subselect)

boolean_primary ::=state_field_path_expression | boolean_literal | input_parameter

entity_expression ::=single_valued_association_path_expression |
  identification_variable | input_parameter

numeric_expression ::= simple_numeric_expression |(subselect)

simple_numeric_expression ::= numeric_term | numeric_expression {+|-} numeric_term

numeric_term ::= numeric_factor | numeric_term {*|/} numeric_factor

numeric_factor ::= {+|-} numeric_primary

numeric_primary ::= single_valued_path_expression | numeric_literal |
  ( numeric_expression ) | input_parameter | functions

fonctions_agrégées :=

AVG([ALL|DISTINCT] identification_variable.
  [ single_valued_association_field. ]*state_field) |

COUNT([ALL|DISTINCT] {single_valued_path_expression |
  identification_variable}) |

MAX([ALL|DISTINCT] identification_variable.[
  single_valued_association_field. ]*state_field) |

MIN([ALL|DISTINCT] identification_variable.[
  single_valued_association_field. ]*state_field) |

SUM([ALL|DISTINCT] identification_variable.[
  single_valued_association_field. ]*state_field)

fonctions ::=

ABS (simple_numeric_expression) |

CONCAT (string_primary , string_primary) |

LOWER (string_primary) |

LENGTH(string_primary) |

LOCATE(string_primary, string_primary [, simple_numeric_expression]) |

MOD (simple_numeric_expression, simple_numeric_expression) |

SIZE (collection_valued_path_expression) |

SQRT (simple_numeric_expression) |

SUBSTRING (string_primary, simple_numeric_expression[, simple_numeric_expression]) |

UPPER (string_primary) |

TRIM ([[LEADING | TRAILING | BOTH] [trim_character]
  FROM] string_primary)
```

Optimisation des performances de requêtes

Les conseils et techniques qui suivent vous aideront à optimiser les performances de vos requêtes.

Utiliser des paramètres

Lorsqu'une requête s'exécute, la syntaxe de la chaîne de la requête doit être analysée et un plan doit être développé pour l'exécution de la requête, ce qui, dans les deux cas, peut s'avérer assez onéreux. WebSphere eXtreme Scale met en cache les plans de requête d'après la chaîne de requête. Le cache ayant une taille non illimitée, il est important de réutiliser autant que possible les chaînes de requêtes. L'utilisation de paramètres nommés ou positionnels est également un facteur de performances car il favorise la réutilisation des plans de requête.

```
Positional Parameter Example  Query q = em.createQuery("select c from
Customer c where c.surname=?1");   q.setParameter(1, "Claus");
```

Utiliser des index

L'indexation correctement conçue d'une mappe peut avoir un impact significatif sur les performances des requêtes, même si l'indexation a par elle-même sa part de charge système dans les performances globales de la mappe. Sans indexation des attributs d'objets impliqués dans les requêtes, pour chacun des attributs, le moteur de requête est en effet obligé d'analyser les tables. Une analyse de tables est l'opération la plus onéreuse au cours de l'exécution d'une requête. L'indexation des attributs d'objets qui sont impliqués dans la requête permet au moteur de requête d'éviter les analyses superflues de tables et ne peut qu'améliorer les performances générales de la requête. Si l'application est conçue pour faire un usage intensif consistant essentiellement à lire une mappe, configurez des index pour les attributs d'objets qui sont impliqués dans la requête. Si au contraire la mappe est le plus souvent actualisée, vous devez trouver un équilibre entre l'amélioration des performances des requêtes et la charge imposée par l'indexation à la mappe. Pour plus d'informations, voir Indexation.

Lorsque des objets POJO sont stockés dans une mappe, une indexation correctement conçue peut éviter une réflexion Java. Dans l'exemple qui suit, la requête remplace la clause WHERE par une recherche d'index de plage si un index est construit sur le champ budget. Sinon, la requête analyse la mappe toute entière et évalue la clause WHERE en obtenant d'abord le budget à l'aide d'une réflexion Java, puis en comparant le budget avec la valeur 50000 :

```
SELECT d FROM DeptBean d WHERE d.budget=50000
```

Voir «Plan de requête», à la page 122 pour savoir comment optimiser au maximum des requêtes individuelles et en quoi différentes syntaxes, différents modèles d'objets et différents index peuvent affecter les performances des requêtes.

Utiliser la pagination

Dans des environnements client-serveur, le moteur de requête transporte vers le client la totalité de la mappe résultante. Les données retournées doivent alors être fractionnées en morceaux raisonnables. Les interfaces Query d'EntityManager et ObjectQuery d'ObjectMap prennent toutes les deux en charge les méthodes setFirstResult et setMaxResults qui permettent à la requête de retourner un sous-ensemble des résultats.

Retourner des valeurs primitives plutôt que des entités

Avec l'API Query d'EntityManager, les entités sont retournées comme des paramètres de requête. Le moteur de requête retourne couramment au client les clés de ces entités. Lorsque le client opère une itération sur ces entités à l'aide de l'Iterator de la méthode getResultIterator, chacune de ces entités est automatiquement agrandie et gérée comme si elle avait été créée avec la méthode find de l'interface EntityManager. Le graphe entier des entités est construit sur le client à partir de la mappe d'objets entités. La résolution des attributs value des entités et des éventuelles entités en rapport s'effectue avec peine.

Pour éviter d'avoir à construire le graphe, opération toujours onéreuse, il suffit de modifier la requête pour qu'elle retourne les attributs individuels avec navigation via le chemin.

Exemple :

```
// Retourne une entité
SELECT p FROM Person p
// Retourne des attributs SELECT p.name, p.address.street, p.address.city, p.gender FROM Person p
```

Plan de requête

Toutes les requêtes eXtreme Scale ont un plan de requête. Le plan décrit la manière dont le moteur de requête va interagir avec les mappes d'objets et les index. L'affichage du plan de requête permet de déterminer si la chaîne de requête ou les index sont utilisés de manière appropriée. Le plan de requête peut également servir à explorer les différences que de légères modifications de la chaîne à rechercher peuvent apporter à l'exécution de la requête par eXtreme Scale.

Le plan de requête peut être affiché de deux manières :

- les méthodes Query d'EntityManager ou API getPlan d'ObjectQuery
- la trace de diagnostics d'ObjectGrid

Méthode getPlan

La méthode getPlan des interfaces ObjectQuery et Query retournent une chaîne qui décrit le plan de requête. Cette chaîne peut être affichée de manière standard ou dans un journal. Remarque : en environnement réparti, la méthode getPlan ne s'exécute pas sur le serveur et elle ne reflétera aucun index défini. Pour afficher le plan, utilisez un agent pour visualiser le plan sur le serveur.

Trace du plan de requête

Le plan de requête peut être affiché à l'aide de la trace d'ObjectGrid. Pour activer la trace des plans de requête, utilisez les spécifications de trace suivantes :

QueryEnginePlan=debug=enabled

Voir «Journaux et trace», à la page 385 pour savoir comment activer la trace et repérer les fichiers de trace.

Exemples de plans de requête

Le plan de requête utilise le terme for pour indiquer que la requête opère une itération dans une collection ObjectMap, dans une collection dérivée du genre q2.getEmps(), q2.dept, ou dans une collection temporaire retournée par une boucle interne. Si la collection provient d'un ObjectMap, le plan de requête indique si un

index unique ou non unique d'analyse séquentielle (signalé par INDEX SCAN) est utilisé. Le plan de requête utilise une chaîne de filtrage pour afficher la liste des expressions de condition appliquées à une collection.

Un produit cartésien est rarement utilisé dans la requête d'objet. La requête qui suit analyse la totalité de la mappe EmpBean dans la boucle externe et elle analyse la totalité de la mappe DeptBean dans la boucle interne :

```
SELECT e, d FROM EmpBean e, DeptBean d

Plan trace:

for q2 in EmpBean ObjectMap using INDEX SCAN
     for q3 in DeptBean ObjectMap using INDEX SCAN
   returning new Tuple( q2,  q3  )
```

La requête qui suit extrait tous les noms de salariés d'un département particulier en analysant de manière séquentielle la mappe EmpBean à la recherche d'un objet employee. A partir de cet objet employee, la requête navigue vers l'objet department de l'objet employee et elle applique le filtre d.no=1. Dans cet exemple, chaque employé n'a qu'une seule référence à l'objet department, ce qui fait que la boucle interne ne s'exécute qu'une seule fois :

```
SELECT e.name FROM EmpBean e JOIN e.dept d WHERE d.no=1

Plan trace:

for q2 in EmpBean ObjectMap using INDEX SCAN
     for q3 in  q2.dept
      filter ( q3.getNo() = 1 )
   returning new Tuple( q2.name  )
```

La requête qui suit équivaut à la requête précédente. Mais la requête ci-dessous offre de meilleures performances car elle commence par restreindre le résultat à un seul objet department en utilisant l'index unique qui est défini sur le numéro du champ clé primaire DeptBean. A partir de cet objet department, la requête navigue vers les objets employee de l'objet department pour obtenir leurs noms :

```
SELECT e.name FROM DeptBean d JOIN d.emps e WHERE d.no=1

Plan trace:

for q2 in DeptBean ObjectMap using UNIQUE INDEX key=(1)
     for q3 in  q2.getEmps()
   returning new Tuple( q3.name  )
```

La requête suivant recherche tous les salariés qui travaillent pour le développement ou pour les ventes. La requête analyse la totalité de la mappe EmpBean et procède à un filtrage supplémentaire en évaluant les expressions d.name = 'Sales' ou d.name='Dev'

```
SELECT e FROM EmpBean e, in (e.dept) d WHERE d.name = 'Sales'
 or d.name='Dev'

Plan trace:

for q2 in EmpBean ObjectMap using INDEX SCAN
     for q3 in  q2.dept
      filter (( q3.getName() = Sales ) OR ( q3.getName() = Dev ) )
   returning new Tuple( q2  )
```

La requête qui suit équivaut à l'exemple précédent, mais elle exécute un autre plan de requête en utilisant l'index de plages construit sur le champ name. En général, cette requête offre de meilleures performances car l'index sur le champ name est

utilisé pour restreindre les objets department, avec à la clé une exécution rapide si seuls quelques départements sont du développement ou des ventes.

```
SELECT e FROM DeptBean d, in(d.emps) e WHERE d.name='Dev' or d.name='Sales'

Plan trace:

IteratorUnionIndex of

     for q2 in DeptBean ObjectMap using INDEX on name = (Dev)
       for q3 in  q2.getEmps()


     for q2 in DeptBean ObjectMap using INDEX on name = (Sales)
       for q3 in  q2.getEmps()
```

La requête qui suit recherche les départements qui n'ont pas de salariés :

```
SELECT d FROM DeptBean d WHERE NOT EXISTS(select e from d.emps e)

Plan trace:

for q2 in DeptBean ObjectMap using INDEX SCAN
    filter ( NOT  EXISTS (    correlated collection defined as

       for q3 in  q2.getEmps()
       returning new Tuple( q3  )

   returning new Tuple( q2  )
```

La requête qui suit équivaut à la requête précédente, mais elle utilise la fonction scalaire SIZE. Cette requête offre des performances similaires, mais elle est plus facile à écrire.

```
SELECT d FROM DeptBean d WHERE SIZE(d.emps)=0
for q2 in DeptBean ObjectMap using INDEX SCAN
    filter (SIZE( q2.getEmps()) = 0 )
   returning new Tuple( q2  )
```

L'exemple qui suit est une autre manière d'écrire la même requête avec des performances similaires, mais, elle aussi, elle est plus facile à écrire.

```
SELECT d FROM DeptBean d WHERE d.emps is EMPTY

Plan trace:

for q2 in DeptBean ObjectMap using INDEX SCAN
    filter ( q2.getEmps() IS EMPTY  )
   returning new Tuple( q2  )
```

La requête qui suit recherche tous les salariés dont le domicile correspond à l'une au moins des adresses du salarié dont le nom égale la valeur du paramètre. La boucle interne n'a aucune dépendance à l'égard de la boucle externe. La requête n'exécute la boucle interne qu'une seule fois.

```
SELECT e FROM EmpBean e WHERE e.home =  any (SELECT e1.home FROM EmpBean e1
 WHERE e1.name=?1)
for q2 in EmpBean ObjectMap using INDEX SCAN
    filter ( q2.home =ANY     temp collection defined as

       for q3 in EmpBean ObjectMap using INDEX on name = ( ?1)
       returning new Tuple( q3.home      )
 )
   returning new Tuple( q2  )
```

La requête qui suit équivaut à la requête précédente, mais elle comporte une sous-requête corrélée. Par ailleurs, la boucle interne s'exécute de manière répétitive.

```
SELECT e FROM EmpBean e WHERE EXISTS(SELECT e1 FROM EmpBean e1 WHERE
 e.home=e1.home and e1.name=?1)

Plan trace:

for q2 in EmpBean ObjectMap using INDEX SCAN
    filter ( EXISTS (    correlated collection defined as

       for q3 in EmpBean ObjectMap using INDEX on name = (?1)
        filter ( q2.home =  q3.home )
       returning new Tuple( q3  )

   returning new Tuple( q2  )
```

Optimisation des requêtes à l'aide d'index

La définition et l'utilisation adéquates des index peut considérablement améliorer les performances des requêtes.

Les requêtes WebSphere eXtreme Scale peuvent utiliser les plug-in HashIndex pré-intégrés pour améliorer leurs performances. Les index peuvent être définis sur des entités ou sur des attributs d'objets. Le moteur de requête utilisera automatiquement les index définis si sa clause WHERE utilise l'une des chaînes suivantes :

- une expression de comparaison avec les opérateurs suivants : =, <, >, <= or >= (en fait, toutes les expressions de comparaison à l'exception de celle utilisant différent de, <>)
- une expression BETWEEN
- des opérandes d'expressions qui sont des constantes ou des termes simples

Conditions requises

Les index utilisés par Query doivent réunir les conditions suivantes :

- tous les index doivent utiliser le plug-in HashIndex pré-intégré
- tous les index doivent être statiques. Les index dynamiques ne sont pas pris en charge
- l'annotation @Index peut être utilisée pour la création automatique de plug-in HashIndex statiques
- la propriété RangeIndex de tous les index mono-attribut doit avoir la valeur true
- la propriété RangeIndex de tous les index composites doit avoir la valeur false
- la propriété RangeIndex de tous les index d'association (relation) doit avoir la valeur false

Pour savoir comment configurer HashIndex, voir Configuration de HashIndex.

Pour des informations concernant l'indexation, voir Indexation.

Pour une manière plus efficace de rechercher les objets mis en cache, voir «Index HashIndex composite», à la page 246.

Utiliser des suggestions pour choisir un index

La méthode setHint des interfaces Query et ObjectQuery, utilisée avec la constante HINT_USEINDEX, permet de sélectionner manuellement un index. Cela peut être

utile lorsqu'on cherche à optimiser une requête pour qu'elle utilise l'index le plus performant.

Exemples de requêtes utilisant des index d'attributs

L'exemple qui suit utilise des termes simples : e.empid, e.name, e.salary, d.name, d.budget et e.isManager. Ces exemples présupposent que des index sont définis sur les champs name, salary et budget d'une entité ou d'un objet value. Le champ empid est une clé primaire et aucun index n'est défini pour isManager.

La requête suivante utilise les deux index sur les champs name et salary. Elle renvoie tous les salariés dont les noms égalent la valeur du premier paramètre ou un salaire égal à la valeur du second paramètre :

```
SELECT e FROM EmpBean e where e.name=?1 or e.salary=?2
```

La requête suivante utilise les deux index sur les champs name et budget. La requête retourne tous les départements nommés "DEV" dont le budget est supérieur à 2000.

```
SELECT d FROM DeptBean dwhere d.name='DEV' and d.budget>2000
```

La requête suivante retourne tous les salariés dont le salaire est supérieur à 3000 et dont la valeur de l'indicateur isManager égale celle du paramètre. La requête utilise l'index qui est défini sur le champ salary et elle procède à un filtrage supplémentaire en évaluant l'expression de comparaison `e.isManager=?1`.

```
SELECT e FROM EmpBean e where e.salary>3000 and e.isManager=?1
```

La requête suivante recherche tous les salariés qui gagnent plus que le premier paramètre ou tous les salariés qui sont un manager. Bien qu'un index soit défini pour le champ salary, la requête examine l'index qui est automatiquement généré sur les clés primaires du champ EmpBean et elle évalue l'expression `e.salary>?1` ou `e.isManager=TRUE`.

```
SELECT e FROM EmpBean e WHERE e.salary>?1 or e.isManager=TRUE
```

La requête suivante retourne les salariés dont le nom contient la lettre a. Bien qu'un index soit défini pour le champ name, la requête n'utilise pas cet index car le champ name est utilisé dans l'expression LIKE.

```
SELECT e FROM EmpBean e WHERE e.name LIKE '%a%'
```

La requête suivante recherche tous les employés dont le nom n'est pas "Smith". Bien qu'un index soit défini pour le champ name, la requête n'utilise pas cet index car elle utilise l'opérateur de comparaison différent de (<>).

```
SELECT e FROM EmpBean e where e.name<>'Smith'
```

La requête suivante recherche tous les départements dont le budget est inférieur à la valeur du paramètre et qui ont un salarié dont la rémunération est supérieure à 3000. La requête utilise un index pour le salaire mais elle n'utilise pas d'index pour le budget car dept.budget n'est pas un terme simple. Les objets dept sont dérivés de la collection e. L'on n'a pas besoin d'utiliser l'index budget pour rechercher des objets dept.

```
SELECT dept from EmpBean e, in (e.dept) dept where e.salary>3000 and
dept.budget<?
```

La requête suivante recherche tous les salariés dont la rémunération est supérieure à celles des salariés dont l'empid est 1, 2 et 3. L'index salary n'est pas utilisé car la comparaison implique une sous-requête. Le champ empid est néanmoins une clé primaire et il est utilisé pour une recherche d'index unique car un index est automatiquement prédéfini pour les clés primaires.

```
SELECT e FROM EmpBean e WHERE e.salary > ALL (SELECT e1.salary    FROM
EmpBean e1 WHERE e1.empid=1 or e1.empid =2 or e1.empid=99)
```

Pour vérifier si l'index est en cours d'utilisation par la requête, l'on peut visualiser le «Plan de requête», à la page 122. Voici un exemple de plan pour la requête précédente :

```
for q2 in EmpBean ObjectMap using INDEX SCAN
    filter ( q2.salary >ALL     temp collection defined as
       IteratorUnionIndex of

         for q3 in EmpBean ObjectMap using UNIQUE INDEX key=(1)
        )

         for q3 in EmpBean ObjectMap using UNIQUE INDEX key=(2)
        )

         for q3 in EmpBean ObjectMap using UNIQUE INDEX key=(99)
        )
       returning new Tuple( q3.salary )
   returning new Tuple( q2  )

for q2 in EmpBean ObjectMap using RANGE INDEX on salary with range(3000,)
     for q3 in  q2.dept
      filter ( q3.budget <  ?1 )
   returning new Tuple( q3  )
```

Indexer des attributs

Les index peuvent être définis sur n'importe quel type individuel d'attribut, moyennant les contraintes définies plus haut.

Définir des index d'entités à l'aide de l'annotation @Index

Pour définir un index sur une entité, il suffit de définir une annotation :

Entités utilisant des annotations

```
  @Entity
  public class Employee {
  @Id int empid;
  @Index String name
  @Index double salary
  @ManyToOne Department dept;
}
@Entity
  public class Department {
  @Id int deptid;
  @Index String name;
  @Index double budget;
  boolean isManager;
  @OneToMany Collection<Employee> employees;
  }
```

Avec XML

Les index peuvent également être définis à l'aide de XML :

Entités sans annotations

```
public class Employee {
int empid;
String name
double salary
Department dept;
}

public class Department {
int deptid;
String name;
double budget;
boolean isManager;
Collection employees;
}
```

XML ObjectGrid avec index d'attributs

```
<?xml version="1.0" encoding="UTF-8"?>
  <objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
  <objectGrid name="DepartmentGrid" entityMetadataXMLFile="entity.xml">
  <backingMap name="Employee" pluginCollectionRef="Emp"/>
  <backingMap name="Department" pluginCollectionRef="Dept"/>
  </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
  <backingMapPluginCollection id="Emp">
  <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex" >
  <property name="Name" type="java.lang.String" value="Employee.name"/>
  <property name="AttributeName" type="java.lang.String" value="name"/>
  <property name="RangeIndex" type="boolean" value="true"
  description="Ranges are must be set to true for attributes." />
  </bean>
  <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex" >
  <property name="Name" type="java.lang.String" value="Employee.salary"/>
  <property name="AttributeName" type="java.lang.String" value="salary"/>
  <property name="RangeIndex" type="boolean" value="true"
  description="Ranges are must be set to true for attributes." />
  </bean>
  </backingMapPluginCollection>
  <backingMapPluginCollection id="Dept">
  <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex" >
  <property name="Name" type="java.lang.String" value="Department.name"/>
  <property name="AttributeName" type="java.lang.String" value="name"/>
  <property name="RangeIndex" type="boolean" value="true"
  description="Ranges are must be set to true for attributes." />
  </bean>
  <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex" >
  <property name="Name" type="java.lang.String" value="Department.budget"/>
  <property name="AttributeName" type="java.lang.String" value="budget"/>
  <property name="RangeIndex" type="boolean" value="true"
  description="Ranges are must be set to true for attributes." />
  </bean>
  </backingMapPluginCollection>
  </backingMapPluginCollections>
  </objectGridConfig>
```

XML d'entités

```
<?xml version="1.0" encoding="UTF-8"?>
  <entity-mappings xmlns="http://ibm.com/ws/projector/config/emd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/projector/config/emd ./emd.xsd">

  <description>Department entities</description>
  <entity class-name="acme.Employee" name="Employee" access="FIELD">
  <attributes>
  <id name="empid" />
  <basic name="name" />
  <basic name="salary" />
  <many-to-one name="department"
  target-entity="acme.Department"
  fetch="EAGER">
  <cascade><cascade-persist/></cascade>
  </many-to-one>
  </attributes>
  </entity>
```

```
<entity class-name="acme.Department" name="Department" access="FIELD">
<attributes>
<id name="deptid" />
<basic name="name" />
<basic name="budget" />
<basic name="isManager" />
<one-to-many name="employees"
target-entity="acme.Employee"
fetch="LAZY" mapped-by="parentNode">
<cascade><cascade-persist/></cascade>
</one-to-many>
</attributes>
</entity>
</entity-mappings>
```

Définir dans XML des index de non-entités

Les index des types non-entité sont définis dans XML. Il n'existe aucune différence lorsqu'on crée le MapIndexPlugin pour des mappes d'entités ou pour des mappes de non-entités.

Bean Java

```
public class Employee {
  int empid;
  String name
  double salary
  Department dept;

 public class Department {
  int deptid;
  String name;
  double budget;
  boolean isManager;
  Collection employees;
  }
```

XML ObjectGrid avec index d'attributs

```
<?xml version="1.0" encoding="UTF-8"?>
  <objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
  <objectGrid name="DepartmentGrid">
  <backingMap name="Employee" pluginCollectionRef="Emp"/>
  <backingMap name="Department" pluginCollectionRef="Dept"/>
  <querySchema>
  <mapSchemas>
  <mapSchema mapName="Employee" valueClass="acme.Employee"
  primaryKeyField="empid" />
  <mapSchema mapName="Department" valueClass="acme.Department"
  primaryKeyField="deptid" />
  </mapSchemas>
  <relationships>
  <relationship source="acme.Employee"
  target="acme.Department"
  relationField="dept" invRelationField="employees" />
  </relationships>
  </querySchema>
  </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
  <backingMapPluginCollection id="Emp">
  <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex" >
  <property name="Name" type="java.lang.String" value="Employee.name"/>
  <property name="AttributeName" type="java.lang.String" value="name"/>
  <property name="RangeIndex" type="boolean" value="true"
  description="Ranges are must be set to true for attributes." />
  </bean>
  <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex" >
  <property name="Name" type="java.lang.String" value="Employee.salary"/>
  <property name="AttributeName" type="java.lang.String" value="salary"/>
  <property name="RangeIndex" type="boolean" value="true"
  description="Ranges are must be set to true for attributes." />
  </bean>
  </backingMapPluginCollection>
  <backingMapPluginCollection id="Dept">
  <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex" >
  <property name="Name" type="java.lang.String" value="Department.name"/>
  <property name="AttributeName" type="java.lang.String" value="name"/>
  <property name="RangeIndex" type="boolean" value="true"
  description="Ranges are must be set to true for attributes." />
  </bean>
```

```
  <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex" >
  <property name="Name" type="java.lang.String" value="Department.budget"/>
  <property name="AttributeName" type="java.lang.String" value="budget"/>
  <property name="RangeIndex" type="boolean" value="true"
  description="Ranges are must be set to true for attributes." />
  </bean>
  </backingMapPluginCollection>
  </backingMapPluginCollections>
  </objectGridConfig>
```

Indexer des relations

WebSphere eXtreme Scale stocke au sein de l'objet parent les clés externes des entités en rapport. Dans le cas d'entités, les clés sont stockées dans le tuple sous-jacent. Pour les objets non-entités, les clés sont stockées de manière explicite dans l'objet parent.

L'ajout d'index sur un attribut relationship peut donner un coup d'accélérateur aux requêtes qui utilisent des références cycliques ou qui utilisent les filtres de requête IS NULL, IS EMPTY, SIZE ou MEMBER OF. Les associations, aussi bien monovaleur que multivaleur, peuvent comporter l'annotation @Index ou avoir une configuration de plug-in HashIndex dans le fichier XML du descripteur.

Définir des index de relations d'entités à l'aide de l'annotation @Index

L'exemple qui suit définit des entités avec des annotations @Index :

Entité avec annotation

```
@Entity
public class Node {
    @ManyToOne @Index
    Node parentNode;

    @OneToMany @Index
    List<Node> childrenNodes = new ArrayList();

    @OneToMany @Index
    List<BusinessUnitType> businessUnitTypes = new ArrayList();
}
```

Définir des index de relations d'entités à l'aide de XML

L'exemple qui suit définit les mêmes entités et les mêmes index, mais cette fois à l'aide de XML et avec des plug-in HashIndex :

Entité sans annotations

```
  public class Node {
  int nodeId;
  Node parentNode;
  List<Node> childrenNodes = new ArrayList();
  List<BusinessUnitType> businessUnitTypes = new ArrayList();
  }
```

XML ObjectGrid

```
<?xml version="1.0" encoding="UTF-8"?>
  <objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
  <objectGrid name="ObjectGrid_Entity" entityMetadataXMLFile="entity.xml>
  <backingMap name="Node" pluginCollectionRef="Node"/>
  <backingMap name="BusinessUnitType" pluginCollectionRef="BusinessUnitType"/>
  </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
  <backingMapPluginCollection id="Node">
  <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex" >
  <property name="Name" type="java.lang.String" value="parentNode"/>
```

```
  <property name="AttributeName" type="java.lang.String" value="parentNode"/>
<property name="RangeIndex" type="boolean" value="false"
  description="Ranges are not supported for association indexes." />  </bean>
  <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex" >
  <property name="Name" type="java.lang.String" value="businessUnitType"/>
  <property name="AttributeName" type="java.lang.String" value="businessUnitTypes"/>

<property name="RangeIndex" type="boolean" value="false"
  description="Ranges are not supported for association indexes." />
 </bean>
  <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex" >
  <property name="Name" type="java.lang.String" value="childrenNodes"/>
  <property name="AttributeName" type="java.lang.String" value="childrenNodes"/>
<property name="RangeIndex" type="boolean" value="false"
  description="Ranges are not supported for association indexes." />
  </bean>
  </backingMapPluginCollection>
  </backingMapPluginCollections>
  </objectGridConfig>
```

XML d'entités

```
  <?xml version="1.0" encoding="UTF-8"?>
  <entity-mappings xmlns="http://ibm.com/ws/projector/config/emd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/projector/config/emd ./emd.xsd">

  <description>My entities</description>
  <entity class-name="acme.Node" name="Account" access="FIELD">
  <attributes>
  <id name="nodeId" />
  <one-to-many name="childrenNodes"
  target-entity="acme.Node"
  fetch="EAGER" mapped-by="parentNode">
  <cascade><cascade-all/></cascade>
  </one-to-many>
  <many-to-one name="parentNodes"
  target-entity="acme.Node"
  fetch="LAZY" mapped-by="childrenNodes">
  <cascade><cascade-none/></cascade>
  </one-to-many>
  <many-to-one name="businessUnitTypes"
  target-entity="acme.BusinessUnitType"
  fetch="EAGER">
  <cascade><cascade-persist/></cascade>
  </many-to-one>
</attributes>
  </entity>
  <entity class-name="acme.BusinessUnitType" name="BusinessUnitType" access="FIELD">
  <attributes>
  <id name="buId" />
  <basic name="TypeDescription" />
  </attributes>
  </entity>
  </entity-mappings>
```

Avec les index définis plus haut, les exemples suivants de requêtes d'entités sont optimisés :

```
SELECT n FROM Node n WHERE n.parentNode is null
SELECT n FROM Node n WHERE n.businessUnitTypes is EMPTY
  SELECT n FROM Node n WHERE size(n.businessUnitTypes)>=10
  SELECT n FROM BusinessUnitType b, Node n WHERE b member of n.businessUnitTypes and b.name='TELECOM'
```

Définir des index de relations de non-entités

L'exemple qui suit définit un plug-in HashIndex pour des mappes de non-entités dans un fichier XML de descripteur d'ObjectGrid :

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="ObjectGrid_POJO">
      <backingMap name="Node" pluginCollectionRef="Node"/>
      <backingMap name="BusinessUnitType" pluginCollectionRef="BusinessUnitType"/>
      <querySchema>
        <mapSchemas>
          <mapSchema mapName="Node"
     valueClass="com.ibm.websphere.objectgrid.samples.entity.Node"
            primaryKeyField="id" />
          <mapSchema mapName="BusinessUnitType"
            valueClass="com.ibm.websphere.objectgrid.samples.entity.BusinessUnitType"
            primaryKeyField="id" />
        </mapSchemas>
        <relationships>
```

```
          <relationship source="com.ibm.websphere.objectgrid.samples.entity.Node"
            target="com.ibm.websphere.objectgrid.samples.entity.Node"
            relationField="parentNodeId" invRelationField="childrenNodeIds" />
          <relationship source="com.ibm.websphere.objectgrid.samples.entity.Node"
            target="com.ibm.websphere.objectgrid.samples.entity.BusinessUnitType"
            relationField="businessUnitTypeKeys" invRelationField="" />
        </relationships>
      </querySchema>
    </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
    <backingMapPluginCollection id="Node">
      <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex" >
        <property name="Name" type="java.lang.String" value="parentNode"/>
<property name="Name" type="java.lang.String" value="parentNodeId"/>
<property name="AttributeName" type="java.lang.String" value="parentNodeId"/>
<property name="RangeIndex" type="boolean" value="false"
  description="Ranges are not supported for association indexes." />
      </bean>
      <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex" >
        <property name="Name" type="java.lang.String" value="businessUnitType"/>
        <property name="AttributeName" type="java.lang.String" value="businessUnitTypeKeys"/>

<property name="RangeIndex" type="boolean" value="false"
  description="Ranges are not supported for association indexes." />
   </bean>
      <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex" >
<property name="Name" type="java.lang.String" value="childrenNodeIds"/>
  <property name="AttributeName" type="java.lang.String" value="childrenNodeIds"/>
  <property name="RangeIndex" type="boolean" value="false"
      description="Ranges are not supported for association indexes." />
</bean>
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

Compte tenu des configurations d'index ci-dessus, les exemples suivants de requêtes d'objets sont optimisés :

```
SELECT n FROM Node n WHERE n.parentNodeId is null
SELECT n FROM Node n WHERE n.businessUnitTypeKeys is EMPTY
SELECT n FROM Node n WHERE size(n.businessUnitTypeKeys)>=10
SELECT n FROM BusinessUnitType b, Node n WHERE
 b member of n.businessUnitTypeKeys and b.name='TELECOM'
```

Rechercher des partitions avec d'autres objets que des clés (interface PartitionableKey)

Lorsqu'une configuration eXtreme Scale utilise la stratégie FIXED_PARTITION de positionnement, elle dépend du hachage de la clé vers une partition pour insérer, lire, actualiser ou supprimer la valeur. La méthode hashCode est appelée sur la clé et elle doit être correctement définie si une clé personnalisée est créée. Mais il existe une autre possibilité : utiliser l'interface PartitionableKey. Cette interface permet d'utiliser un autre objet que la clé pour le hachage vers une partition.

Vous pouvez utiliser l'interface PartitionableKey dans des cas où il existe des mappes multiples et où les données que vous validez sont liées entre elles et doivent donc être mises dans la même partition. WebSphere eXtreme Scale ne prend pas en charge la validation en deux phases ; c'est pourquoi des transactions de mappes multiples ne doivent pas être validées si elles doivent s'étendre sur plusieurs partitions. Si PartitionableKey hache vers la même partition pour des clés situées dans différentes mappes du même groupe de mappes, ces transactions peuvent être validées ensemble.

Vous pouvez également utiliser l'interface PartitionableKey lorsque des groupes de clés doivent être placés dans la même partition, mais pas nécessairement au cours de la même transaction. Si les clés doivent être hachées en fonction du site, du département, du type de domaine ou de toute autre sorte d'identificateur, les clés enfants peuvent se voir attribuer une clé partitionnable parent.

Par exemple, les employés doivent hacher vers la même partition que leur département. Chaque clé d'employé aura un objet PartitionableKey qui appartient à la mappe Département. Alors, employés et département hacheront vers la même partition.

L'interface PartitionableKey fournit une seule méthode, appelée ibmGetPartition. L'objet retourné par cette méthode doit implémenter la méthode hashCode. Le résultat retourné de ce hashCode sera utilisé pour router la clé vers une partition.

Ecriture du code de transactions

Les applications qui nécessitent des transactions obligent à prendre en considération des réalités comme la gestion des verrous et des collisions ainsi que l'isolement des transactions.

Traitement des transactions

Sessions et traitement des transactions

WebSphere eXtreme Scale utilise les transactions comme mécanisme d'interaction avec les données.

Pour interagir avec les données, l'unité d'exécution de votre application requiert sa propre Session. Lorsque l'application souhaite utiliser ObjectGrid sur une unité d'exécution, appelez l'une des méthodes ObjectGrid.getSession pour obtenir une unité d'exécution. Avec la session, l'application peut travailler avec les données stockées dans les mappes ObjectGrid.

Lorsqu'une application utilise un objet Session, la session doit être dans le contexte d'une transaction. Une transaction commence et se valide ou commence et s'annule en utilisant les méthodes begin, commit et rollback sur l'objet Session. Les applications fonctionnent également en mode d'auto-validation : la session commence et valide automatiquement une transaction chaque fois qu'une opération est effectuée sur la mappe. Le mode d'auto-validation ne permet pas de regrouper des opérations multiples en une seule transaction. Il s'agit donc de l'option la plus lente si vous créez un lot d'opérations multiples dans une seule transaction. Cependant, pour les transactions contenant une seule opération, l'auto-validation est l'option la plus rapide.

Transactions

Les transactions disposent de nombreux avantages pour le stockage et la manipulation de données. Vous pouvez utiliser les transactions pour protéger la grille contre les changements simultanés, appliquer plusieurs changements comme unité simultanée, répliquer des données et implémenter un cycle de vie aux verrous appliqués aux changements.

Quand une transaction démarre, WebSphere eXtreme Scale alloue une mappe spéciale des différences pour contenir les changements en cours ou des copies des paires de valeur-clé que la transaction utilise. En règle générale, quand un accès à une paire valeur-clé se produit, la valeur est copiée avant que l'application ne reçoive la valeur. La mappe des différences contrôle tous les changements pour les opérations telles qu'insérer, mettre à jour, obtenir, supprimer, etc. Les clés ne sont pas copiées, car elles sont considérées comme non modifiables. Si un objet ObjectTransformer est spécifié, il est utilisé pour copier la valeur. Si la transaction utilise un verrouillage optimiste, les images précédentes des valeurs sont également suivies afin d'être comparées quand la transaction est validée.

Si une transaction est annulée, les informations de la mappe des différences sont supprimées et les verrous sur les entrées sont retirés. Quand une transaction est validée, les changements sont appliqués à la mappe et les verrous retirés. Si un verrouillage optimiste est utilisé, eXtreme Scale compare les versions des images précédentes des valeurs avec les valeurs qui se trouvent dans la mappe. Ces valeurs doivent correspondre pour que la transaction soit validée. Cette comparaison autorise un plan de verrouillage à version multiple, mais au prix de deux copies créées quand la transaction accède à l'entrée. Toute les valeurs sont à nouveau copiées et la nouvelle copie est stockée dans la mappe. WebSphere eXtreme Scale crée cette copie pour se protéger contre le fait que l'application change sa référence à la valeur après validation.

Il est possible de ne pas utiliser plusieurs copies des informations. L'application peut enregistrer une copie en utilisant un verrouillage pessimiste au lieu d'un verrouillage optimiste au prix d'une limitation des accès simultanés. La copie de la valeur au moment de la validation peut également être évitée si l'application accepte de ne pas changer la valeur après une validation.

Avantages des transactions

Utilisez les transactions pour les raisons suivantes :

Par le biais des transactions, vous pouvez :

- Annuler les modifications si une exception se produit ou si la logique application requiert l'annulation des changements d'état.
- Pour appliquer plusieurs changements en tant qu'unité atomique au moment de la validation
- Verrouiller et déverrouiller les données afin d'appliquer des changements multiples en tant qu'unité atomique au moment de la validation.
- Protéger une unité d'exécution de modifications simultanées.
- Implémenter un cycle de vie pour les verrous appliqués aux modifications.
- Produire une unité atomique de réplication.

Taille des transactions

Les transactions les plus grandes sont plus efficaces, en particulier pour la réplication. Cependant, les grandes transactions peuvent avoir un impact sur les accès simultanés car les verrous sur entrées sont maintenus plus longtemps. Si vous utilisez de grandes instructions, vous pouvez accroître les performances de réplication. Cette augmentation des performances est important lors du pré-chargement d'une mappe. Essayez différentes tailles de lots pour déterminer ce qui fonctionne le mieux pour votre scénario.

Les grandes transactions aident également avec les programmes de chargement. Si un chargeur utilisé peut effectuer du traitement par lots SQL, alors des gains de performances significatifs peuvent être réalisés sur la transaction, ainsi que des réductions de charges significatives du côté de la base de données. Ces gains de performances dépendent de l'implémentation du chargeur.

Mode de validation automatique

Si aucune transaction n'a démarré activement, quand une application interagit avec un objet ObjectMap, une opération automatique de démarrage et de validation est effectuée pour l'application. Cette opération fonctionne, mais elle empêche

l'annulation et le verrouillage de fonctionner efficacement. La vitesse de réplication synchrone est affectée à cause de la très petite taille des transactions. Si vous utilisez une application de gestion des entités, n'utilisez pas le mode de validation automatique car les objets recherchés avec la méthode EntityManager.find deviennent immédiatement non gérés au retour de la méthode et deviennent inutilisables.

Coordinateurs de transactions externes

En règle générale, les transactions commencent avec la méthode session.begin et se termine par la méthode session.commit. Cependant, quand eXtreme Scale est imbriqué, les transactions peuvent être démarrées et terminées par un coordinateur de transactions externes. Si vous en utilisez un, vous n'avez pas besoin d'appeler la méthode session.begin et de terminer avec la méthode session.commit. Pour plus d'informations sur eXtreme Scale et l'interaction des transactions externes, voir *Guide de programmation*. Si vous utilisez WebSphere Application Server, vous pouvez utiliser le plug-in WebSphereTransactionCallback. Voir *Guide de programmation* pour plus d'informations sur les plug-in disponibles avec WebSphere eXtreme Scale.

Attribut CopyMode

Vous pouvez ajuster le nombre de copies en définissant l'attribut CopyMode des objets BackingMap et ObjectMap.

Vous pouvez ajuster le nombre de copies en définissant l'attribut CopyMode des objets BackingMap et ObjectMap. Cet attribut a les valeurs suivantes :

- COPY_ON_READ_AND_COMMIT
- COPY_ON_READ
- NO_COPY
- COPY_ON_WRITE
- COPY_TO_BYTES

COPY_ON_READ_AND_COMMIT est la valeur par défaut. La valeur COPY_ON_READ copie les données initiales récupérées, mais ne copie pas au moment de la validation. Ce mode est sûr si l'application ne modifie pas une valeur après la validation d'une transaction. La valeur NO_COPY ne copie pas de données, ce qui n'est sûr que pour les données en lecture seule. Si les données ne changent jamais, vous n'avez alors pas besoin de les copier pour des raisons d'isolement.

Lorsque vous utilisez la valeur d'attribut NO_COPY, faites attention aux mappes pouvant être mises à jour. WebSphere eXtreme Scale utilise la copie en premier appui pour autoriser l'annulation de la transaction. L'application a changé uniquement la copie, et par conséquent, eXtreme Scale supprime la copie. Si la valeur d'attribut NO_COPY est utilisée et que l'application modifie la valeur validée, il est impossible de procéder à une annulation. La modification de la valeur validée génère des problèmes d'index, de réplication, etc. car les index et les fragments répliques se mettent à jour à la validation de la transaction. Si vous modifiez des données validées puis annulez la transaction, qui n'est pas vraiment annulée, alors les index ne sont pas mis à jour et les réplications ne se produisent pas. D'autres unités d'exécution peuvent voir les changements non validés immédiatement, même s'ils sont verrouillés. Utilisez la valeur d'attribut NO_COPY pour les mappes en lecture seule pour les applications qui effectuent la copie appropriée avant la modification de la valeur. Si vous utilisez la valeur d'attribut NO_COPY et que vous contactez le support technique IBM pour un problème

d'intégrité de données, il vous est demandé de reproduire le problème avec le mode de copie défini sur COPY_ON_READ_AND_COMMIT.

La valeur COPY_TO_BYTES stocke les valeurs dans la mappe dans un formulaire sérialisé. Au moment de la lecture, eXtreme Scale gonfle la valeur depuis un formulaire sérialisé et la stocke dans un autre au moment de la validation. Avec cette méthode, une copie est créée au moment de la lecture et au moment de la validation.

Le mode de copie par défaut pour une mappe est configurable sur l'objet BackingMap. Vous pouvez également changer le mode de copie sur les mappes avant de commencer une transaction en utilisant la méthode ObjectMap.setCopyMode.

Un exemple de fragment de mappe de sauvegarde provenant d'un fichier `objectgrid.xml` qui montre comment définir le mode de copie pour une mappe de sauvegarde donnée suit. Cet exemple part du principe que vous utilisez `cc` comme espace de noms `objectgrid/config`.

```
<cc:backingMap name="RuntimeLifespan" copyMode="NO_COPY"/>
```

Voir les informations concernant les meilleures pratiques de CopyMode dans le *Guide de programmation* pour plus d'informations.

Verrouillage d'entrées de mappe

Une mappe de sauvegarde ObjectGrid prend en charge plusieurs stratégies de verrouillage pour les mappes à des fins de cohérence des entrées de cache.

Chaque mappe de sauvegarde peut être configurée pour utiliser l'une de ces stratégies de verrouillage :

1. mode de verrouillage optimiste
2. mode de verrouillage pessimiste
3. aucune

La stratégie de verrouillage OPTIMISTIC est le mode par défaut. Utilisez le verrouillage optimiste lorsque les données sont modifiées rarement. Les verrous sont uniquement maintenus pendant un laps de temps limité tandis que les données sont lues depuis le cache et copiées dans la transaction. Lorsque le cache de transaction est synchronisé avec le cache principal, tous les objets mis en cache qui ont été mis à jour sont vérifiés avec la version d'origine. Si la vérification échoue, la transaction est annulée et une exception OptimisticCollisionException est provoquée.

La stratégie de verrouillage PESSIMISTIC obtient des verrous pour les entrées de cache et doit être utilisée lorsque les données sont modifiées fréquemment. A chaque lecture d'une entrée de cache, un verrou est obtenu et maintenu de façon conditionnelle jusqu'à la fin de la transaction. La durée de certains verrous peut être paramétrée à l'aide des niveaux d'isolement de transaction pour la session.

Si le verrouillage n'est pas obligatoire car les données ne sont jamais mises à jour ou le sont au cours de période calmes, vous pouvez le désactiver à l'aide de la stratégie de verrouillage NONE. Cette stratégie est très rapide car un gestionnaire de verrou n'est pas requis. La stratégie de verrouillage NONE est idéale pour les tables de recherche et les mappes en lecture seule.

Pour plus d'informations sur les stratégies de verrouillage, consultezles informations concernant les stratégies de verrouillage dans le *Présentation du produit*.

Spécification d'une stratégie de verrouillage

L'exemple ci-dessous illustre comment la stratégie de verrouillage peut être définie sur les mappes de sauvegarde map1, map2 et map3, où chaque mappe utilise une stratégie de verrouillage différente. Le premier fragment de code montre comment utiliser le langage XML pour la configuration des stratégies de verrouillage et le deuxième fragment de code illustre une approche à l'aide d'un programme.

Approche XML

Configuration des mappes de sauvegarde : exemple XML

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
    xmlns="http://ibm.com/ws/objectgrid/config">

    <objectGrids>
        <objectGrid name="test">
            <backingMap name="map1"
                lockStrategy="PESSIMISTIC" numberOfLockBuckets="31"/>
            <backingMap name="map2"
                lockStrategy="OPTIMISTIC" numberOfLockBuckets="409"/>
            <backingMap name="map3"
                lockStrategy="NONE"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

Approche à l'aide d'un programme

Configuration des mappes de sauvegarde : exemple à l'aide d'un programme

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.LockStrategy;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
...
ObjectGrid og =
 ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("test");
BackingMap bm = og.defineMap("map1");
bm.setLockStrategy( LockStrategy.PESSIMISTIC );
bm.setNumberOfLockBuckets(31);
bm = og.defineMap("map2");
bm.setNumberOfLockBuckets(409);
bm.setLockStrategy( LockStrategy.OPTIMISTIC );
bm = og.defineMap("map3");
bm.setLockStrategy( LockStrategy.NONE );
```

Pour éviter une exception java.lang.IllegalStateException, la méthode setLockStrategy doit être appelée avant d'appeler les méthodes initialize ou getSession sur une instance ObjectGrid locale.

Pour plus d'informations, consultez la rubrique relative aux stratégies de verrouillage dans le *Présentation du produit*.

Configuration du gestionnaire de verrou

Lors de l'utilisation d'une stratégie de verrouillage PESSIMISTIC ou OPTIMISTIC, un gestionnaire de verrou est créé pour la mappe de sauvegarde. Le gestionnaire de verrou utilise une mappe de hachage pour rechercher les entrées verrouillées

par une ou plusieurs transactions. S'il existe plusieurs entrées de mappe dans la mappe hash, plus les compartiments de verrouillage sont nombreux, plus les performances sont meilleures. Le risque de collision de synchronisation Java diminue lorsque le nombre de compartiments augmente. Plus les compartiments de verrouillage sont nombreux, plus les accès simultanés le sont également. Les exemples précédents montrent comment une application peut définir le nombre de compartiments de verrouillage à utiliser pour une instance BackingMap donnée.

Pour éviter une exception java.lang.IllegalStateException, la méthode setNumberOfLockBuckets doit être appelée avant d'appeler les méthodes initialize ou getSession sur une instance ObjectGrid. Le paramètre de la méthode setNumberOfLockBuckets est un entier primitif Java spécifiant le nombre de compartiments de verrouillage à utiliser. L'utilisation d'un nombre primitif peut permettre une distribution uniforme des entrées de mappe entre les compartiments de verrouillage. Pour optimiser les performances, commencez par définir le nombre de compartiments de verrouillage sur environ 10 % du nombre attendu d'entrées BackingMap.

LockDeadlockException

Voici un exemple de code illustrant l'interception de l'exception et le message résultant s'affiche ensuite.

```
try {
...
} catch (ObjectGridException oe) {
System.out.println(oe);
}
```

Le résultat est le suivant :

```
com.ibm.websphere.objectgrid.plugins.LockDeadlockException: _Message
```

Ce message représente la chaîne transmise en tant que paramètre lorsque l'exception est créée et émise.

Cause de l'exception

Le type le plus courant d'exception d'interblocage survient lorsque vous utilisez la stratégie de verrouillage pessimiste et que deux clients distincts possède chacun un verrou partagé sur un objet particulier. Les deux clients tentent ensuite de promouvoir un verrou exclusif sur cet objet. Le schéma ci-dessous illustre cette situation, notamment les blocs de transaction à l'origine de l'exception.

Le fragment de code Java ci-dessous montre comment transmettre un fichier XML de configuration pour créer une ObjectGrid.

Il s'agit d'un résumé de ce qu'il se produit dans votre programme lorsque cette exception survient. Dans une application impliquant plusieurs unités d'exécution mettant à jour la même ObjectMap, il est possible de rencontrer cette situation. Voici un exemple de deux clients exécutant les blocs de code de transaction, tel qu'illustré d dans la figure précédente.

Solutions possibles

Vous pouvez rencontrer la situation illustrée à la Figure 1 lorsque plusieurs unités d'exécution démarrent les transactions sur une mappe particulière. Dans ce cas, l'exception est émise pour éviter le blocage du programme. Vous pouvez définir un

rappel et ajouter du code dans le bloc catch pour obtenir plus de détails sur la cause. Etant donné que vous ne rencontrez cette exception que dans une stratégie de verrouillage pessimiste, l'utilisation d'une stratégie de verrouillage optimiste peut tout simplement remédier à la situation. Cependant, si vous avez besoin d'une stratégie de verrouillage pessimiste, vous pouvez utiliser la méthode getForUpdate et non la méthode get. Vous éviterez ainsi de recevoir des exceptions lors des situations décrites plus haut.

Stratégies de verrouillage

Les stratégies de verrouillage sont de type pessimiste, optimiste ou aucune. Pour choisir une stratégie de verrouillage, vous devez prendre en compte les aspects tels que le pourcentage des types d'opérations, l'utilisation ou non d'un chargeur, etc.

Les verrous sont liés aux transactions. Vous pouvez spécifier les paramètres de verrouillage suivants :

- **Aucun verrouillage** : l'exécution sans verrouillage est la plus rapide. Si vous utilisez des données en lecture seule, vous n'avez peut-être pas besoin de verrouillage.
- **Verrouillage pessimiste** : place des verrous sur les entrées, puis les maintient jusqu'au moment de la validation. Cette stratégie offre une bonne cohérence au prix de la capacité de traitement.
- **Verrouillage optimiste** : prend une image précédente de chaque enregistrement sur lequel la transaction appuie et compare l'image avec les valeurs d'entrées en cours quand la transaction est validée. Si les valeurs d'entrées changent, la transaction est annulée. Aucun verrou n'est maintenu jusqu'au moment de la validation. Cette stratégie de verrouillage offre un meilleur accès simultané que les stratégies pessimistes, au risque que la transaction soit annulée et au prix de la mémoire nécessaire pour une copie supplémentaire de l'entrée.

Définissez la stratégie de verrouillage sur la mappe de sauvegarde. Il n'est pas possible de changer de stratégie pour chaque transaction. Un fragment de code XML suit, montrant comment définir le mode de verrouillage sur une mappe à l'aide du fichier XML, en partant du principe que `cc` est le nom d'espace pour `objectgrid/config` :

```
<cc:backingMap name="RuntimeLifespan" lockStrategy="PESSIMISTIC" />
```

Verrouillage pessimiste

La stratégie de verrouillage pessimiste est à utiliser pour les opérations de mappe en lecture et en écriture lorsqu'aucune autre stratégie de verrouillage n'est possible. Lorsqu'une mappe ObjectGrid est configurée en mode de stratégie de verrouillage pessimiste, un verrou de transaction pessimiste est obtenu pour une entrée de mappe à la première transmission de l'entrée à la mappe de sauvegarde. Le verrou pessimiste est maintenu jusqu'à la fin de la transaction. La stratégie de verrouillage pessimiste est généralement utilisée dans les cas suivants :

- Lorsque la mappe de sauvegarde est configurée avec ou sans chargeur et que les informations sur les versions ne sont pas disponibles.
- Lorsque la mappe de sauvegarde est utilisée directement par une application qui nécessite l'assistance de eXtreme Scale pour le contrôle des accès simultanés.
- Lorsque les informations sur les versions sont disponibles mais que les transactions de mise à jour entrent régulièrement en conflit avec les entrées de sauvegarde, ce qui entraîne des échecs de mise à jour optimiste.

Etant donné que la stratégie de verrouillage pessimiste a un impact majeur sur les performances et l'évolutivité, elle doit uniquement être utilisée pour les mappes en

lecture et écriture lorsque les autres stratégies de verrouillage ne sont pas adaptées. Par exemple, ces situations peuvent être : échecs réguliers des mises à jour optimistes ou reprise après échec optimiste difficile à gérer pour une application.

Verrouillage optimiste

La stratégie de verrouillage optimiste présuppose qu'il ne peut se faire que deux transactions tentent d'actualiser la même entrée de mappe au même moment. A partir de ce principe, il n'est pas nécessaire de maintenir le mode de verrouillage pendant tout le cycle de vie de la transaction car il est peu probable que plusieurs transactions puissent actualiser la même entrée de mappe exactement au même moment. La stratégie de verrouillage optimiste est généralement utilisée dans les situations suivantes :

- Lorsqu'une mappe de sauvegarde est configurée avec ou sans chargeur et que les informations sur les versions sont disponibles.
- Lorsqu'une mappe de sauvegarde contient essentiellement des transactions qui exécutent des opérations de lecture. Les opérations insert, update ou remove sur les entrées de mappe ne sont pas fréquentes pour une mappe de sauvegarde.
- Lorsqu'une mappe de sauvegarde est insérée, mise à jour ou supprimée plus fréquemment qu'elle n'est lue mais que les transactions entrent rarement en conflit sur la même entrée de mappe.

A l'instar de la stratégie de verrouillage pessimiste, les méthodes dans l'interface ObjectMap déterminent comment eXtreme Scale tente automatiquement d'obtenir un mode de verrouillage pour l'entrée de mappe à laquelle l'accès est octroyé. Toutefois, les stratégies pessimiste et optimiste diffèrent sur les points suivants :

- Comme la stratégie de verrouillage pessimiste, un mode de verrou S est obtenu par les méthodes get et getAll lorsque la méthode est appelée. En revanche, avec le verrouillage optimiste, le mode de verrou S n'est maintenu que lorsque la transaction s'achève. Le mode de verrou S est libéré avant que la méthode soit renvoyée à l'application. L'objectif d'un mode de verrou consiste en ce que eXtreme Scale vérifie que seules les données validées provenant d'autres transactions soient visibles pour la transaction en cours. Une fois que eXtreme Scale a confirmé la validation des données, le mode de verrou S est libéré. Au moment de la validation, une vérification optimiste des versions est effectuée pour faire en sorte qu'aucune autre transaction n'a modifié l'entrée de mappe après la libération du mode de verrou S par la transaction en cours. Si une entrée n'est pas extraite d'une mappe avant d'être mise à jour, invalidée ou supprimée, l'exécution eXtreme Scale extrait implicitement l'entrée de la mappe. Cette opération get implicite est effectuée pour obtenir la valeur actuelle au moment de la demande de modification de l'entrée.
- Contrairement à la stratégie de verrouillage pessimiste, les méthodes getForUpdate et getAllForUpdate sont traitées exactement comme les méthodes get et getAll de la stratégie de verrouillage optimiste. Un mode de verrou S est obtenu au début de la méthode et est libéré avant d'être renvoyé à l'application.

Toutes les autres méthodes ObjectMap sont traitées exactement comme elles le sont dans la stratégie de verrouillage pessimiste. Lorsque la méthode commit est appelée, un mode de verrou X est obtenu pour toutes les entrées de mappe insérées, mises à jour, supprimées, corrigées ou invalidées et le mode de verrou X est maintenu jusqu'à ce que la transaction effectue le processus de validation.

La stratégie de verrouillage optimiste considère qu'aucune transaction simultanée ne tente de mettre à jour la même entrée de mappe. A partir de ce principe, le mode de verrouillage ne doit pas être maintenu pour le cycle de vie de la

transaction car il est peu probable qu'une transaction puisse mettre à jour simultanément la même entrée de mappe. Toutefois, étant donné qu'aucun mode de verrouillage n'est maintenu, une autre transaction simultanée peut potentiellement mettre à jour l'entrée de mappe après la libération du mode de verrou S par la transaction en cours.

Pour gérer cette possibilité, eXtreme Scale obtient un verrou X au moment de la validation et effectue une vérification optimiste des versions pour faire en sorte qu'aucune autre transaction n'a modifié l'entrée de mappe après la lecture de l'entrée de mappe de la mappe de sauvegarde par la transaction en cours. Si une autre transaction modifie l'entrée de mappe, la vérification de versions échoue et une exception OptimisticCollisionException se produit. Cette exception force l'annulation de la transaction en cours et l'application doit réessayer la transaction entière. La stratégie de verrouillage optimiste s'avère très utile lors qu'une mappe est principalement lue et que les mises à jour de la même entrée de mappe sont peu probables.

Aucun verrouillage

Lorsqu'une mappe de sauvegarde est configurée pour n'utiliser aucune stratégie de verrouillage, la valeur retournée est aucun verrou de transaction pour une entrée de mappe.

La non-utilisation d'une stratégie de verrouillage est utile lorsqu'une application est un gestionnaire de persistance tel qu'un conteneur EJB (Enterprise JavaBeans) ou qu'une application utilise Hibernate pour obtenir les données persistantes. Dans ce scénario, la mappe de sauvegarde est configurée sans chargeur et le gestionnaire de persistance utilise la mappe de sauvegarde comme cache de données. Dans ce scénario, le gestionnaire de persistance fournit le contrôle des accès simultanés entre les transactions qui accèdent aux mêmes entrées de mappe.

WebSphere eXtreme Scale n'a pas besoin d'obtenir de verrous de transaction à des fins de contrôle des accès simultanés. Cette situation considère que le gestionnaire de persistance ne libère pas ses verrous de transaction avant de mettre à jour la mappe ObjectGrid avec les modifications validées. Si le gestionnaire de persistance libère ses verrous, une stratégie de verrouillage pessimiste ou optimiste doit être utilisée. Par exemple, supposons que le gestionnaire de persistance d'un conteneur d'EJB met à jour une mappe ObjectGrid avec les données validées dans la transaction EJB gérée par conteneur. Si la mise à jour de la mappe ObjectGrid a lieu avant la libération des verrous du gestionnaire de persistance, vous pouvez utiliser la stratégie sans verrou. Si la mise à jour de la mappe ObjectGrid a lieu après la libération des verrous du gestionnaire de persistance, vous devez utiliser la stratégie optimiste ou pessimiste.

La stratégie sans verrou peut être utilisée également lorsque l'application utilise directement une mappe de sauvegarde et qu'un chargeur est configuré pour la mappe. Dans ce scénario, le chargeur utilise le fonction de contrôle des accès simultanés fournies par un système de gestion de base de données relationnelle (SGBDR) à l'aide de la connectivité JDBC (Java Database Connectivity) ou le plug-in Hibernate pour accéder aux données dans une base de données relationnelle. L'implémentation du chargeur peut utiliser une approche optimiste ou pessimiste. Un chargeur qui utilise une approche optimiste de verrouillage ou de gestion des versions contribue à optimiser les performances et l'accès simultané. Pour plus d'informations relatives à l'implémentation d'une approche de verrouillage optimiste, reportez-vous à la section OptimisticCallback de la rubrique les informations relatives aux remarques sur le chargeur dans le*Guide*

d'administration. Si vous utilisez un chargeur qui utilise le verrouillage pessimiste d'un programme d'arrière plan, il est conseillé d'utiliser le paramètre forUpdate transmis à la méthode get de l'interface Loader. Définissez ce paramètre sur true si la méthode getForUpdate de l'interface ObjectMap a été utilisée par l'application pour obtenir les données. Le chargeur peut se servir de ce paramètre pour déterminer s'il convient de demander un verrou pouvant être mis à niveau sur la ligne en cours de lecture. Par exemple, DB2 obtient un verrou pouvant être mis à niveau lorsqu'une instruction SQL select contient une clause `FOR UPDATE`. Cette approche offre la même protection contre les interblocages que celle décrite dans la rubrique«Verrouillage pessimiste», à la page 139.

JMS pour la répartition des modifications de transaction

Utilisez JMS (Java Message Service) pour les modifications de transaction répartie entre différents groupes de serveurs ou dans des environnements mixtes.

JMS est un protocole idéal pour les modifications réparties entre différents groupes de serveurs ou dans des environnements mixtes. Par exemple, certaines applications qui utilisent eXtreme Scale peuvent être déployées sur IBM WebSphere Application Server Community Edition, Apache Geronimo ou Apache Tomcat alors que d'autres applications peuvent être exécutées sur WebSphere Application Server version 6.x. JMS se prête parfaitement aux modifications réparties entre des homologues eXtreme Scale dans ces environnements différents. Les messages du gestionnaire de haute disponibilité sont transférés très rapidement mais peuvent uniquement répartir les modifications vers les machines virtuelles Java rassemblées dans un groupe central unique. JMS est plus lent mais il autorise le partage d'un ObjectGrid par des ensembles de clients d'applications plus importants et plus variés. JMS est adapté au partage de données dans un ObjectGrid entre un client Swing lourd et une application déployée sur WebSphere Extended Deployment.

Deux fonctionnalités pré-intégrées, le mécanisme d'invalidation client et la réplication entre homologues, sont des exemples de répartition de modifications transactionnelles JMS. Reportez-vous aux informations relatives à la configuration de la réplication entre homologues avec JMS du *Guide d'administration* pour plus de détails.

Implémentation de JMS

JMS est implémenté pour la répartition de modifications transactionnelles à l'aide d'un objet Java qui se comporte comme un ObjectGridEventListener. Cet objet peut propager l'état à l'aide de l'une des quatre méthodes suivantes :

1. Invalidate : toute entrée expulsée, mise à jour ou supprimée est retirée de toutes les machines virtuelles Java homologues à la réception du message.
2. Invalidate conditional : l'entrée est expulsée uniquement si la version locale est identique ou ultérieure à celle disponible sur le diffuseur de publications.
3. Push : toute entrée expulsée, mise à jour, supprimée ou insérée est ajoutée ou écrasée sur toutes les machines virtuelles Java homologues à la réception du message JMS.
4. Push conditional : l'entrée est uniquement mise à jour ou ajoutée côté récepteur si l'entrée locale est moins récente que la version en cours de publication.

Mode écoute pour les modifications de publication

Le plug-in implémente l'interface ObjectGridEventListener pour intercepter l'événement transactionEnd. Lorsque eXtreme Scale appelle cette méthode, le plug-in tente de convertir la liste LogSequence pour toutes les mappes concernées

par la transaction en un message JMS et ensuite de la publier. Le plug-in peut être configuré de façon à publier les modifications pour toutes mappes ou un sous-ensemble de mappes. Les objets LogSequence sont traités pour les mappes pour lesquelles la publication est activée. La classe LogSequenceTransformer de l'ObjectGrid sérialise vers un flux une liste LogSequence filtrée pour chaque mappe. Après sérialisation de toutes les listes LogSequence vers le flux, un message ObjectMessage JMS est créé et publié dans une rubrique connue.

Mode écoute pour les messages JMS et application à l'ObjectGrid locale

Le même plug-in lance une unité d'exécution qui tourne en boucle, recevant tous les messages publiés dans la rubrique connue. A l'arrivée d'un message, il transmet le contenu de ce dernier à la classe LogSequenceTransformer au niveau de laquelle il est converti en un ensemble d'objets LogSequence. Une transaction no-write-through est ensuite démarrée. Chaque objet LogSequence est fourni à la méthode Session.processLogSequence qui met à jour les mappes locales pour refléter les modifications. La méthode processLogSequence comprend le mode de répartition. La transaction est validée et le cache local reflète désormais les modifications. Pour plus d'informations sur l'utilisation de JMS pour la répartition de modifications transactionnelles, reportez-vous à la rubrique les informations relatives à la répartition des modifications entre les machines virtuelles Java homologues dans le *Guide d'administration*.

Transactions dans une partition unique et transactions dans plusieurs partitions de la grille

La différence majeure entre WebSphere eXtreme Scale et les solutions classiques de stockage de données (bases de données relationnelles ou bases de données en mémoire) consiste en l'utilisation du partitionnement, qui permet au cache d'évoluer de manière linéaire. Les principaux types de transactions sont les transactions impliquant une seule partition et les transactions impliquant toutes les partitions (d'une grille).

Les interactions avec le cache se rangent dans deux catégories : les transactions impliquant une seule partition et les transactions impliquant l'ensemble de la grille. Ces deux types de transactions sont décrits ci-dessous.

Transactions dans une partition unique

Les transactions dans une partition unique constituent le mode préférentiel d'interaction avec les mémoires cache hébergées par WebSphere eXtreme Scale. Lorsqu'une transaction est limitée à une partition, elle se limite par défaut à une seule machine virtuelle Java et donc à un seul serveur. Un serveur peut exécuter un nombre *M* de transactions par seconde. Si votre système contient *N* ordinateurs, vous pouvez exécuter `M*N` transactions par seconde. Si votre activité augmente et que vous ayez besoin de doubler le nombre de transactions par seconde, vous pouvez doubler *N* en installant davantage d'ordinateurs. Vous pouvez alors répondre à vos besoins en capacité sans modifier l'application, mettre à niveau le matériel ni utiliser l'application hors ligne.

Outre qu'elles permettent au cache d'évoluer de manière significative, les transactions s'exécutant dans une seule partition permettent aussi d'optimiser la disponibilité de ce dernier. Chaque transaction est liée à un seul ordinateur. Une défaillance peut se produire sur l'un des autres ordinateurs `(N-1)` sans que la réussite ou le temps de réponse de la transaction en soit affecté. Si 100 ordinateurs sont en cours d'exécution et que l'un d'eux échoue, 1 % des transactions en cours

au moment de l'échec sont annulées. Après cet échec, WebSphere eXtreme Scale relocalise les partitions qui sont hébergées par le serveur défaillant vers les 99 autres ordinateurs. Pendant cette courte période, ces ordinateurs continuent à exécuter des transactions. Seules les transactions impliquant les partitions qui sont en cours de relocalisation sont bloquées. Une fois le basculement terminé, le cache peut continuer à s'exécuter en étant pleinement opérationnel, c'est-à-dire à 99 % de sa capacité de traitement d'origine. Une fois le serveur défaillant remplacé et revenu dans la grille, le cache retrouve 100 % de sa capacité de traitement.

Transactions dans l'ensemble de la grille

En termes de performances, de disponibilité et d'évolutivité, les transactions s'exécutant dans l'ensemble de la grille sont l'opposé des transactions impliquant une seule partition. Elles accèdent à toutes les partitions et donc à chaque ordinateur de la configuration. Chaque ordinateur de la grille est sollicité pour rechercher des données, puis renvoyer le résultat. La transaction n'est pas terminée tant que tous les ordinateurs n'ont pas répondu. La capacité de traitement de la grille est donc limitée par l'ordinateur le plus lent. L'ajout d'ordinateurs ne permet pas d'améliorer la vitesse de l'ordinateur le plus lent ni d'augmenter la capacité de traitement du cache.

Les transactions impliquant l'ensemble de la grille ont des effets semblables sur la disponibilité. Reprenons l'exemple précédent : si 100 serveurs sont en cours d'exécution et que l'un d'eux échoue, 100 % des transactions en cours sont annulées. Après cet échec, WebSphere eXtreme Scale commence à relocaliser les partitions qui sont hébergées par ce serveur vers les 99 autres ordinateurs. En attendant la fin du basculement, la grille ne parvient à traiter aucune de ces transactions. Une fois le basculement terminé, le cache continue à s'exécuter, mais à un niveau de capacité réduit. Si chaque ordinateur de la grille gère 10 partitions, ces 10 ordinateurs reçoivent au moins une partition supplémentaire dans le cadre du basculement. L'ajout d'une partition supplémentaire augmente la charge de travail de cet ordinateur d'au moins 10 %. La capacité de la grille de traitement étant limitée à la capacité de traitement de l'ordinateur le plus lent, cette capacité est limitée en moyenne de 10 %.

Les transactions s'exécutant dans une partition unique sont préférables aux partitions impliquant l'ensemble de la grille pour garantir l'évolutivité d'un cache objet réparti hautement disponible comme WebSphere eXtreme Scale. Pour optimiser les performance de ces systèmes, vous devez utiliser des techniques autres que les méthodologies relationnelles traditionnelles, mais vous pouvez transformer les transactions impliquant l'ensemble de la grille en transactions s'exécutant dans une seule partition.

Méthodes recommandées en matière de génération de modèles de données évolutifs

Les méthodes recommandées pour générer des applications évolutives avec des produits tels que WebSphere eXtreme Scale sont au nombre de deux : les principes fondamentaux et les conseils relatifs à l'implémentation. Les principes fondamentaux sont les grandes idées que vous devez capturer lors de la conception des données. Une application n'observant pas ces principes aura peu de chance de pouvoir évoluer de manière satisfaisante, même pour les transactions principales. Les conseils d'implémentation s'appliquent aux transactions posant problème dans une application par ailleurs bien conçue et observant les principes généraux relatifs aux modèles de données évolutifs.

Principes fondamentaux

Certains concepts ou principes de base à ne pas oublier constituent les éléments clés pour optimiser l'évolutivité.

Duplication plutôt que normalisation

Il est essentiel de bien comprendre que les produits tels que WebSphere eXtreme Scale sont conçus pour répartir des données dans un grand nombre d'ordinateurs. Si votre objectif est d'exécuter la plupart ou l'ensemble des transactions sur un même ordinateur, le modèle de données doit s'assurer que toutes les données nécessaires à la transaction sont situées dans cette partition. Dans la plupart des cas, la seule manière d'y parvenir consiste à dupliquer les données.

Prenons l'exemple d'un forum électronique. Deux transactions très importantes pour ce forum présentent tous les messages publiés par un utilisateur donné et tous les messages publiés sur un sujet donné. Imaginez tout d'abord comment ces transactions se comporteraient dans le cadre d'un modèle de données normalisé contenant un enregistrement utilisateur, un enregistrement relatif au sujet et un enregistrement relatif au message et contenant le texte réel. Si les messages publiés sont partitionnés avec les enregistrements utilisateur, l'affichage du sujet devient une transaction impliquant l'ensemble de la grille, et inversement. Il est impossible de partitionner les sujets et les utilisateurs car leur relation est de type many-to-many.

La meilleure méthode pour permettre à ce forum électronique d'évoluer est de dupliquer les messages publiés, c'est-à-dire de copier chaque message avec l'enregistrement relatif au sujet et avec l'enregistrement utilisateur. L'affichage des messages publiés à partir d'un utilisateur constitue alors une transaction dans une partition unique, de même que l'affichage des messages publiés dans un sujet, tandis que la mise à jour ou la suppression d'un message publié est une transaction impliquant deux partitions. Ces trois transactions évoluent de manière linéaire au fur et à mesure que le nombre d'ordinateurs de la grille augmente.

L'évolutivité plutôt que les ressources

Le principal obstacle à surmonter en matière de modèles de données dénormalisés est l'impact de ces modèles sur les ressources. La conservation de deux ou trois copies, voire plus, de certaines données risque d'entraîner la consommation d'une trop grande quantité de ressources pour être pratique. Lorsque vous êtes confronté à un tel scénario, gardez ceci en mémoire : les coûts liés à l'achat de matériel diminuent d'année en année. Autre considération, et non des moindres : WebSphere eXtreme Scale permet de supprimer la plupart des coûts associés au déploiement de ressources supplémentaires.

Mesurez les ressources en termes de coût plutôt qu'en termes purement informatiques comme les mégaoctets et les processeurs. Les magasins de données utilisant des données relationnelles normalisées doivent généralement être situés sur le même ordinateur. Cela signifie que vous devez acheter un seul ordinateur d'entreprise puissant, plutôt que plusieurs machines modestes. Il n'est pas rare qu'un ordinateur d'entreprise pouvant exécuter un million de transactions par seconde soit bien plus onéreux que dix ordinateurs pouvant traiter 100 000 transactions par seconde.

L'ajout de ressources a également un coût. Une entreprise en pleine croissance finit par manquer de capacité. Lorsque vous vous trouvez dans cette situation, vous devez soit arrêter votre système lors du passage à un ordinateur plus rapide et plus puissant, soit créer un second environnement de production. Quelle que soit l'option choisie, vous devrez prendre en charge des coûts supplémentaires liés à la réduction du volume de traitement ou au maintien de la capacité de traitement, pratiquement doublée pendant la durée de la transaction.

Avec WebSphere eXtreme Scale, il n'est pas nécessaire de fermer l'application lors de l'accroissement de la capacité. Si les prévisions relatives à votre entreprise indiquent que vous devez augmenter de 10 % votre capacité pour l'année à venir, augmentez d'autant le nombre d'ordinateurs de la grille. Ce pourcentage peut augmenter sans entraîner d'indisponibilité de l'application et sans que vous ayez à accroître la capacité.

Des transformations de données inutiles

Lorsque vous utilisez WebSphere eXtreme Scale, vous devez stocker les données dans un format directement utilisable par la logique métier. La répartition des données dans un format plus primitif consomme davantage de ressources. La transformation doit se faire lors de l'écriture et lors de la lecture des données. Dans le cas d'une base de données relationnelle, cette transformation est nécessaire car les données sont enregistrées fréquemment sur le disque, mais avec WebSphere eXtreme Scale, elle n'est pas obligatoire. La plupart des données sont stockées en mémoire et peuvent donc être stockées au format requis par l'application.

L'observation de cette règle simple vous aide à dénormaliser les données conformément au premier principe. Le type de transformation des données métier le plus commun est l'opération JOIN nécessaire pour transformer des données normalisées en un ensemble de résultats correspondant aux besoins de l'application. Le stockage des données au format correct permet implicitement d'éviter d'avoir à exécuter ces opérations de type JOIN et génère un modèle de données dénormalisé.

Abandon des requêtes illimitées

Quelle que soit la structure de vos données, les requêtes illimitées évoluent mal. Nous ne vous recommandons par exemple pas d'exécuter une transaction visant à obtenir une liste de tous les articles triés par valeur. Elle peut renvoyer un résultat correct au début, lorsque le nombre d'articles est égal à 1 000, mais lorsque celui-ci atteint 10 millions, la transaction renvoie ces 10 millions d'articles. L'exécution d'une telle transaction risque d'entraîner un délai d'inactivité de celle-ci ou une erreur liée à une insuffisance de mémoire du client.

La meilleure solution consiste à modifier la logique métier de façon que seuls les 10 ou 20 vingt premiers articles soient renvoyés. Cette modification permet de faire en sorte que la transaction soit gérable quel que soit le nombre d'articles présents en cache.

Définition d'un schéma

Le principal avantage lié à la normalisation des données est que la base de données peut prendre en charge la cohérence des données en arrière-plan. Lorsque les données sont dénormalisées à des fins d'évolutivité, la gestion automatique de la cohérence des données disparaît. Pour la rétablir, vous

devez implémenter un modèle de données pouvant fonctionner dans la couche d'applications ou en tant que plug-in de la grille répartie.

Prenons l'exemple du forum électronique. Si une transaction supprime un message publié d'un sujet, sa copie dans l'enregistrement utilisateur doit également être supprimée. Sans modèle de données, il est possible que le développeur prévoie la suppression du message publié dans le code de l'application, mais qu'il oublie de le supprimer de l'enregistrement utilisateur. Toutefois, s'il avait utilisé un modèle de données au lieu d'interagir directement avec le cache, la méthode removePost aurait permis d'extraire l'ID utilisateur du message, de rechercher l'enregistrement utilisateur et de supprimer la copie du message en arrière-plan.

Vous pouvez également implémenter un programme d'écoute s'exécutant sur la partition et capable de détecter la modification apportée au sujet et de modifier automatiquement l'enregistrement utilisateur. Un tel programme peut se révéler avantageux car la modification de l'enregistrement utilisateur est effectuée localement si celui-ci se trouve sur la partition ou, s'il se trouve sur une autre partition, la transaction est exécutée entre deux serveurs et non entre le client et le serveur. La connexion réseau entre des serveurs est probablement plus rapide que la connexion existant entre le client et le serveur.

Disparition des conflits

Evitez les scénarios contenant par exemple un compteur global. La grille ne parvient pas à évoluer si un enregistrement unique est utilisé un nombre disproportionné de fois par rapport aux autres enregistrements. Les performances de la grille seront limitées par les performances de l'ordinateur détenant cet enregistrement.

Dans une telle situation, essayez de fractionner l'enregistrement afin qu'il soit géré au niveau de la partition. Prenons le cas d'une transaction renvoyant le nombre total d'entrées présentes dans le cache réparti. Plutôt que d'exécuter une opération d'insertion et de suppression accédant à un enregistrement unique qui s'incrémente, installez un programme d'écoute sur chaque partition pour suivre ces opérations. Grâce à ce suivi, les opérations d'insertion et de suppression deviennent des transactions s'exécutant dans une partition unique.

La lecture du compteur devient une opération impliquant l'ensemble de la grille, mais elle était déjà en grande partie aussi inefficace qu'une telle opération car ses performances étaient liées à celles de l'ordinateur hébergeant l'enregistrement.

Conseils relatifs à l'implémentation

Pour optimiser l'évolutivité, prenez en compte les conseils suivants.

Utilisation des index de recherche inversée

Prenons le cas d'un modèle de données dénormalisé dans lequel les enregistrements client sont partitionnés en fonction du numéro d'ID du client. Cette méthode de partitionnement est un choix logique car pratiquement toutes les opérations métier exécutées avec l'enregistrement client utilisent cet ID. Toutefois, la transaction de connexion, qui est essentielle, ne l'utilise pas. Les données utilisées pour la connexion sont plus fréquemment le nom d'utilisateur ou l'adresse électronique.

L'approche la plus simple consiste alors à utiliser une transaction impliquant l'ensemble de la grille pour rechercher l'enregistrement client. Comme expliqué précédemment, cette approche n'est pas évolutive.

Autre option : procéder à un partitionnement en fonction du nom d'utilisateur ou de l'adresse électronique. Cette option n'est pas pratique car toutes les opérations liées à l'ID client deviendraient alors des transactions impliquant l'ensemble de la grille. De plus, les clients de votre site pourraient souhaiter modifier leur nom d'utilisateur ou leur adresse électronique. Dans les produits tels que WebSphere eXtreme Scale, la valeur utilisée pour partitionner les données doit rester constante.

La bonne solution consiste alors à utiliser un index de recherche inversée. Avec WebSphere eXtreme Scale, il est possible de créer un cache dans la même grille répartie que le cache contenant tous les enregistrements utilisateur. Ce cache est à haute disponibilité, partitionnée et évolutif. Il permet de mapper un nom d'utilisateur ou une adresse électronique vers un ID client. Plutôt que d'avoir une opération impliquant l'ensemble de la grille, il transforme la procédure de connexion en transaction s'exécutant sur deux partitions. Ce scénario n'est pas aussi optimal qu'une transaction impliquant une seule partition, mais la capacité de traitement augmente cependant de manière linéaire par rapport au nombre d'ordinateurs.

Calcul des valeurs lors de l'écriture

Les calculs les plus fréquents, tels que les moyennes ou les totaux, peuvent consommer une grande quantité de ressources car ils supposent la lecture d'un grand nombre d'entrées. Les lectures étant plus fréquentes que les écritures dans la plupart des applications, il est plus efficace de calculer ces valeurs lors de l'écriture, puis de stocker le résultat dans le cache. Les opérations gagnent ainsi en rapidité et en évolutivité.

Zones facultatives

Prenons l'exemple d'un enregistrement utilisateur contenant un numéro de téléphone professionnel, un numéro de téléphone personnel et un numéro de téléphone portable. Tous ces numéros ou une combinaison d'entre eux (ou aucun) peuvent être définis pour un utilisateur. Si les données ont été normalisées, une table utilisateur et une table contenant les numéros de téléphone existent. Vous pouvez alors identifier les numéros de téléphone d'un utilisateur donné à l'aide d'une opération JOIN entre les deux tables.

Pour dénormaliser cet enregistrement, aucune duplication des données n'est nécessaire, car la plupart des utilisateurs ne partagent pas leurs numéros de téléphone. Au contraire, les emplacements vides doivent être autorisés dans l'enregistrement utilisateur. Au lieu de constituer une table contenant les numéros de téléphone, ajoutez trois attributs à l'enregistrement utilisateur, chacun correspondant à un type de numéro de téléphone. L'ajout de ces attributs rend superflue l'opération JOIN et permet d'effectuer la recherche des numéros de téléphone d'un utilisateur en tant qu'opération impliquant une seule partition.

Positionnement des relations many-to-many

Prenons l'exemple d'une application qui assure le suivi de certains produits et des magasins dans lesquels ceux-ci sont commercialisés. Un même produit est vendu dans plusieurs magasins et un même magasin vend plusieurs produits. Supposons que cette application assure le suivi de 50 revendeurs importants. Chaque produit est vendu dans 50 magasins au maximum, chaque magasin commercialisant des milliers de produits.

Constituez une liste des magasins dans l'entité produit (organisation A) plutôt qu'une liste des produits dans chaque entité magasin (organisation B). Une simple observation des transactions que cette application devrait exécuter permet de comprendre pourquoi l'organisation A est la plus évolutive.

Considérons d'abord les mises à jour. Dans l'organisation A, la suppression d'un produit du stock d'un magasin verrouille l'entité produit. Si la grille contient 10 000 produits, seul 1/10 000è de la grille doit être verrouillé lors de la mise à jour. Dans l'organisation B, la grille contient 50 magasins, si bien qu'1/50è de la grille doit être verrouillé pendant la mise à jour. Même si les deux opérations peuvent être considérées comme des opérations impliquant une seule partition, l'organisation A permet une meilleure évolutivité.

Considérons maintenant les opérations de lecture avec l'organisation A : la recherche des magasins dans lesquels un produit est commercialisé est une transaction impliquant une seule partition, pouvant évoluer et rapide car elle transmet une faible quantité de données. Avec l'organisation B, cette transaction devient une transaction impliquant plusieurs grilles car chaque entité de magasin doit faire l'objet d'un accès permettant d'identifier si le produit est vendu dans ce magasin, ce qui donne un avantage certain à l'organisation A en termes de performances.

Evolutivité avec les données normalisées

Les transactions impliquant l'ensemble de la grille sont utilisées à juste titre pour faire évoluer le traitement des données. Si une grille contient cinq ordinateurs et qu'une transaction visant à trier environ 100 000 enregistrements dans chaque ordinateur est lancée sur l'ensemble de la grille, cette transaction trie 500 000 enregistrements. Si l'ordinateur le plus lent peut traiter 10 de ces transactions par seconde, la grille peut trier 5 millions d'enregistrements par seconde. Si les données de la grille doublent, chaque ordinateur doit trier 200 000 enregistrements et chaque transaction trie 1 million d'enregistrements. Cette progression des données ramène la capacité de traitement de l'ordinateur le plus lent à cinq transactions par seconde et celle de la grille à cinq transactions par seconde. La grille trie toutefois 5 millions d'enregistrements par seconde.

Dans un tel scénario, le fait de doubler le nombre d'ordinateurs permet à chaque ordinateur de revenir à sa charge précédente et à l'ordinateur le plus lent de traiter 10 de ces transactions par seconde. La capacité de traitement de la grille reste la même (10 demandes par seconde), mais chaque transaction traite désormais 1 million d'enregistrements, si bien que la grille a doublé sa capacité en termes de traitement d'enregistrements, atteignant les 10 millions d'enregistrements par seconde.

Avec les applications telles que les moteurs de recherche, qui ont besoin d'évoluer en termes de traitement des données pour s'adapter à la taille croissante de l'Internet et en termes de capacité afin de s'adapter à la croissance du nombre d'utilisateurs, vous devez créer plusieurs grilles avec permutation circulaire des demandes entre les grilles. Si vous avez besoin de faire évoluer la capacité de traitement, ajoutez des ordinateurs et ajoutez une grille pour gérer les demandes. Si le traitement des données doit évoluer, ajoutez des ordinateurs et conservez le même nombre de grilles.

Gestion des verrous

Les verrous comportent des cycles de vie et leurs différents types sont compatibles entre eux selon plusieurs critères. Les verrous doivent être traités dans un ordre approprié pour éviter les situations d'interblocage.

Expiration des verrous

Chaque interface BackingMap est associée à un délai d'expiration par défaut du verrou. La valeur de ce délai d'expiration permet de faire en sorte qu'une application n'attende pas sans fin l'octroi d'un mode de verrouillage en raison d'une situation d'interblocage due à une erreur de l'application. L'application peut utiliser l'interface BackingMap pour remplacer le délai d'expiration par défaut du verrou. L'exemple ci-dessous illustre comment définir la valeur du délai d'expiration du verrou pour la mappe de sauvegarde map1 sur 60 secondes :

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.LockStrategy;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
...
ObjectGrid og =
 ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("test");
BackingMap bm = og.defineMap("map1");
bm.setLockStrategy( LockStrategy.PESSIMISTIC );
bm.setLockTimeout( 60 );
```

Pour éviter une exception java.lang.IllegalStateException, appelez les méthodes setLockStrategy et setLockTimeout avant d'appeler la méthode initialize ou getSession sur l'instance ObjectGrid. Le paramètre de la méthode setLockTimeout est un entier primitif Java spécifiant le délai en secondes au bout duquel eXtreme Scale autorise l'octroi d'un mode de verrouillage. Si le délai d'exécution d'une transaction excède la valeur du délai d'expiration du verrou configurée pour la mappe de sauvegarde, une exception com.ibm.websphere.objectgrid.LockTimeoutException est déclenchée.

Lorsqu'une exception LockTimeoutException se produit, l'application doit déterminer si le dépassement du délai est dû à une exécution de l'application plus lente que prévu ou à une situation d'interblocage. Si une situation d'interblocage réelle a lieu, l'augmentation de la valeur du délai d'expiration du verrou n'élimine pas l'exception. L'augmentation du délai d'expiration entraîne uniquement le retard de l'exécution de l'exception. Toutefois, l'augmentation de la valeur du délai d'expiration du verrou élimine l'exception, cela signifie que le problème est survenu parce que l'exécution de l'application était plus lente que prévu. Dans ce cas, l'application doit déterminer pourquoi les performances sont lentes.

Délai d'expiration du verrou pour les ObjectMaps

Le délai d'expiration du verrou peut être substitué pour une seule instance ObjectMap en utilisant la méthode ObjectMap.setLockTimeout. La valeur du délai d'expiration du verrou affecte toutes les transactions démarrées après la définition de la nouvelle valeur de délai d'expiration. Cette méthode peut être utilisée lors les conflits de verrou sont possibles ou prévisibles dans les transactions select.

Verrou partagé, pouvant être mis à niveau et exclusif

Lorsqu'une application appelle une méthode de l'interface ObjectMap, utilise les méthodes find sur un index ou exécute une requête, eXtreme Scale tente automatiquement d'obtenir un verrou pour la mappe en cours d'accès. WebSphere

eXtreme Scale utilise les modes de verrouillage suivants en fonction de la méthode avec laquelle l'application appelle l'interface ObjectMap.

- Les méthodes get et getAll de l'interface ObjectMap, les méthodes d'index et les requêtes obtiennent un *verrou S*, ou un mode de verrou partagé pour la clé d'une entrée de mappe. La durée pendant laquelle le verrou S est maintenu dépend du niveau d'isolement de transaction utilisé. Un mode de verrou S permet l'accès simultané entre les transactions qui tentent d'obtenir un verrou S ou un verrou pouvant être mis à niveau (verrou U) pour la même clé mais bloque d'autres transactions qui tentent d'obtenir un verrou exclusif (verrou X) pour la même clé.
- Les méthodes getForUpdate et getAllForUpdate obtiennent un *verrou U* ou un mode de verrou pouvant être mis à niveau pour la clé d'une entrée de mappe. Le verrou U est maintenu jusqu'à la fin de la transaction. Un mode de verrou U permet l'accès simultané entre les transactions qui obtiennent un verrou S pour la même clé mais bloque d'autres transactions qui tentent d'obtenir un verrou U ou un mode de verrou X pour la même clé.
- Les opérations put, putAll, remove, removeAll, insert, update et touch obtiennent un *verrou X* ou le mode de verrou exclusif pour la clé d'une entrée de mappe. Le verrou X est maintenu jusqu'à la fin de la transaction. Un mode de verrou X fait en sorte qu'une seule transaction insère, mette à jour ou supprime une entrée de mappe d'une valeur de clé donnée. Un verrou X bloque toutes les autres transactions qui tentent d'obtenir un verrou S, U ou X pour la même clé.
- Les méthodes global invalidate et global invalidateAll obtiennent un verrou X pour chaque entrée de mappe invalidée. Le verrou X est maintenu jusqu'à la fin de la transaction. Aucun verrou n'est obtenu pour les méthodes local invalidate et local invalidateAll car aucune des entrées de la mappe de sauvegarde n'est invalidée par les appels de méthode local invalidate.

Eu égard aux définitions précédentes, il est évident qu'un mode de verrouillage S est plus faible qu'un mode de verrouillage U car ce mode permet l'exécution simultanée d'un nombre plus important de transactions lors de l'accès à la même entrée de mappe. Le mode de verrouillage U est légèrement plus fort que le mode de verrouillage S lock car il bloque les autres transactions qui demandent un mode de verrouillage U ou X. Le mode de verrouillage S bloque uniquement les autres transactions qui demandent un mode de verrouillage X. Cette petite différence est pourtant importante lorsqu'il s'agit d'empêcher l'occurrence de certaines interblocages. Le mode de verrouillage X est le mode de verrouillage le plus fort car il bloque toutes les autres transactions en tentant d'obtenir un mode de verrouillage S, U ou X pour la même entrée de mappe. L'effet d'un mode de verrouillage X consiste à garantir qu'une seule transaction peut insérer, mettre à jour ou supprimer une entrée de mappe et à empêcher la perte des mise à jour lorsque plusieurs transactions tentent de mettre à jour la même entrée de mappe.

Le tableau ci-dessous est une matrice de compatibilité des modes de verrouillage qui résume les modes de verrouillage décrits et qui sert à déterminer la compatibilité entre les différents modes. Pour lire cette matrice, la ligne dans la matrice indique un mode de verrouillage déjà octroyé. La colonne décrit le mode de verrouillage demandé par une autre transaction. Si la valeur Oui s'affiche dans la colonne, cela signifie que le mode de verrouillage demandé par l'autre transaction est octroyé car il est compatible avec celui qui a déjà été octroyé. La valeur Non indique que le mode de verrouillage n'est pas compatible et que l'autre transaction doit attendre que la première transaction libère le verrou dont elle est propriétaire.

Tableau 4. Matrice de compatibilité du mode verrouillage

Verrou	Verrou type S (partagé)	Verrou type U (pouvant être mis à niveau)	Verrou type X (exclusif)	Puissance
S (partagé)	Oui	Oui	Non	le plus faible
U (pouvant être mis à niveau)	Oui	Non	Non	normal
X (exclusif)	Non	Non	Non	le plus fort

Interblocages de verrouillage

Examinez la séquence de demandes de mode de verrouillage suivante :

1. Verrou X octroyé à la transaction 1 pour key1.
2. Verrou X octroyé à la transaction 2 pour key2.
3. Verrou X demandé par la transaction 1 pour key2. (La transaction 1 se bloque en attente du verrou acquis par la transaction 2.)
4. Verrou X demandé par la transaction 2 pour key1. (La transaction 2 se bloque en attente du verrou acquis par la transaction 1.)

La séquence précédente est l'exemple type d'un interblocage de deux transactions qui tentent d'acquérir plusieurs verrous et des transactions qui obtiennent les verrous dans un ordre différent. Pour éviter cet interblocage, chaque transaction doit obtenir plusieurs verrous dans le même ordre. Si la stratégie de verrouillage OPTIMISTE est utilisée et la méthode flush sur l'interface ObjectMap n'est jamais utilisée par l'application, les modes de verrouillage sont demandés par la transaction uniquement lors du cycle de validation. Pendant le cycle de validation, eXtreme Scale détermine les clés pour les entrées de mappe qui doivent être verrouillées et demande les modes de verrouillage en une séquence de clés (comportement déterministe). Avec cette méthode, eXtreme Scale évite la majorité des interblocages classiques. Toutefois, eXtreme Scale n'empêche pas et ne peut pas empêcher tous les interblocages. Quelques scénarios doivent être pris en compte par l'application. Voici les cas dont l'application doit tenir compte et pour lesquels elle doit prendre des mesures préventives.

Un scénario existe où eXtreme Scale est capable de détecter un interblocage sans avoir à attendre l'expiration du délai d'attente du verrou. Si ce scénario se produit, une exception com.ibm.websphere.objectgrid.LockDeadlockException est émise. Examinez le fragment de code suivant :

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
sess.begin();
Person p = (IPerson)person.get("Lynn");
// Lynn a fêté son anniversaire, donc nous l'avons vieilli d'1 an.
p.setAge( p.getAge() + 1 );
person.put( "Lynn", p );
sess.commit();
```

Dans cette situation, l'ami de Lynn veut la vieillir d'un an et Lynn et son ami exécutent cette transaction simultanément. Dans cette situation, les deux transactions possèdent un mode de verrou S sur l'entrée Lynn de la mappe PERSON suite à l'appel de la méthode person.get("Lynn"). En raison de l'appel de la méthode person.put ("Lynn", p), les deux transactions tentent de mettre à niveau le mode de verrou S vers un mode de verrou X. Les deux transactions se bloquent dans l'attente de libération du mode de verrou S par l'autre transaction. Par conséquent, une interblocage se produit car une condition d'attente circulaire existe entre les deux transactions. Une condition d'attente circulaire a lieu lorsque

plusieurs transactions tentent de promouvoir un verrou d'un mode faible vers un mode fort pour la même entrée de mappe. Dans ce scénario, une exception LockDeadlockException est émise au lieu d'une exception LockTimeoutException.

L'application peut empêcher l'exception LockDeadlockException pour l'exemple précédent à l'aide de la stratégie de verrouillage optimiste au lieu de la stratégie de verrouillage pessimiste. La stratégie de verrouillage optimiste constitue la solution privilégiée lors que la mappe est essentiellement lue et que les mises à jour ne sont pas fréquentes. Si la stratégie de verrouillage pessimiste doit être utilisée, la méthode getForUpdate peut être utilisée à la place de la méthode get de l'exemple ci-dessus ou un niveau d'isolement de transaction TRANSACTION_READ_COMMITTED peut être utilisé.

Pour plus d'informations, consultez la rubrique relative aux stratégies de verrouillage dans la *présentation du produit*.

L'utilisation du niveau d'isolement de transaction TRANSACTION_READ_COMMITTED empêche le verrou S acquis par la méthode get d'être maintenu jusqu'à la fin de la transaction. Si la clé n'est jamais invalidée dans le cache transactionnel, les lectures reproductibles sont toujours garanties.

Pour plus d'informations, consultez la rubrique relative au verrouillage des entrées de mappe dans le *guide d'administration*

Pour changer le niveau d'isolement, vous pouvez aussi utiliser la méthode getForUpdate. La première transaction pour appeler la méthode getForUpdate obtient un mode de verrouillage U et non un mode de verrouillage S. Ce mode de verrouillage entraîne le blocage de la deuxième transaction lors de l'appel de la méthode getForUpdate car un mode de verrouillage U n'est octroyé qu'à une seule transaction. En raison du blocage de la deuxième transaction, cette dernière ne possède aucun mode de verrouillage sur l'entré de mappe Lynn. La première transaction ne se bloque pas lorsqu'elle tente de mettre à niveau le mode de verrouillage U vers le mode de verrouillage X après l'appel de la méthode put à partir de la première transaction. Cette fonction démontre pourquoi le mode de verrouillage U est appelé le mode de verrouillage *pouvant être mis à niveau*. Lorsque la première transaction est terminée, la deuxième se débloque et le mode de verrouillage U lui est octroyé. Une application peut empêcher le scénario d'interblocage de la promotion de verrouillage en utilisant la méthode getForUpdate au lieu de la méthode get en cas d'utilisation de la stratégie de verrouillage pessimiste.

Important : Cette solution n'empêche pas les transactions en lecture seule de lire une entrée de mappe. Les transactions en lecture seule appellent la méthode get mais jamais les méthodes put, insert, update et remove. L'accès simultané est aussi élevé que lors de l'utilisation de la méthode get classique. Il s'affaiblit uniquement lorsque la méthode getForUpdate est appelée par plusieurs transactions pour la même entrée de mappe.

Vous devez garder cela à l'esprit lorsqu'une transaction appelle la méthode getForUpdate dans plusieurs entrées de mappe pour garantir que les verrous U sont acquis dans le même ordre par chaque transaction. Par exemple, supposons que la première transaction appelle la méthode getForUpdate pour la clé 1 et la méthode getForUpdate pour la clé 2. Une autre transaction simultanée appelle la méthode getForUpdate pour les mêmes clés mais dans l'ordre inverse. Cette séquence entraîne l'interblocage classique car plusieurs verrous sont obtenus dans des ordres différents par diverses transactions. L'application doit toujours vérifier

que chaque transaction accède à plusieurs entrées de mappe dans la séquence de clés pour éviter tout interblocage. Etant donné que le verrou U est obtenu simultanément à l'appel de la méthode getForUpdate et non au moment de la validation, eXtreme Scale ne peut pas ordonner les demandes de verrouillage de la même manière que lors du cycle de validation. L'application doit contrôler l'ordre de verrouillage dans ce cas.

L'utilisation de la méthode flush dans l'interface ObjectMap avant une validation peut impliquer d'autres considérations en matière de définition de l'ordre de verrouillage. La méthode flush est généralement utilisée pour forcer les modifications apportées à la mappe et les refléter dans le programme d'arrière plan via le plug-in Loader. Dans ce cas, le programme d'arrière plan utilise son gestionnaire de verrou pour contrôler les accès simultanés de sorte que la condition d'attente du verrou et l'interblocage peuvent se produire dans le programme d'arrière plan et non dans le gestionnaire de verrou eXtreme Scale. Examinons la transaction suivante :

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
boolean activeTran = false;
try
{
    sess.begin();
    activeTran = true;
    Person p = (IPerson)person.get("Lynn");
    p.setAge( p.getAge() + 1 );
    person.put( "Lynn", p );
    person.flush();
    ...
    p = (IPerson)person.get("Tom");
    p.setAge( p.getAge() + 1 );
    sess.commit();
    activeTran = false;
}
finally
{
    if ( activeTran ) sess.rollback();
}
```

Supposons qu'une autre transaction a également mis à jour la personne Tom, a appelé la méthode flush, puis a mis à jour la personne Lynn. Si cette situation se présente, l'entrelacement ci-dessous des deux transactions entraîne une condition d'interblocage de base de données :

```
Verrou X octroyé à la transaction 1 pour "Lynn" lorsque la méthode flush est

exécutée.
Verrou X octroyé à la transaction 2 pour "Tom" lorsque la méthode flush est
exécutée.
Verrou X demandé par la transaction 1 pour "Tom" pendant le traitement commit.
(La transaction 1 se bloque en attente du verrou acquis par la transaction 2.)
Verrou X demandé par la transaction 2 pour "Lynn" pendant le traitement commit.
(La transaction 2 se bloque en attente du
verrou acquis par la transaction 1.)
```

Cet exemple montre que l'utilisation de la méthode flush peut provoquer un interblocage dans la base de données plutôt que dans eXtreme Scale. Cet exemple d'interblocage peut se produire indépendamment de la stratégie de verrouillage utilisée. L'application doit veiller à empêcher ce type d'interblocage lors de l'utilisation de la méthode flush et lorsqu'un chargeur est connecté à la mappe de sauvegarde. L'exemple précédent illustre également pourquoieXtreme Scale comporte un mécanisme de délai d'attente de verrou. Une transaction qui attend

un verrou de base de données peut patienter alors qu'elle possède un verrou d'entrée de mappe eXtreme Scale. Par conséquent, des problèmes rencontrés au niveau de la base de données peuvent causer des temps d'attente excessifs pour un mode de verrou eXtreme Scale et entraîner une exception LockTimeoutException.

Scénarios d'interblocage courants

Les sections ci-dessous décrivent certains scénarios d'interblocage courants et des suggestions permettant de les éviter.

Scénario : interblocage à clé unique

Les scénarios suivants décrivent comment des interblocages peuvent se produire lorsque l'accès à une clé unique est octroyé à l'aide d'un verrou S mis à jour ultérieurement. Lorsque cette situation se produit sur deux transactions simultanément, un interblocage a lieu.

Tableau 5. Scénario d'interblocage à clé unique

	Unité d'exécution 1	**Unité d'exécution 2**	
1	session.begin()	session.begin()	Chaque unité d'exécution effectue une transaction indépendante.
2	map.get(key1)	map.get(key1)	Verrou S octroyé pour les deux transactions pour key1
3	map.update(Key1,v)		Aucun verrou U. Mise à jour effectuée dans le cache transactionnel.
4		map.update(key1,v)	Aucun verrou U. Mise à jour effectuée dans le cache transactionnel
5	session.commit()		Bloquée : le verrou S pour key1 ne peut pas être mis à niveau vers un verrou X car l'unité d'exécution 2 comporte un verrou S.
6		session.commit()	Interblocage : le verrou S pour key1 ne peut pas être mis à niveau vers un verrou X car l'unité d'exécution 1 comporte un verrou S.

Tableau 6. Interblocages à clé unique (suite)

	Unité d'exécution 1	**Unité d'exécution 2**	
1	session.begin()	session.begin()	Chaque unité d'exécution effectue une transaction indépendante.
2	map.get(key1)		Verrou S octroyé pour key1
3	map.getForUpdate(key1,v)		Verrou S mis à niveau vers un verrou U pour key1.
4		map.get(key1)	Verrou S octroyé pour key1
5		map.getForUpdate(key1,v)	Bloquée : T1 comporte déjà un verrou U.
6	session.commit()		Interblocage : le verrou U pour key1 ne peut pas être mis à niveau.

Tableau 6. Interblocages à clé unique (suite) (suite)

	Unité d'exécution 1	**Unité d'exécution 2**	
7		session.commit()	Interblocage : le verrou S pour key1 ne peut pas être mis à niveau.

Tableau 7. Interblocages à clé unique (suite)

	Unité d'exécution 1	**Unité d'exécution 2**	
1	session.begin()	session.begin()	Chaque unité d'exécution effectue une transaction indépendante.
2	map.get(key1)		Verrou S octroyé pour key1
3	map.getForUpdate(key1,v)		Verrou S mis à niveau vers un verrou U pour key1.
4		map.get(key1)	Verrou S octroyé pour key1
5		map.getForUpdate(key1,v)	Bloquée : unité d'exécution 1 comporte déjà un verrou U.
6	session.commit()		Interblocage : le verrou U pour key1 ne peut pas être mis à niveau vers un verrou X car l'unité d'exécution 2 comporte un verrou S.

Si ObjectMap.getForUpdate est utilisé pour éviter le verrou S, l'interblocage est évité :

Tableau 8. Interblocages à clé unique (suite)

	Unité d'exécution 1	**Unité d'exécution 2**	
1	session.begin()	session.begin()	Chaque unité d'exécution effectue une transaction indépendante.
2	map.getForUpdate(key1)		Verrou U octroyé à l'unité d'exécution 1 pour key1.
3		map.getForUpdate(key1)	Demande de verrou U bloquée.
4	map.update(key1,v)	<bloquée>	
5	session.commit()	<bloquée>	Le verrou U pour key1 peut être mis à niveau vers un verrou X.
6		<libérée>	Le verrou U est finalement octroyé à key1 pour l'unité d'exécution 2.
7		map.update(key2,v)	Verrou U octroyé à l'unité d'exécution 2 pour key2.
8		session.commit()	Le verrou U pour key1 peut être mis à niveau vers un verrou X.

Solutions

1. Utilisez la méthode getForUpdate et non la méthode get pour obtenir un verrou U et non un verrou S.
2. Utilisez un niveau d'isolement de transaction lecture validée pour éviter de maintenir des verrous S. La réduction du niveau d'isolement de transaction

augmente la possibilité de lectures non reproductibles. Toutefois, les lectures non reproductibles sont uniquement possibles si le cache de transaction est explicitement invalidé.

3. Utilisez la stratégie de verrouillage optimiste. L'utilisation de la stratégie de verrouillage optimiste requiert le traitement des exceptions de conflits optimistes.

Scénario : interblocage à clés multiples

Ce scénario décrit ce qu'il se passe si deux transactions tentent de mettre à jour la même entrée et maintiennent des verrous S sur les autres entrées.

Tableau 9. Cas d'interblocage à clés multiples (suite)

	Unité d'exécution 1	**Unité d'exécution 2**	
1	session.begin()	session.begin()	Chaque unité d'exécution effectue une transaction indépendante.
2	map.get(key1)	map.get(key1)	Verrou S octroyé pour les deux transactions pour key1
3	map.get(key2)	map.get(key2)	Verrou S octroyé pour les deux transactions pour key2
4	map.update(key1,v)		Aucun verrou U. Mise à jour effectuée dans le cache transactionnel.
5		map.update(key2,v)	Aucun verrou U. Mise à jour effectuée dans le cache transactionnel.
6.	session.commit()		Bloquée : le verrou S pour key1 ne peut pas être mis à niveau vers un verrou X car l'unité d'exécution 2 comporte un verrou S.
7		session.commit()	Bloquée : le verrou S pour key2 ne peut pas être mis à niveau car l'unité d'exécution 1 comporte un verrou S.

La méthode ObjectMap.getForUpdate permet d'éviter le verrou S, ce qui réduit l'interblocage :

Tableau 10. Cas d'interblocage à clés multiples (suite)

	Unité d'exécution 1	**Unité d'exécution 2**	
1	session.begin()	session.begin()	Chaque unité d'exécution effectue une transaction indépendante.
2	map.getForUpdate(key1)		Verrou U octroyé à la transaction T1 pour key1.
3		map.getForUpdate(key1)	Demande de verrou U bloquée.
4	map.get(key2)	<bloquée>	Verrou S octroyé pour T1 pour key2
5	map.update(key1,v)	<bloquée>	
6	session.commit()	<bloquée>	Le verrou U pour key1 peut être mis à niveau vers un verrou X.
7		<libérée>	Le verrou U est finalement octroyé à key1 pour l'unité d'exécution 2.

Tableau 10. Cas d'interblocage à clés multiples (suite) (suite)

	Unité d'exécution 1	Unité d'exécution 2	
8		map.get(key2)	Verrou S octroyé à T2 pour key2
9		map.update(key2,v)	Verrou U octroyé à T2 pour key2
10		session.commit()	Le verrou U pour key1 peut être mis à niveau vers un verrou X.

Solutions

1. Utilisez getForUpdate et non la méthode get pour acquérir un verrou U directement pour la première clé. Cette stratégie n'est possible que si l'ordre de la méthode est déterministe.
2. Utilisez un niveau d'isolement de transaction lecture validée pour éviter de maintenir des verrous S. Il s'agit de la solution d'implémentation la plus simple si l'ordre de la méthode n'est pas déterministe. La réduction du niveau d'isolement de transaction augmente la possibilité de lectures non reproductibles. Toutefois, les lectures non reproductibles sont uniquement possibles si le cache de transaction est explicitement invalidé.
3. Utilisez la stratégie de verrouillage optimiste. L'utilisation de la stratégie de verrouillage optimiste requiert le traitement des exceptions de conflits optimistes.

Scénario : erreur d'ordre avec le verrou U

Si l'ordre dans lequel les clés sont demandées ne peut pas être garantie, un interblocage peut avoir lieu :

Tableau 11. Scénario d'erreur d'ordre avec le verrou U

	Unité d'exécution 1	Unité d'exécution 2	
1	session.begin()	session.begin()	Chaque unité d'exécution effectue une transaction indépendante.
2	map.getforUpdate(key1)	map.getForUpdate(key2)	Verrou U octroyé pour key1 et key2
3	map.get(key2)	map.get(key1)	Verrou S octroyé pour key1 et key2
4	map.update(key1,v)	map.update(key2,v)	
5	session.commit()		Le verrou U ne peut pas être mis à niveau vers un verrou X car l'unité d'exécution 2 comporte un verrou S.
6		session.commit()	Le verrou U ne peut pas être mis à niveau vers un verrou X car l'unité d'exécution 1 comporte un verrou S.

Solutions

1. Recherchez en boucle tous les travaux avec un verrou U simple global (mutex). Cette méthode réduit l'accès simultané mais gère tous les scénarios lorsque l'accès et l'ordre ne sont pas déterministes.
2. Utilisez un niveau d'isolement de transaction lecture validée pour éviter de maintenir des verrous S. Il s'agit de la solution d'implémentation la plus simple si l'ordre de la méthode n'est pas déterministe et offre un grand nombre d'accès

simultanés. La réduction du niveau d'isolement de transaction augmente la possibilité de lectures non reproductibles. Toutefois, les lectures non reproductibles sont uniquement possibles si le cache de transaction est explicitement invalidé.

3. Utilisez la stratégie de verrouillage optimiste. L'utilisation de la stratégie de verrouillage optimiste requiert le traitement des exceptions de conflits optimistes.

Traitement des exceptions dans les scénarios de verrouillage

Les exemple précédents ne présentent pas de traitement des exceptions. Pour éviter que les verrous soient maintenus pendant une durée trop importante lorsqu'une exception LockTimeoutException ou une exception LockDeadlockException se produit, une application doit faire en sorte qu'elle intercepte des exceptions inattendues et qu'elle appelle la méthode rollback lorsqu'un événement imprévu survient. Modifiez le fragment de code précédent tel qu'illustré dans l'exemple suivant :

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
boolean activeTran = false;
try
{
    sess.begin();
    activeTran = true;
    Person p = (IPerson)person.get("Lynn");
    // Lynn a fêté son anniversaire, donc nous l'avons vieilli d'1 an.
    p.setAge( p.getAge() + 1 );
    person.put( "Lynn", p );
    sess.commit();
    activeTran = false;
}
finally
{
    if ( activeTran ) sess.rollback();
}
```

Le dernier bloc du fragment de code assure l'annulation d'une transaction lorsqu'une exception inattendue se produit. Il ne gère pas seulement une exception de type LockDeadlockException mais également les autres exceptions imprévues éventuelles. Le dernier bloc traite le cas où une exception est émise lors d'un appel de méthode commit. Cet exemple ne constitue pas le seul moyen pour traiter les exceptions inattendues et il existe des cas où une application tente d'intercepter certaines des exceptions inattendues susceptibles de se produire et d'afficher une de ses exceptions d'application. Vous pouvez ajouter des blocs catch comme il convient mais l'application doit faire en sorte que le fragment de code ne se ferme pas sans terminer la transaction.

Stratégies de verrouillage

Les stratégies de verrouillage sont de type pessimiste, optimiste ou aucune. Pour choisir une stratégie de verrouillage, vous devez prendre en compte les aspects tels que le pourcentage des types d'opérations, l'utilisation ou non d'un chargeur, etc.

Les verrous sont liés aux transactions. Vous pouvez spécifier les paramètres de verrouillage suivants :

- **Aucun verrouillage** : l'exécution sans verrouillage est la plus rapide. Si vous utilisez des données en lecture seule, vous n'avez peut-être pas besoin de verrouillage.

- **Verrouillage pessimiste** : place des verrous sur les entrées, puis les maintient jusqu'au moment de la validation. Cette stratégie offre une bonne cohérence au prix de la capacité de traitement.
- **Verrouillage optimiste** : prend une image précédente de chaque enregistrement sur lequel la transaction appuie et compare l'image avec les valeurs d'entrées en cours quand la transaction est validée. Si les valeurs d'entrées changent, la transaction est annulée. Aucun verrou n'est maintenu jusqu'au moment de la validation. Cette stratégie de verrouillage offre un meilleur accès simultané que les stratégies pessimistes, au risque que la transaction soit annulée et au prix de la mémoire nécessaire pour une copie supplémentaire de l'entrée.

Définissez la stratégie de verrouillage sur la mappe de sauvegarde. Il n'est pas possible de changer de stratégie pour chaque transaction. Un fragment de code XML suit, montrant comment définir le mode de verrouillage sur une mappe à l'aide du fichier XML, en partant du principe que `cc` est le nom d'espace pour `objectgrid/config` :

```
<cc:backingMap name="RuntimeLifespan" lockStrategy="PESSIMISTIC" />
```

Verrouillage pessimiste

La stratégie de verrouillage pessimiste est à utiliser pour les opérations de mappe en lecture et en écriture lorsqu'aucune autre stratégie de verrouillage n'est possible. Lorsqu'une mappe ObjectGrid est configurée en mode de stratégie de verrouillage pessimiste, un verrou de transaction pessimiste est obtenu pour une entrée de mappe à la première transmission de l'entrée à la mappe de sauvegarde. Le verrou pessimiste est maintenu jusqu'à la fin de la transaction. La stratégie de verrouillage pessimiste est généralement utilisée dans les cas suivants :

- Lorsque la mappe de sauvegarde est configurée avec ou sans chargeur et que les informations sur les versions ne sont pas disponibles.
- Lorsque la mappe de sauvegarde est utilisée directement par une application qui nécessite l'assistance de eXtreme Scale pour le contrôle des accès simultanés.
- Lorsque les informations sur les versions sont disponibles mais que les transactions de mise à jour entrent régulièrement en conflit avec les entrées de sauvegarde, ce qui entraîne des échecs de mise à jour optimiste.

Etant donné que la stratégie de verrouillage pessimiste a un impact majeur sur les performances et l'évolutivité, elle doit uniquement être utilisée pour les mappes en lecture et écriture lorsque les autres stratégies de verrouillage ne sont pas adaptées. Par exemple, ces situations peuvent être : échecs réguliers des mises à jour optimistes ou reprise après échec optimiste difficile à gérer pour une application.

Verrouillage optimiste

La stratégie de verrouillage optimiste présuppose qu'il ne peut se faire que deux transactions tentent d'actualiser la même entrée de mappe au même moment. A partir de ce principe, il n'est pas nécessaire de maintenir le mode de verrouillage pendant tout le cycle de vie de la transaction car il est peu probable que plusieurs transactions puissent actualiser la même entrée de mappe exactement au même moment. La stratégie de verrouillage optimiste est généralement utilisée dans les situations suivantes :

- Lorsqu'une mappe de sauvegarde est configurée avec ou sans chargeur et que les informations sur les versions sont disponibles.
- Lorsqu'une mappe de sauvegarde contient essentiellement des transactions qui exécutent des opérations de lecture. Les opérations insert, update ou remove sur les entrées de mappe ne sont pas fréquentes pour une mappe de sauvegarde.

- Lorsqu'une mappe de sauvegarde est insérée, mise à jour ou supprimée plus fréquemment qu'elle n'est lue mais que les transactions entrent rarement en conflit sur la même entrée de mappe.

A l'instar de la stratégie de verrouillage pessimiste, les méthodes dans l'interface ObjectMap déterminent comment eXtreme Scale tente automatiquement d'obtenir un mode de verrouillage pour l'entrée de mappe à laquelle l'accès est octroyé. Toutefois, les stratégies pessimiste et optimiste diffèrent sur les points suivants :

- Comme la stratégie de verrouillage pessimiste, un mode de verrou S est obtenu par les méthodes get et getAll lorsque la méthode est appelée. En revanche, avec le verrouillage optimiste, le mode de verrou S n'est maintenu que lorsque la transaction s'achève. Le mode de verrou S est libéré avant que la méthode soit renvoyée à l'application. L'objectif d'un mode de verrou consiste en ce que eXtreme Scale vérifie que seules les données validées provenant d'autres transactions soient visibles pour la transaction en cours. Une fois que eXtreme Scale a confirmé la validation des données, le mode de verrou S est libéré. Au moment de la validation, une vérification optimiste des versions est effectuée pour faire en sorte qu'aucune autre transaction n'a modifié l'entrée de mappe après la libération du mode de verrou S par la transaction en cours. Si une entrée n'est pas extraite d'une mappe avant d'être mise à jour, invalidée ou supprimée, l'exécution eXtreme Scale extrait implicitement l'entrée de la mappe. Cette opération get implicite est effectuée pour obtenir la valeur actuelle au moment de la demande de modification de l'entrée.
- Contrairement à la stratégie de verrouillage pessimiste, les méthodes getForUpdate et getAllForUpdate sont traitées exactement comme les méthodes get et getAll de la stratégie de verrouillage optimiste. Un mode de verrou S est obtenu au début de la méthode et est libéré avant d'être renvoyé à l'application.

Toutes les autres méthodes ObjectMap sont traitées exactement comme elles le sont dans la stratégie de verrouillage pessimiste. Lorsque la méthode commit est appelée, un mode de verrou X est obtenu pour toutes les entrées de mappe insérées, mises à jour, supprimées, corrigées ou invalidées et le mode de verrou X est maintenu jusqu'à ce que la transaction effectue le processus de validation.

La stratégie de verrouillage optimiste considère qu'aucune transaction simultanée ne tente de mettre à jour la même entrée de mappe. A partir de ce principe, le mode de verrouillage ne doit pas être maintenu pour le cycle de vie de la transaction car il est peu probable qu'une transaction puisse mettre à jour simultanément la même entrée de mappe. Toutefois, étant donné qu'aucun mode de verrouillage n'est maintenu, une autre transaction simultanée peut potentiellement mettre à jour l'entrée de mappe après la libération du mode de verrou S par la transaction en cours.

Pour gérer cette possibilité, eXtreme Scale obtient un verrou X au moment de la validation et effectue une vérification optimiste des versions pour faire en sorte qu'aucune autre transaction n'a modifié l'entrée de mappe après la lecture de l'entrée de mappe de la mappe de sauvegarde par la transaction en cours. Si une autre transaction modifie l'entrée de mappe, la vérification de versions échoue et une exception OptimisticCollisionException se produit. Cette exception force l'annulation de la transaction en cours et l'application doit réessayer la transaction entière. La stratégie de verrouillage optimiste s'avère très utile lors qu'une mappe est principalement lue et que les mises à jour de la même entrée de mappe sont peu probables.

Aucun verrouillage

Lorsqu'une mappe de sauvegarde est configurée pour n'utiliser aucune stratégie de verrouillage, la valeur retournée est aucun verrou de transaction pour une entrée de mappe.

La non-utilisation d'une stratégie de verrouillage est utile lorsqu'une application est un gestionnaire de persistance tel qu'un conteneur EJB (Enterprise JavaBeans) ou qu'une application utilise Hibernate pour obtenir les données persistantes. Dans ce scénario, la mappe de sauvegarde est configurée sans chargeur et le gestionnaire de persistance utilise la mappe de sauvegarde comme cache de données. Dans ce scénario, le gestionnaire de persistance fournit le contrôle des accès simultanés entre les transactions qui accèdent aux mêmes entrées de mappe.

WebSphere eXtreme Scale n'a pas besoin d'obtenir de verrous de transaction à des fins de contrôle des accès simultanés. Cette situation considère que le gestionnaire de persistance ne libère pas ses verrous de transaction avant de mettre à jour la mappe ObjectGrid avec les modifications validées. Si le gestionnaire de persistance libère ses verrous, une stratégie de verrouillage pessimiste ou optimiste doit être utilisée. Par exemple, supposons que le gestionnaire de persistance d'un conteneur d'EJB met à jour une mappe ObjectGrid avec les données validées dans la transaction EJB gérée par conteneur. Si la mise à jour de la mappe ObjectGrid a lieu avant la libération des verrous du gestionnaire de persistance, vous pouvez utiliser la stratégie sans verrou. Si la mise à jour de la mappe ObjectGrid a lieu après la libération des verrous du gestionnaire de persistance, vous devez utiliser la stratégie optimiste ou pessimiste.

La stratégie sans verrou peut être utilisée également lorsque l'application utilise directement une mappe de sauvegarde et qu'un chargeur est configuré pour la mappe. Dans ce scénario, le chargeur utilise le fonction de contrôle des accès simultanés fournies par un système de gestion de base de données relationnelle (SGBDR) à l'aide de la connectivité JDBC (Java Database Connectivity) ou le plug-in Hibernate pour accéder aux données dans une base de données relationnelle. L'implémentation du chargeur peut utiliser une approche optimiste ou pessimiste. Un chargeur qui utilise une approche optimiste de verrouillage ou de gestion des versions contribue à optimiser les performances et l'accès simultané. Pour plus d'informations relatives à l'implémentation d'une approche de verrouillage optimiste, reportez-vous à la section OptimisticCallback de la rubrique les informations relatives aux remarques sur le chargeur dans le*Guide d'administration*. Si vous utilisez un chargeur qui utilise le verrouillage pessimiste d'un programme d'arrière plan, il est conseillé d'utiliser le paramètre forUpdate transmis à la méthode get de l'interface Loader. Définissez ce paramètre sur true si la méthode getForUpdate de l'interface ObjectMap a été utilisée par l'application pour obtenir les données. Le chargeur peut se servir de ce paramètre pour déterminer s'il convient de demander un verrou pouvant être mis à niveau sur la ligne en cours de lecture. Par exemple, DB2 obtient un verrou pouvant être mis à niveau lorsqu'une instruction SQL select contient une clause `FOR UPDATE`. Cette approche offre la même protection contre les interblocages que celle décrite dans la rubrique«Verrouillage pessimiste», à la page 139.

Meilleures pratiques pour les performances de verrouillage

Les stratégies de verrouillage et les paramètres d'isolement de transactions affectent les performances de vos applications.

Extraction d'une instance mise en cache

Pour plus d'informations sur le verrouillage des entrées de mappe, consultez le manuel *Guide d'administration*.

Stratégie de verrouillage pessimiste

Recourez à la stratégie de verrouillage pessimiste pour les opérations de mappe en lecture et en écriture pour lesquelles les clés entrent généralement en conflit. La stratégie de verrouillage pessimiste a une incidence considérable sur les performances.

Isolement de transactions lecture validée et lecture non validée

Lorsque vous utilisez la stratégie de verrouillage pessimiste, définissez le niveau d'isolement de transaction à l'aide de la méthode Session.setTransactionIsolation. Pour l'isolement de transactions lecture validée et lecture non validée, utilisez l'argument Session.TRANSACTION_READ_COMMITTED ou Session.TRANSACTION_READ_UNCOMMITTED en fonction de l'isolement. Pour rétablir le comportement de verrouillage pessimiste par défaut pour le niveau d'isolement de transaction, faites appel à la méthode Session.setTransactionIsolation avec l'argument Session.REPEATABLE_READ.

L'isolement lecture validée réduit la durée des verrous partagés, ce qui peut améliorer l'accès simultané et limiter le risque d'interblocage. Ce niveau d'isolement doit être utilisé lorsqu'une transaction ne doit pas garantir l'invariabilité des valeurs de lecture pendant la durée de la transaction.

Utilisez la valeur lecture non validée lorsque la transaction ne doit pas tenir compte les données validées.

Stratégie de verrouillage optimiste

Il s'agit de la configuration par défaut. Cette stratégie améliore les performances et l'évolutivité par rapport à la stratégie pessimiste. Utilisez cette stratégie lorsque vos applications peuvent tolérer certains échecs de la mise à jour optimiste tout en offrant cependant de meilleures performances que la stratégie pessimiste. Cette stratégie s'adapte particulièrement aux opérations de lecture et aux applications de mise à jour exceptionnelles.

Plug-in OptimisticCallback

La stratégie de verrouillage optimiste effectue une copie des entrées de cache et les compare si nécessaire. Cette opération peut être coûteuse car la copie des entrées risque d'entraîner des tâches de clonage ou de sérialisation. Pour augmenter les performances le plus rapidement possible, implémentez le plug-in personnalisé pour les mappes de non-entité.

Pour plus d'informations, voir. Pour plus d'informations sur le plug-in OptimisticCallback, consultez le manuel *Présentation du produit*.

Utilisation de champs version pour les entités

Lorsque vous utilisez le verrouillage optimiste avec les entités, recourez à l'annotation @Version ou l'attribut équivalent dans le fichier descripteur de métadonnées Entité . L'annotation de version permet à l'ObjectGrid de suivre de

façon efficace la version d'un objet. Si l'entité ne comporte pas de champ version et si le verrouillage optimiste est utilisé pour l'entité, l'entité entière doit être copiée et comparée.

Stratégie sans verrouillage

Utilisez la stratégie sans verrouillage pour les applications en lecture seule. Cette stratégie ne déclenche aucun verrouillage et n'utilise aucun gestionnaire de verrous. Par conséquent, elle offre plus d'accès simultanés, de performances et d'évolutivité.

Verrous d'entrée de mappe avec requêtes et index

Cette rubrique décrit comment les API de création de requêtes eXtreme Scale et le plug-in d'indexation MapRangeIndex interagissent avec les verrous et présente les meilleures pratiques pour augmenter les accès simultanés et réduire les interblocages lors de l'utilisation de la stratégie de verrouillage pessimiste pour les mappes.

Présentation

L'API de création de requêtes ObjectGrid active les requêtes SELECT sur les objets et les entités cache ObjectMap. Lorsqu'une requête est exécutée, le moteur de requête utilise un MapRangeIndex dans la mesure du possible pour trouver des clés qui correspondent aux valeurs de la clause WHERE de la requête ou pour établir des relations. Lorsqu'un index n'est pas disponible, le moteur de requête analyse chaque entrée dans une ou plusieurs mappes pour trouver les entrées appropriées. Le moteur de requête et les plug-in d'index obtiennent des verrous pour vérifier la cohérence des données en fonction de la stratégie de verrouillage, le niveau d'isolement de la transaction et l'état de la transaction.

Verrouillage à l'aide du plug-in HashIndex

Le plug-in eXtreme Scale HashIndex permet de trouver des clés en fonction d'un attribut unique stocké dans la valeur d'entrée du cache. L'index stocke la valeur indexée dans une structure de données distincte de la mappe du cache. L'index valide les clés avec les entrées de mappe avant d'être renvoyé à l'utilisateur afin de tenter d'obtenir un ensemble de résultats exact. Lorsque la stratégie de verrouillage pessimiste est utilisée et que l'index est utilisé avec une instance d'ObjectMap locale (contrairement à une instance d'ObjectMap client-serveur), l'index obtient des verrous pour chaque entrée correspondante. Lors de l'utilisation d'un verrouillage optimiste ou d'une instance ObjectMap distante, les verrous sont toujours immédiatement libérés.

Le type de verrou obtenu dépend de l'argument forUpdate transmis à la méthode ObjectMap.getIndex. L'argument forUpdate spécifie le type de verrou que l'index doit obtenir. Si la valeur est égale à false, un verrou (S) pouvant être partagé est obtenu et si la valeur est égale à true, un verrou (U) pouvant être mis à niveau est obtenu.

Si le type de verrou peut être partagé, le paramètre d'isolement de la transaction pour la session est appliqué et affecte la durée du verrouillage. Reportez-vous à la rubrique relative à l'isolement de transaction pour plus de détails sur l'utilisation de l'isolement de transaction pour ajouter des accès simultanés aux applications.

Verrous partagés avec requête

Le moteur de requête eXtreme Scale obtient des verrous S si nécessaire pour introspecter les entrées de cache et découvrir si elles répondent aux critères de filtrage de la requête. Lors de l'utilisation de l'isolement de transaction de lecture reproductible avec le verrouillage pessimiste, les verrous S sont uniquement conservés pour les éléments inclus dans le résultat de la requête et sont libérés pour toutes les entrées non incluses dans le résultat. Dans le cas de l'utilisation d'un niveau d'isolement de transaction inférieur ou du verrouillage optimiste, les verrous S ne sont pas conservés.

Verrous partagés avec requête client-serveur

Lors de l'utilisation de la requête eXtreme Scale depuis un client, la requête s'exécute généralement sur le serveur à moins que toutes les mappes ou entités référencées dans la requête soient stockées en local sur le client (par exemple : une mappe client repliquée ou une entité de résultat de requête). Toutes les requêtes exécutées dans une transaction en lecture/écriture conservent les verrous S, tel que décrit dans la section précédente. Si la transaction n'est pas en lecture/écriture, une session n'est pas conservée sur le serveur et les verrous S sont libérés.

Une transaction en lecture/écriture est uniquement acheminée vers une partition principale et une session est conservée sur le serveur pour la session client. Une transaction peut être promue en lecture/écriture sous les conditions suivantes :

1. L'accès à toute mappe configurée pour utiliser le verrouillage pessimiste est rendu possible par les méthodes de l'API ObjectMap get et getAll ou les méthodes EntityManager.find.
2. La transaction est vidée, ce qui provoque l'envoi de mises à jour au serveur.
3. L'accès à toute mappe configurée pour utiliser le verrouillage optimiste est rendu possible par la méthode ObjectMap.getForUpdate ou EntityManager.findForUpdate.

Verrous pouvant être mis à niveau avec requête

Les verrous partageables sont utiles lorsque l'accès simultané et la cohérence sont prioritaires. Ils garantissent l'inaltérabilité de la valeur d'une entrée pendant toute la durée de vie de la transaction. Aucune autre transaction ne peut modifier la valeur alors que d'autres verrous S sont maintenus et seule une autre transaction peut établir un objectif de mise à jour de l'entrée. Consultez la rubrique Mode de verrouillage pessimiste pour obtenir des détails sur les modes de verrouillage S, U et X.

Les verrous pouvant être mis à niveau permettent d'identifier l'objectif de mise à jour d'une entrée de cache lors de l'utilisation d'une stratégie de verrouillage pessimiste. Ils permettent la synchronisation entre les transactions qui visent à modifier une entrée de cache. Les transactions peuvent toujours visualiser l'entrée à l'aide d'un verrou S mais certaines ne peuvent pas obtenir un verrou U ou X. Dans de nombreux scénarios, il est nécessaire d'obtenir un verrou U sans acquérir un verrou S au préalable pour éviter tout interblocage. Consultez la rubrique Mode de verrouillage pessimiste pour obtenir des exemples d'interblocage.

Les interfaces de requête ObjectQuery et EntityManager offrent la méthode setForUpdate pour identifier l'utilisation voulue pour le résultat de requête. Le

moteur de requête obtient notamment les verrous U et non les verrous S pour chaque entrée de mappe impliquée dans le résultat de requête :

```
ObjectMap orderMap = session.getMap("Order");
ObjectQuery q = session.createQuery("SELECT o FROM Order o WHERE o.orderDate=?1");
q.setParameter(1, "20080101");
q.setForUpdate(true);
session.begin();
// Exécutez la requête.  Chaque commande comporte  verrou U
Iterator result = q.getResultIterator();
// Pour chaque commande, mettez à jour l'état.
while(result.hasNext()) {
    Order o = (Order) result.next();
    o.status = "shipped";
    orderMap.update(o.getId(), o);
}
// Une fois validée, la
session.commit();
```

```
Query q = em.createQuery("SELECT o FROM Order o WHERE o.orderDate=?1");
q.setParameter(1, "20080101");
q.setForUpdate(true);
emTran.begin();
// Exécutez la requête.  Chaque commande comporte  verrou U
Iterator result = q.getResultIterator();
// Pour chaque commande, mettez à jour l'état.
while(result.hasNext()) {
    Order o = (Order) result.next();
    o.status = "shipped";
}
tmTran.commit();
```

Lorsque l'attribut **setForUpdate** est activé, la transaction est automatiquement convertie en une transaction lecture-écriture et les verrous sont maintenus sur le serveur comme prévu. Si la requête ne peut pas utiliser d'index, la mappe doit être analysée, ce qui entraîne des verrous U temporaires pour les entrées de mappe qui ne répondent pas au résultat de la requête et maintient les verrous U pour les entrées qui correspondent au résultat de la requête.

Isolement de transactions

Pour les transactions, vous pouvez configurer chaque configuration de mappe de sauvegarde à l'aide de l'une des trois stratégies de verrouillage : pessimiste, optimiste ou aucune. Lorsque vous utilisez les verrouillages pessimiste et optimiste, eXtreme Scale utilise des verrous partagés (S), pouvant être mise à niveau (U) et exclusifs (X) pour maintenir la cohérence. Ce comportement de verrouillage est plus marqué lors de l'utilisation du verrouillage pessimiste car les verrous optimistes ne sont pas maintenus. Vous pouvez utiliser l'un des trois niveaux d'isolement de transaction afin de paramétrer la sémantique de verrouillage utilisée par eXtreme Scale pour maintenir la cohérence dans chaque mappe de cache : lecture reproductible, lecture validée et lecture non validée.

Présentation de l'isolement de transaction

L'isolement de transaction définir comment les modifications apportées par une opération deviennent visibles aux autres opérations simultanées.

WebSphere eXtreme Scale prend en charge trois niveaux d'isolement de transaction qui peuvent vous permettre de peaufiner le paramétrage de la sémantique de verrouillage utilisée par l'eXtreme Scale pour maintenir la cohérence dans chaque mappe de cache : lecture reproductible, lecture validée et lecture non validée. Le niveau d'isolement de transaction est défini sur l'interface Session à l'aide de la

méthodesetTransactionIsolation. L'isolement de transaction peut être modifié à tout moment pendant la durée de vie de la session si une transaction n'est pas en cours d'exécution.

Le produit impose les divers aspects de la sémantique d'isolement de transaction en paramétrant la demande et le maintien des verrous (S) partagés. L'isolement de transaction n'a aucune incidence sur les mappes configurées pour utiliser les stratégies de verrouillage optimiste ou aucune ou lorsque des verrous (U) pouvant être mis à jour sont obtenus.

Lecture reproductible avec verrouillage pessimiste

Le niveau d'isolement de transaction lecture reproductible est le niveau par défaut. Ce niveau d'isolement empêche les lectures de pages modifiées et les lectures non reproductibles mais non les lectures fantômes. Une lecture de pages modifiées est une opération de lecture de données qui ont été modifiées par une transaction mais n'ont pas été validées. Une lecture non reproductible peut survenir lorsqu'aucun verrou en lecture n'a été obtenu lors de l'opération de lecture. Une lecture fantôme a lieu lorsque deux opérations de lecture identiques ont été effectuées mais que deux jeux de résultats ont été renvoyés en raison d'une mise à jour entre les opérations de lecture. Le produit obtient une lecture reproductible en maintenant les verrous S jusqu'à la fin de la transaction qui possède le verrou concerné. Etant donné qu'un verrou X n'est octroyé que lorsque tous les S sont libérés, toutes les transactions maintenant le verrou S obtiendront obligatoirement la même valeur lors de la relecture.

```
map = session.getMap("Order");
session.setTransactionIsolation(Session.TRANSACTION_REPEATABLE_READ);
session.begin();

// Un verrou S est demandé et maintenu et la valeur est copiée dans
// le cache transactionnel.
Order order = (Order) map.get("100");
// L'entrée est expulsée du cache transactionnel.
map.invalidate("100", false);

// La même valeur est demandée de nouveau.  Elle contient déjà le
// verrou, donc la même valeur est extraite et copiée dans le
// cache transactionnel.
Order order2 (Order) = map.get("100");

// Tous les verrous sont libérés après la synchronisation de la transaction
// avec la mappe de cache.
session.commit();
```

Les lectures fantômes sont possibles lorsque vous utilisez des requêtes ou des index car les verrous ne sont pas obtenus pour des plages de données, uniquement pour les entrées de cache qui correspondent à l'index ou aux critères de requête. Par exemple :

```
session1.setTransactionIsolation(Session.TRANSACTION_REPEATABLE_READ);
session1.begin();

// Une requête qui sélectionne une plage de valeurs est exécutée.
ObjectQuery query = session1.createObjectQuery
    ("SELECT o FROM Order o WHERE o.itemName='Widget'");

// Dans ce cas, une seule commande correspond au filtre de requête.
// La commande a une clé de "100".
// Le moteur de requête obtient automatiquement un verrou S pour la commande "100".
Iterator result = query.getResultIterator();
```

```
// Une deuxième transaction insère une commande qui correspond également
// à la requête.
Map orderMap = session2.getMap("Order");
orderMap.insert("101", new Order("101", "Widget"));

// Lorsque la requête est réexécutée dans la transaction en cours, la
// nouvelle commande est visible et renvoie les commandes "100" et "101".
result = query.getResultIterator();

// Tous les verrous sont libérés après la synchronisation de la transaction
// avec la mappe de cache.
session.commit();
```

Lecture validée avec verrouillage pessimiste

Le niveau d'isolement de transaction de lecture validée peut être utilisé avec eXtreme Scale qui empêche les lectures de pages modifiées mais non les lectures non reproductibles ou les lectures fantômes ; eXtreme Scale continue ainsi à utiliser les verrous S pour lire les données de la mappe de cache mais libère immédiatement les verrous.

```
map1 = session1.getMap("Order");
session1.setTransactionIsolation(Session.TRANSACTION_READ_COMMITTED);
session1.begin();

// Un verrou S est demandé mais immédiatement libéré et
//la valeur est copiée dans le cache transactionnel.

Order order = (Order) map1.get("100");

// L'entrée est expulsée du cache transactionnel.
map1.invalidate("100", false);

// Une deuxième transaction met à jour la même commande.
// Elle obtient un verrou U, met à jour la valeur et valide.
// L'ObjectGrid obtient correctement le verrou X lors de la
// validation car la première transaction utilise l'isolement lecture
// validée.

Map orderMap2 = session2.getMap("Order");
session2.begin();
order2 = (Order) orderMap2.getForUpdate("100");
order2.quantity=2;
orderMap2.update("100", order2);
session2.commit();

// La même valeur est demandée de nouveau.  Cette fois, la valeur doit être
// mise à jour mais elle reflète maintenant la
// nouvelle valeur
Order order1Copy (Order) = map1.getForUpdate("100");
```

Lecture non validée avec verrouillage pessimiste

Le niveau d'isolement de transaction lecture non validée peut être utilisé avec eXtreme Scale qui autorise les lectures de pages modifiées, les lectures non reproductibles et les lectures fantômes.

Exception OptimisticCollisionException

Vous pouvez recevoir une exception OptimisticCollisionException soit directement, soit avec une exception ObjectGridException.

Le code suivant montre comment intercepter l'exception et afficher son message :

```
try {
...
} catch (ObjectGridException oe) {
    System.out.println(oe);
}
```

Cause de l'exception

L'exception OptimisticCollisionException est créée lorsque deux clients différents essaient d'actualiser à peu près en même temps la même entrée de mappe. Supposons par exemple qu'un client tente de valider une session et d'actualiser l'entrée de mappe. Au même moment ou presque, juste avant cette validation, un autre client vient de lire ces données. Les données qu'aura lues le second client seront donc erronées. L'exception est créée lorsque le l'autre client tente de valider les données erronées.

Récupérer la clé qui a déclenché l'exception

Il peut être utile, lorsqu'on cherche à résoudre ce genre d'exception, de récupérer la clé correspondant à l'entrée qui a déclenché l'exception. L'avantage de l'exception OptimisticCollisionException est justement qu'elle contient la méthode getKey, laquelle retourne l'objet représentant cette clé. L'exemple suivant montre comment récupérer et imprimer la clé lors de l'interception de cette exception :

```
try {
...
} catch (OptimisticCollisionException oce) {
    System.out.println(oce.getKey());
}
```

L'exception ObjectGridException provoque une exception OptimisticCollisionException

L'exception OptimisticCollisionException peut être la raison pour laquelle s'affiche l'exception ObjectGridException. Si c'est le cas, le code suivant vous aidera à déterminer le type de l'exception et à imprimer la clé. Il utilise la méthode findRootCause décrite dans la section ci-dessous.

```
try {
...
}
catch (ObjectGridException oe) {
    Throwable Root = findRootCause( oe );
    if (Root instanceof OptimisticCollisionException) {
        OptimisticCollisionException oce = (OptimisticCollisionException)Root;
        System.out.println(oce.getKey());
    }
}
```

Technique générale de gestion des exceptions

Connaître la cause fondamentale d'un objet Throwable est utile pour isoler la source d'une erreur. L'exemple suivant monte comment un gestionnaire d'exceptions emploie une méthode utilitaire pour déterminer la cause fondamentale de l'objet Throwable.

Exemple :

```
static public Throwable findRootCause( Throwable t )
{
     // L'on démarre à partir du Throwable qui s'est produit
     Throwable root = t;
```

```
        // L'on descend la chaîne des causes jusqu'à ce que l'on trouve
        // le dernier Throwable de la chaîne
        Throwable cause = root.getCause();
        while ( cause != null )
        {
           root = cause;
           cause = root.getCause();
        }

        // L'on retourne comme cause fondamentale le dernier Throwable de la chaîne.
        return root;
    }
```

Configuration de clients avec WebSphere eXtreme Scale

Vous pouvez configurer un client eXtreme Scale en fonction de vos besoins, par exemple modifier certains paramètres.

Configuration du client avec XML

Vous pouvez utiliser un fichier XML ObjectGrid pour modifier les paramètres côté client. Pour modifier les paramètres d'un client eXtreme Scale , vous devez créer un fichier XML ObjectGrid ayant une structure semblable à celle du fichier utilisé pour le serveur eXtreme Scale.

Nous supposerons que le fichier XML suivant a été couplé à un fichier XML de règle de déploiement et que ces fichiers ont été utilisés pour démarrer un serveur eXtreme Scale.

companyGridServerSide.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
    xmlns="http://ibm.com/ws/objectgrid/config">

    <objectGrids>
        <objectGrid name="CompanyGrid">
            <bean id="TransactionCallback"
                className="com.company.MyTxCallback" />
            <bean id="ObjectGridEventListener"
                className="com.company.MyOgEventListener" />
            <backingMap name="Customer"
                pluginCollectionRef="customerPlugins" />
            <backingMap name="Item" />
            <backingMap name="OrderLine" numberOfBuckets="1049"
                timeToLive="1600" ttlEvictorType="LAST_ACCESS_TIME" />
            <backingMap name="Order" lockStrategy="PESSIMISTIC"
                pluginCollectionRef="orderPlugins" />
        </objectGrid>
    </objectGrids>

    <backingMapPluginCollections>
        <backingMapPluginCollection id="customerPlugins">
            <bean id="Evictor"
                className="com.ibm.websphere.objectGrid.plugins.builtins.LRUEvictor" />
            <bean id="MapEventListener"
                className="com.company.MyMapEventListener" />
        </backingMapPluginCollection>
        <backingMapPluginCollection id="orderPlugins">
            <bean id="MapIndexPlugin"
                className="com.company.MyMapIndexPlugin" />
        </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

Sur un serveur eXtreme Scale, l'instance ObjectGrid nommée CompanyGrid se comporte conformément au comportement défini par le fichier

companyGridServerSide.xml. Par défaut, les paramètres du client CompanyGrid sont identiques à ceux de l'instance CompanyGrid qui s'exécute sur le serveur. Certains paramètres du client peuvent cependant être remplacés, comme suit :

1. Créez une instance ObjectGrid propre au client.
2. Copiez le fichier XML ObjectGrid utilisé pour ouvrir le serveur.
3. Editez le nouveau fichier pour personnaliser le côté client.
   - Pour définir ou mettre à jour les attributs du client, indiquez une nouvelle valeur ou modifiez la valeur existante.
   - Pour supprimer un plug-in du client, utilisez la chaîne vide comme valeur pour l'attribut className.
   - Pour modifier un plug-in existant, indiquez une nouvelle valeur pour l'attribut className.
   - Vous pouvez aussi ajouter les plug-in pris en charge pour les remplacements par les clients : TRANSACTION_CALLBACK, OBJECTGRID_EVENT_LISTENER, EVICTOR et MAP_EVENT_LISTENER.
4. Créez un client à l'aide du nouveau fichier XML de remplacement par les clients.

Le fichier XML ObjectGrid suivant peut être utilisé pour définir certains attributs et plug-in du client CompanyGrid.

companyGridClientSide.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
    xmlns="http://ibm.com/ws/objectgrid/config">

    <objectGrids>
        <objectGrid name="CompanyGrid">
            <bean id="TransactionCallback"
                className="com.company.MyClientTxCallback" />
            <bean id="ObjectGridEventListener" className="" />
            <backingMap name="Customer" numberOfBuckets="1429"
                pluginCollectionRef="customerPlugins" />
            <backingMap name="Item" />
            <backingMap name="OrderLine" numberOfBuckets="701"
                timeToLive="800" ttlEvictorType="LAST_ACCESS_TIME" />
            <backingMap name="Order" lockStrategy="PESSIMISTIC"
                pluginCollectionRef="orderPlugins" />
        </objectGrid>
    </objectGrids>

    <backingMapPluginCollections>
        <backingMapPluginCollection id="customerPlugins">
            <bean id="Evictor"
                className="com.ibm.websphere.objectGrid.plugins.builtins.LRUEvictor" />
            <bean id="MapEventListener" className="" />
        </backingMapPluginCollection>
        <backingMapPluginCollection id="orderPlugins">
            <bean id="MapIndexPlugin"
                className="com.company.MyMapIndexPlugin" />
        </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

- Le TransactionCallback du client est com.company.MyClientTxCallback et non le paramètre com.company.MyTxCallback du côté serveur.
- Le client n'est associé à aucun plug-in ObjectGridEventListener car la valeur className est la chaîne vide.
- Le client associe numberOfBuckets à la valeur 1429 pour le backingMap Customer, conserve le plug-in Evictor et supprime le plug-in MapEventListener.
- Les attributs numberOfBuckets et timeToLive du backingMap OrderLine ont été modifiés.

- Bien qu'un attribut lockStrategy différent ait été indiqué, les conséquences sont nulles car cet attribut n'est pas pris en charge pour un remplacement par le client.

Pour créer le client CompanyGrid à l'aide du fichier `companyGridClientSide.xml`, transmettez le fichier XML ObjectGrid en tant qu'URL à l'une des méthodes de connexion de l'ObjectGridManager.

Création du client pour XML
```
ObjectGridManager ogManager =
 ObjectGridManagerFactory.ObjectGridManager();
ClientClusterContext clientClusterContext =
 ogManager.connect("MyServer1.company.com:2809", null, new URL(
                "file:xml/companyGridClientSide.xml"));
```

Configuration du client à l'aide d'un programme

Vous pouvez aussi remplacer les paramètres ObjectGrid côté client à l'aide d'un programme. Créez un objet ObjectGridConfiguration dont la structure est semblable à celle de l'instance ObjectGrid côté serveur. Le code suivant crée une instance ObjectGrid côté client qui est fonctionnellement équivalente au remplacement par le client de la section précédente où un fichier XML était utilisé.

Remplacement côté client à l'aide d'un programme
```
ObjectGridConfiguration companyGridConfig = ObjectGridConfigFactory
    .createObjectGridConfiguration("CompanyGrid");
Plugin txCallbackPlugin = ObjectGridConfigFactory.createPlugin(
    PluginType.TRANSACTION_CALLBACK, "com.company.MyClientTxCallback");
companyGridConfig.addPlugin(txCallbackPlugin);

Plugin ogEventListenerPlugin = ObjectGridConfigFactory.createPlugin(
    PluginType.OBJECTGRID_EVENT_LISTENER, "");
companyGridConfig.addPlugin(ogEventListenerPlugin);

BackingMapConfiguration customerMapConfig = ObjectGridConfigFactory
    .createBackingMapConfiguration("Customer");
customerMapConfig.setNumberOfBuckets(1429);
Plugin evictorPlugin = ObjectGridConfigFactory.createPlugin(PluginType.EVICTOR,
    "com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor");
customerMapConfig.addPlugin(evictorPlugin);

companyGridConfig.addBackingMapConfiguration(customerMapConfig);

BackingMapConfiguration orderLineMapConfig = ObjectGridConfigFactory
    .createBackingMapConfiguration("OrderLine");
orderLineMapConfig.setNumberOfBuckets(701);
orderLineMapConfig.setTimeToLive(800);
orderLineMapConfig.setTtlEvictorType(TTLType.LAST_ACCESS_TIME);

companyGridConfig.addBackingMapConfiguration(orderLineMapConfig);

List ogConfigs = new ArrayList();
ogConfigs.add(companyGridConfig);

Map overrideMap = new HashMap();
overrideMap.put(CatalogServerProperties.DEFAULT_DOMAIN, ogConfigs);

ogManager.setOverrideObjectGridConfigurations(overrideMap);
ClientClusterContext client = ogManager.connect(catalogServerAddresses, null, null);
ObjectGrid companyGrid = ogManager.getObjectGrid(client, objectGridName);
```

L'instance ogManager de l'interface ObjectGridManager ne vérifie les remplacements que dans les objets ObjectGridConfiguration et BackingMapConfiguration inclus dans la mappe overrideMap. Par exemple, le code précédent remplace le nombre de compartiments de la mappe OrderLine. La mappe Order reste cependant inchangée côté client car aucune configuration de cette mappe n'est incluse.

Configuration du client dans Spring

Les paramètres ObjectGrid côté client peuvent également être remplacés à l'aide de Spring Framework. L'exemple de fichier XML suivant montre comment générer un élément ObjectGridConfiguration et l'utiliser pour remplacer certains paramètres côté client. Cet exemple appelle les mêmes API que celles montrées dans la configuration par programmation. Cet exemple est également fonctionnellement équivalent à l'exemple de la configuration XML ObjectGrid.

Configuration du client avec Spring

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
  "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
  <bean id="companyGrid" factory-bean="manager" factory-method="getObjectGrid"
    singleton="true">
    <constructor-arg type="com.ibm.websphere.objectgrid.ClientClusterContext">
      <ref bean="client" />
    </constructor-arg>
    <constructor-arg type="java.lang.String" value="CompanyGrid" />
</bean>

  <bean id="manager" class="com.ibm.websphere.objectgrid.ObjectGridManagerFactory"
    factory-method="getObjectGridManager" singleton="true">
    <property name="overrideObjectGridConfigurations">
      <map>
        <entry key="DefaultDomain">
          <list>
            <ref bean="ogConfig" />
          </list>
        </entry>
      </map>
    </property>
  </bean>

  <bean id="ogConfig"
    class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
    factory-method="createObjectGridConfiguration">
    <constructor-arg type="java.lang.String">
      <value>CompanyGrid</value>
    </constructor-arg>
    <property name="plugins">
      <list>
   <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
          factory-method="createPlugin">
          <constructor-arg type="com.ibm.websphere.objectgrid.config.PluginType"
            value="TRANSACTION_CALLBACK" />
          <constructor-arg type="java.lang.String"
            value="com.company.MyClientTxCallback" />
        </bean>
        <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
          factory-method="createPlugin">
          <constructor-arg type="com.ibm.websphere.objectgrid.config.PluginType"
            value="OBJECTGRID_EVENT_LISTENER" />
          <constructor-arg type="java.lang.String" value="" />
        </bean>
      </list>
 </property>
      <property name="backingMapConfigurations">
        <list>
    <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
           factory-method="createBackingMapConfiguration">
           <constructor-arg type="java.lang.String" value="Customer" />
             <property name="plugins">
     <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
                factory-method="createPlugin">
                <constructor-arg type="com.ibm.websphere.objectgrid.config.PluginType"
    value="EVICTOR" />
   <constructor-arg type="java.lang.String"
                  value="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
              </bean>
            </property>
            <property name="numberOfBuckets" value="1429" />
          </bean>
          <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
            factory-method="createBackingMapConfiguration">
              <constructor-arg type="java.lang.String" value="OrderLine" />
```

```
                <property name="numberOfBuckets" value="701" />
   <property name="timeToLive" value="800" />
   <property name="ttlEvictorType">
                    <value type="com.ibm.websphere.objectgrid.
           TTLType">LAST_ACCESS_TIME</value>
   </property>
            </bean>
          </list>
        </property>
      </bean>

      <bean id="client" factory-bean="manager" factory-method="connect"
        singleton="true">
        <constructor-arg type="java.lang.String">
     <value>localhost:2809</value>
        </constructor-arg>
   <constructor-arg
          type="com.ibm.websphere.objectgrid.security.
      config.ClientSecurityConfiguration">
          <null />
        </constructor-arg>
   <constructor-arg type="java.net.URL">
     <null />
        </constructor-arg>
      </bean>
</beans>
```

Après avoir créé le fichier XML, chargez le fichier et générez l'ObjectGrid à l'aide du fragment de code suivant :

```
BeanFactory beanFactory = new XmlBeanFactory(new

UrlResource("file:test/companyGridSpring.xml"));

ObjectGrid companyGrid = (ObjectGrid) beanFactory.getBean("companyGrid");
```

Pour plus d'informations sur la création d'un fichier XML de descripteur, lire Présentation générale de l'intégration à Spring.

Désactivation du cache local du client

Le cache local est activé par défaut lorsque le verrouillage est configuré sur OPTIMISTIC ou sur NONE. Lorsqu'il est configuré sur PESSIMISTIC, le cache n'est pas activé. Pour désactiver le cache local, vous devez donner la valeur 0 à l'attribut numberOfBuckets dans le fichier de descripteur des remplacements ObjectGrid par les clients.

Suivi par une application des mises à jour des mappes

Lorsqu'une application apporte des modifications à une mappe pendant une transaction, un objet LogSequence assure le suivi de ces modifications. Si l'application modifie une entrée de la mappe, un objet LogElement correspondant détaille la modification.

Les chargeurs reçoivent un objet LogSequence pour une mappe particulière dès qu'une application appelle une méthode flush ou commit dans la transaction. Le chargeur itère sur les objets LogElement dans l'objet LogSequence et applique chaque objet LogElement au dorsal.

Les écouteurs ObjectGridEventListener enregistrés auprès d'une ObjectGrid utilisent également les objets LogSequence. Ces écouteurs reçoivent un objet LogSequence pour chaque mappe dans une transaction validée. Les applications peuvent utiliser ces écouteurs pour attendre la modification de certaines entrées, comme un déclencheur dans une base de données traditionnelle.

Les interfaces ou classes liées aux journaux présentées ci-dessous sont fournies par la structure d'eXtreme Scale :

- com.ibm.websphere.objectgrid.plugins.LogElement
- com.ibm.websphere.objectgrid.plugins.LogSequence
- com.ibm.websphere.objectgrid.plugins.LogSequenceFilter
- com.ibm.websphere.objectgrid.plugins.LogSequenceTransformer

Interface LogElement

Un objet LogElement représente une opération effectuée sur une entrée pendant une transaction. Un objet LogElement comporte plusieurs méthodes permettant d'obtenir les divers attributs associés. Les attributs les plus couramment utilisés sont les attributs type et valeur actuelle extraits par les méthodes getType() et getCurrentValue().

Le type est représenté par une des constantes définies dans l'interface LogElement : INSERT, UPDATE, DELETE, EVICT, FETCH ou TOUCH.

La valeur actuelle représente la nouvelle valeur pour l'opération si cette dernière est INSERT, UPDATE ou FETCH. Si l'opération est TOUCH, DELETE ou EVICT, la valeur actuelle est égale à null. Cette valeur peut être transtypée en ValueProxyInfo lorsqu'une interface ValueInterface est utilisée.

Pour plus de détails sur l'interface LogElement, consultez la documentation relative à l'API.

Interface LogSequence

Dans la plupart des transactions, des opérations sont effectuées sur plusieurs entrées d'une mappe, ce qui implique la création de plusieurs objets LogElement. Il est conseillé de créer un objet qui se comporte en tant que composite de plusieurs objets LogElement. L'interface LogSequence remplit cette fonction en contenant une liste d'objets LogElement.

Pour plus de détails sur l'interface LogSequence, consultez la documentation relative à l'API.

Utilisation des objets LogElement et LogSequence

Les objets LogElement et LogSequence sont fréquemment utilisés dans eXtreme Scale et par les plug-in ObjectGrid écrits par les utilisateurs lorsque des opérations sont propagées d'un composant ou d'un serveur sur un autre composant ou serveur. Par exemple, un objet LogSequence peut être utilisé par la fonction de propagation de la transaction ObjectGrid répartie afin de propager les modifications sur les autres serveurs, ou il peut être mis en oeuvre au niveau du stockage de persistance par le chargeur. L'objet LogSequence est principalement utilisé par les interfaces suivantes.

- com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener
- com.ibm.websphere.objectgrid.plugins.Loader
- com.ibm.websphere.objectgrid.plugins.Evictor
- com.ibm.websphere.objectgrid.Session

Exemple de chargeur

Cette section illustre l'utilisation des objets LogSequence et LogElement dans un chargeur. Un chargeur permet de charger des données à partir d'un stockage de persistance et de les y conserver. La méthode batchUpdate de l'interface Loader utilise l'objet LogSequence :

```
void batchUpdate(TxID txid, LogSequence sequence) throws
   LoaderException, OptimisticCollisionException;
```

La méthode batchUpdate est appelée lorsqu'une ObjectGrid doit refléter toutes les modifications en cours dans le chargeur. Le chargeur obtient une liste des objets LogElement pour la mappe, encapsulés dans un objet LogSequence. L'implémentation de la méthode batchUpdate doit parcourir les modifications et les appliquer au programme d'arrière plan. Le fragment de code suivant montre comment un chargeur utilise un objet LogSequence. Le fragment de code parcourt l'ensemble des modifications et traite par lots trois instructions JDBC (Java Database Connectivity) : inserts, updates et deletes :

```
public void batchUpdate(TxID tx, LogSequence sequence) throws LoaderException
{
    // Etablissez une connexion SQL à utiliser.
    Connection conn = getConnection(tx);
    try
    {
    // Traitez la liste des modifications et créez un ensemble d'instructions
    // préparées pour l'exécution d'une opération SQL par lots update, insert
    // // ou delete. Les instructions sont mises en cache dans stmtCache.
    Iterator iter = sequence.getPendingChanges();
    while(iter.hasNext())
    {
        LogElement logElement = (LogElement)iter.next();
        Object key = logElement.getCacheEntry().getKey();
        Object value = logElement.getCurrentValue();
        switch ( logElement.getType().getCode() )
        {
            case LogElement.CODE_INSERT:
                buildBatchSQLInsert( key, value, conn );
                break;
            case LogElement.CODE_UPDATE:
                buildBatchSQLUpdate( key, value, conn );
                break;
            case LogElement.CODE_DELETE:
                buildBatchSQLDelete( key, conn );
                break;
        }
    }
    // Exécutez les instructions par lots créées par la boucle au-dessus.
    Collection statements = getPreparedStatementCollection( tx, conn );
    iter = statements.iterator();
    while(iter.hasNext())
    {
        PreparedStatement pstmt = (PreparedStatement) iter.next();
        pstmt.executeBatch();
    }
} catch (SQLException e)
{
    LoaderException ex = new LoaderException(e);
    throw ex;
}
}
```

L'exemple précédent illustre la logique de niveau supérieur du traitement de l'argument LogSequence. Toutefois, les détails de la création d'une instruction SQL insert, update ou delete ne sont pas illustrés. La méthode getPendingChanges est

appelée sur l'argument LogSequence pour obtenir un itérateur des objets LogElement qu'un chargeur doit traiter et la méthode LogElement.getType().getCode() sert à déterminer si un objet LogElement correspond à une opération SQL insert, update, ou delete.

Exemple d'expulseur

Les objets LogSequence et LogElement peuvent également être utilisés en tant qu'expulseur. Un expulseur permet d'expulser les entrées de mappe de la mappe de sauvegarde en fonction de certains critères. La méthode apply de l'interface Evictor utilise l'objet LogSequence.

```
/**
* Elle est appelée lors de la validation du cache pour permettre à l'expulseur de
* contrôler l'utilisation des objets
* dans une mappe de sauvegarde. Toutes les entrées correctement expulsées
* sont également signalées.
*
* @param sequence LogSequence des modifications apportées à la mappe
*/
void apply(LogSequence sequence);
```

Interfaces LogSequenceFilter et LogSequenceTransformer

Il est parfois nécessaire de filtrer les objets LogElement de façon à en accepter uniquement certains répondant à des critères spécifiques et à rejeter les autres. Par exemple, il est conseillé de sérialiser un objet LogElement donné en fonction de certains critères.

L'interface LogSequenceFilter résout ce problème à l'aide de la méthode suivante.

```
public boolean accept (LogElement logElement);
```

Cette méthode renvoie la valeur true si l'objet LogElement donné doit être utilisé dans l'opération et renvoie la valeur false si l'objet LogElement donné ne doit pas être utilisé.

LogSequenceTransformer est une classe qui utilise la fonction LogSequenceFilter. Elle utilise l'interface LogSequenceFilter pour filtrer certains objets LogElement, puis pour sérialiser les objets LogElement acceptés. Cette classe comprend deux méthodes. La première méthode est la suivante.

```
public static void serialize(Collection logSequences, ObjectOutputStream stream,
   LogSequenceFilter filter, DistributionMode mode) throws IOException
```

Cette méthode permet au demandeur de fournir un filtre pour déterminer les objets LogElements à inclure dans le processus de sérialisation. Le paramètre DistributionMode permet au demandeur de contrôler le processus de sérialisation. Par exemple, si le mode de distribution est invalidation uniquement, la valeur n'a pas besoin d'être sérialisée. La deuxième méthode de cette classe est la méthode inflate, telle que décrite ci-après.

```
public static Collection inflate(ObjectInputStream stream, ObjectGrid
   objectGrid) throws IOException, ClassNotFoundException
```

La méthode inflate lit le formulaire sérialisé de la séquence de journal qui a été créé par la méthode à partir du flux d'entrée de l'objet fourni.

Activer la réplication de mappes côté client

Il est également possible d'activer la réplication des mappes côté client afin d'accélérer la disponibilité.

Avec eXtreme Scale, vous pouvez répliquer une mappe serveur vers un ou plusieurs clients à l'aide de la réplication asynchrone. Un client peut demander une copie locale en lecture seule d'une mappe côté serveur à l'aide de la méthode ClientReplicableMap.enableClientReplication.

```
void enableClientReplication(Mode mode, int[] partitions,
ReplicationMapListener listener) throws ObjectGridException;
```

Le premier paramètre est le mode de réplication. Il peut s'agir d'une réplication continue ou d'une réplication instantanée. Le deuxième paramètre est une matrice de partitions représentant les partitions à partir desquelles la réplication doit se faire. Si la valeur est nulle ou si la matrice est vide, les données sont répliquées à partir de toutes les partitions. Le dernier paramètre est programme d'écoute permettant de recevoir les événements de réplication du client. Pour plus d'informations, voir les sections sur ClientReplicableMap et ReplicationMapListener dans la documentation relative aux API.

Une fois la réplication activée, le serveur démarre le processus de réplication de la mappe vers le client. A tout moment, le client est en retard de quelques transactions seulement par rapport au serveur.

Exemple d'API DataGrid

Les API DataGrid prennent en charge deux modèles communs de programmation de grille : les mappes parallèles et les réductions parallèles.

Mappe parallèle

Une mappe parallèle permet de traiter les entrées correspondant à un ensemble de clés et renvoie un résultat pour chaque entrée traitée. L'application dresse une liste des clés et reçoit une mappe de paires clé/résultat après avoir appelé une opération Map. Le résultat est le résultat de l'exécution d'une fonction sur l'entrée correspondant à chaque clé. La fonction est fournie par l'application.

Flux d'appels MapGridAgent

Lorsque la méthode AgentManager.callMapAgent est appelée avec une collection de clés, l'instance MapGridAgent est sérialisée et envoyée à chaque partition principale dans laquelle la clé est résolue. Ceci signifie que les données d'instance stockées dans l'agent peuvent être envoyées au serveur. Chaque partition principale détient donc une instance de l'agent. La méthode process est appelée pour chaque instance une fois par clé se résolvant dans la partition. Le résultat est alors sérialisé vers le client et renvoyé à l'appelant dans une instance Map, dans laquelle le résultat est représenté en tant que valeur de la mappe.

Lorsque la méthode AgentManager.callMapAgent est appelée sans collection de clés, l'instance MapGridAgent est sérialisée et envoyée à chaque partition principale. Ceci signifie que les données d'instance stockées dans l'agent peuvent être envoyées au serveur. Chaque partition principale détient donc une instance (partition) de l'agent. La méthode processAllEntries est appelée pour chaque partition. Le résultat de chaque méthode processAllEntries est alors sérialisé vers le

client et renvoyé à l'appelant dans une instance Map. L'exemple suivant suppose la présence d'une entité Person ayant la forme suivante :

```
import com.ibm.websphere.projector.annotations.Entity;
import com.ibm.websphere.projector.annotations.Id;
@Entity
public class Person {
  @Id String ssn;
  String firstName;
  String surname;
  int age;
}
```

La fonction fournie par l'application est écrite en tant que classe implémentant l'interface MapAgentGrid. L'exemple d'agent suivant présente une fonction permettant de renvoyer l'âge d'une personne multiplié par deux.

```
public class DoublePersonAgeAgent implements MapGridAgent, EntityAgentMixin
{
  private static final long serialVersionUID = -2006093916067992974L;

  int lowAge;
  int highAge;

  public Object process(Session s, ObjectMap map, Object key)
  {
    Person p = (Person)key;
    return new Integer(p.age * 2);
  }

  public Map processAllEntries(Session s, ObjectMap map)
  {
    EntityManager em = s.getEntityManager();
    Query q = em.createQuery("select p from Person p where p.age > ?1 and p.age < ?2");
    q.setParameter(1, lowAge);
    q.setParameter(2, highAge);
    Iterator iter = q.getResultIterator();
    Map<Person, Interger> rc = new HashMap<Person, Integer>();
   while(iter.hasNext())
    {
   Person p = (Person)iter.next();
   rc.put(p, (Integer)process(s, map, p));
  }
  return rc;
 }
 public Class getClassForEntity()
 {
  return Person.class;
 }
}
```

Cet exemple présente l'agent Map utilisé pour doubler une personne. Commençons par observer les méthodes process. La première est fournie avec la personne concernée. Elle renvoie simplement le double de l'âge de l'entrée correspondant à cette personne. La seconde est appelée pour chaque partition. Elle recherche tous les objets Person dont l'âge est compris entre lowAge et highAge et renvoie leur âge après l'avoir multiplié par deux.

```
Session s = grid.getSession();
ObjectMap map = s.getMap("Person");
AgentManager amgr = map.getAgentManager();

DoublePersonAgeAgent agent = new DoublePersonAgeAgent();

// l'on fabrique une liste de clés
ArrayList<Person> keyList = new ArrayList<Person>();
Person p = new Person();
p.ssn = "1";
keyList.add(p);
p = new Person ();
p.ssn = "2";
keyList.add(p);

// l'on obtient les résultats pour ces entrées
Map<Tuple, Object> = amgr.callMapAgent(agent, keyList);
```

Cet exemple présente un client obtenant une Session et une référence à la mappe Person. L'opération d'agent est exécutée sur une mappe donnée. L'interface AgentManager est extraite de cette mappe. Une instance de l'agent à appeler est créée et les états nécessaires sont ajoutés à l'objet en définissant des attributs. Dans ce cas, il n'y en a aucun. La liste des clés est alors établie. Une mappe contenant les valeurs correspondant à la personne 1 multipliées par deux et ces mêmes valeurs pour la personne 2 est renvoyée.

L'agent est alors appelé pour cet ensemble de clés. La méthode process de l'agent est appelée sur chaque partition avec certaines clés définies dans la grille en parallèle. Une mappe contenant les résultats fusionnés relatifs à la clé définie est renvoyée. Dans ce cas, une mappe contenant les valeurs correspondant à l'âge de la personne 1 multiplié par deux et les mêmes informations pour la personne 2 est renvoyée.

Si la clé n'existe pas, l'agent est toutefois appelé. Il a ainsi l'opportunité de créer l'entrée de mappe. Si vous utilisez la méthode EntityAgentMixin, la clé à traiter n'est pas l'entité, mais la valeur de la clé du bloc de données pour l'entité. Si les clés ne sont pas connues, il est alors possible de demander à toutes les partitions de rechercher des objets Person d'une certaine forme et de renvoyer leur âge multiplié par deux. Par exemple :

```
Session s = grid.getSession();
  ObjectMap map = s.getMap("Person");
  AgentManager amgr = map.getAgentManager();

  DoublePersonAgeAgent agent = new DoublePersonAgeAgent();
  agent.lowAge = 20;
  agent.highAge = 9999;

  Map m = amgr.callMapAgent(agent);
```

L'exemple précédent présente le gestionnaire d'agents obtenu pour la mappe Person, ainsi que l'agent construit et initialisé avec les âges inférieur et supérieur pour les personnes concernées. L'agent est alors appelé à l'aide de la méthode callMapAgent. Notez qu'aucune clé n'est fournie. La grille d'objets appelle alors l'agent sur chaque partition de la grille en parallèle, puis renvoie les résultats fusionnés au client. Cette opération permet de rechercher tous les objets Person de la grille dont l'âge est compris entre la valeur inférieure et la valeur supérieure et de calculer leur âge multiplié par deux. Les API de la grille peuvent ainsi être utilisés pour exécuter une requête permettant de rechercher des entités correspondant à une certaine requête. L'agent est simplement sérialisé et transporté par l'ObjectGrid vers les partitions pour lesquelles ces entrées sont recherchées. Les résultats sont sérialisés de manière similaire en vue de leur transport vers le client. Utilisez l'API Map avec prudence. Si la grille d'objets hébergeait des téraoctets d'objets et s'exécutait sur plusieurs serveurs, seules les machines les plus puissantes exécutant le client auraient à supporter une charge importante. Cette API doit être utilisée pour traiter un sous-ensemble. Si le nombre de transactions à traiter est plus élevé, nous vous recommandons d'utiliser un agent de réduction pour exécuter le traitement dans la grille plutôt que sur le client.

Réduction parallèle ou agents de regroupement

Ce type de programmation permet de traiter un sous-ensemble d'entrées et de calculer un résultat unique pour celles-ci. Voici des exemples d'un tel résultat :

- valeur minimale
- valeur maximale

- autre fonction propre au métier

Un agent de réduction est codé et appelé de la même manière qu'un agent Map.

Flux d'appels ReduceGridAgent

Lorsque la méthode AgentManager.callReduceAgent est appelée avec une collection de clés, l'instance ReduceGridAgent est sérialisée et envoyée à chaque partition principale dans laquelle la clé est résolue. Ceci signifie que les données d'instance stockées dans l'agent peuvent être envoyées au serveur. Chaque partition principale détient donc une instance de l'agent. La méthode reduce(Session s, ObjectMap map, Collection keys) est appelée une fois par instance (partition) avec le sous-ensemble de clés se résolvant dans la partition. Le résultat de chaque méthode reduce est alors sérialisé vers le client. La méthode reduceResults est appelée sur l'instance ReduceGridAgent du client avec la collection des résultats de chaque appel distant de la méthode reduce. Le résultat de la méthode reduceResults est renvoyé à l'appelant de la méthode callReduceAgent.

Lorsque la méthode AgentManager.callReduceAgent est appelée sans collection de clés, l'instance ReduceGridAgentinstance est sérialisée et envoyée à chaque partition principale. Ceci signifie que les données d'instance stockées dans l'agent peuvent être envoyées au serveur. Chaque partition principale détient donc une instance de l'agent. La méthode reduce(Session s, ObjectMap map) est appelée une fois par instance (partition). Le résultat de chaque méthode reduce est alors sérialisé vers le client. La méthode reduceResults est appelée sur l'instance ReduceGridAgent du client avec la collection des résultats de chaque appel distant de la méthode reduce. Le résultat de la méthode reduceResults est renvoyé à l'appelant de la méthode callReduceAgent. Voici un exemple d'un agent de réduction qui ajoute simplement les âges des entrées correspondantes.

```
public class SumAgeReduceAgent implements ReduceGridAgent, EntityAgentMixin
{
  private static final long serialVersionUID = 2521080771723284899L;

  int lowAge;
  int highAge;

  public Object reduce(Session s, ObjectMap map, Collection keyList)
  {
    Iterator<Person> iter = keyList.iterator();
  int sum = 0;
  while(iter.hasNext())
  {
   Person p = iter.next();
   sume += p.age;
  }
  return new Integer(sum);
 }

 public Object reduce(Session s, ObjectMap map)
 {
  EntityManager em = s.getEntityManager ();
  Query q = em.createQuery("select p from Person p where p.age > ?1 and p.age < ?2");
  q.setParameter(1, lowAge);
  q.setParameter(2, highAge);
  Iterator<Person> iter = q.getResultIterator();
  int sum = 0;
  while(iter.hasNext())
  {
   sum += iter.next().age;
  }
  return new Integer(sum);
 }

 public Class getClassForEntity()
 {
  return Person.class;
 }
}
```

L'exemple précédent présente cet agent. Il se compose de trois parties principales. La première permet le traitement d'un ensemble précis de données sans requête. Elle interagit simplement avec les entrées, en ajoutant leur âge. La somme est renvoyée à la méthode. La deuxième utilise une requête pour sélectionner les entrées à regrouper. Elle ajoute alors tous les âges correspondants. La troisième méthode permet de regrouper les résultats de chaque partition dans un seul résultat. La grille d'objets procède à ce regroupement en parallèle dans la grille. Chaque partition produit un résultat intermédiaire à regrouper avec les résultats intermédiaires des autres partitions. La troisième méthode exécute cette tâche. Dans l'exemple suivant, l'agent est appelé et seuls les âges des personnes âgées de 10 à 20 ans sont regroupés :

```
Session s = grid.getSession();
  ObjectMap map = s.getMap("Person");
  AgentManager amgr = map.getAgentManager();

  SumAgeReduceAgent agent = new SumAgeReduceAgent();

  Person p = new Person();
  p.ssn = "1";
  ArrayList<Person> list = new ArrayList<Person>();
 list.add(p);
 p = new Person ();
 p.ssn = "2";
 list.add(p);
 Integer v = (Integer)amgr.callReduceAgent(agent, list);
```

Fonctions de l'agent

L'agent est libre d'exécuter des opérations ObjectMap ou EntityManager dans le fragment local dans lequel il s'exécute. Il reçoit une Session et il peut ajouter, mettre à jour, interroger, lire ou supprimer des données de la partition que la Session représente. Certaines applications interrogent uniquement les données de la grille, mais vous pouvez aussi écrire permettant d'incrémenter de 1 les âges des personnes correspondant à une certaine requête. Une transaction s'exécute sur la Session lors de l'appel de l'agent. Elle est validée lorsque l'agent renvoie un résultat, à moins qu'une exception se produise.

Traitement des erreurs

Si un agent de mappe est appelé avec une clé inconnue, la valeur renvoyée est un objet d'erreur qui implémente l'interface EntryErrorValue.

Transactions

Un agent de mappe s'exécute dans une transaction distincte du client. Les appels d'agent peuvent être groupés dans une même transaction. Si un agent échoue (renvoie une exception), la transaction est annulée. Tous les agents dont l'exécution dans une transaction a abouti seront annulés en même temps que l'agent ayant échoué. Le gestionnaire d'agents relance l'exécution des agents annulés dont l'exécution a abouti dans une nouvelle transaction.

Pour plus d'informations, consultez la documentation relative à l'API DataGrid.

Chapitre 4. Accès aux données avec le service de données REST

Developpez des applications qui effectuent des opérations à l'aide des protocoles de services de données REST.

Opérations avec le service de données REST

Après avoir démarré le service de données REST d'eXtreme Scale, vous pouvez utiliser n'importe quel client HTTP pour interagir avec ce service. Un navigateur Web, un client PHP, un client Java ou un client WCF Data Services, tous peuvent être utilisés pour lancer n'importe quelle opération de demande prise en charge.

Le service de données REST implémente un sous-ensemble de Microsoft Atom Publishing Protocol : Data Services URI et la spécification Payload Extensions version 1.0 qui fait partie du protocole OData. Le présent chapitre explique quelles sont les fonctionnalités de la spécification qui sont prises en charge et comment elles sont mappées à eXtreme Scale.

URI de racine du service

Microsoft WCF Data Services définit normalement un service par source de données ou par modèle d'entité. Le service de données REST d'eXtreme Scale définit un service par ObjectGrid défini. Chaque ObjectGrid qui est défini dans le fichier XML de substitution par client ObjectGrid eXtreme Scale est automatiquement exposé comme racine distincte de service REST.

L'URI de la racine du service est :

`http://`*`hôte`*`:`*`port`*`/`*`racine_contexte`*`/restservice/`*`nom_grille`*

Où :

- *racine_contexte* est défini lorsqu'on déploie l'application de service de données REST et dépend du serveur d'applications
- *nom_grille* est le nom de l'ObjectGrid

Types de demande

La liste suivante décrit les types de demande Microsoft WCF Data Services pris en charge par le service de données REST d'eXtreme Scale. Pour les détails de chacun des types de demande pris en charge par WCF Data Services, voir MSDN: Request Types.

Types de demande Insert

Les clients peuvent insérer des ressources à l'aide du verbe HTTP POST, mais avec les limitations suivantes :

- demande InsertEntity : prise en charge
- demande InsertLink : prise en charge
- demande InsertMediaResource : non prise en charge en raison des restrictions sur la prise en charge des supports

Pour des informations complémentaires, voir MSDN: Insert Request Types.

Types de demande Update

Les clients peuvent actualiser des ressources à l'aide des verbes HTTP PUT et MERGE, avec les limitations suivantes :

- demande UpdateEntity : prise en charge
- demande UpdateComplexType : non prise en charge en raison des restrictions sur les types complexes
- demande UpdatePrimitiveProperty : prise en charge
- demande UpdateValue : prise en charge
- demande UpdateLink : prise en charge
- demande UpdateMediaResource : non prise en charge en raison des restrictions sur la prise en charge des supports

Pour des informations complémentaires, voir MSDN: Insert Request types.

Types de demande Delete

Les clients peuvent supprimer des ressources à l'aide du verbe HTTP DELETE, mais avec les limitations suivantes :

- demande DeleteEntity : prise en charge
- demande DeleteLink : prise en charge
- demande DeleteValue : prise en charge

Pour des informations complémentaires, voir MSDN: Delete Request Types.

Types de demande Retrieve

Les clients peuvent extraire des ressources à l'aide du verbe HTTP GET, mais avec les limitations suivantes :

- demande RetrieveEntitySet : prise en charge
- demande RetrieveEntity : prise en charge
- demande RetrieveComplexType : non prise en charge en raison des restrictions sur les types complexes
- demande RetrievePrimitiveProperty : prise en charge
- demande RetrieveValue : prise en charge
- demande RetrieveServiceMetadata : prise en charge
- demande RetrieveServiceDocument : prise en charge
- demande RetrieveLink : prise en charge
- demande Retrieve contenant un mappage personnalisable de flux : non prise en charge
- demande RetrieveMediaResource : non prise en charge en raison des restrictions sur les ressources de supports

Pour des informations complémentaires, voir MSDN: Retrieve Request Types.

Options système des requêtes

Les requêtes sont prises en charge qui permettent aux clients d'identifier une collection d'entités ou une entité seule. Les options système de requête sont spécifiées dans un URI de service de données et sont prises en charge avec les limitations suivantes :

- $expand: prise en charge
- $filter: prise en charge
- $orderby: prise en charge

- $format: non pris en charge. Le format acceptable est identifié dans l'en-tête Accept des demandes HTTP
- $skip: prise en charge
- $top: prise en charge

Pour des informations complémentaires, voir MSDN: System Query Options.

Routage des partitions

Le routage des partitions se base sur l'entité racine. Un URI de demande infère une entité racine si son chemin de ressource commence par une entité racine ou par une entité qui a une association directe ou indirecte avec l'entité. Dans un environnement partitionné, toute demande incapable d'inférer une entité racine sera rejetée. Toute demande qui infère une entité racine sera routée vers la bonne partition.

Pour des informations complémentaires sur la définition de schémas avec des associations et des entités racines, voir Evolutivité du modèle de données dans eXtreme Scale et Partitionnement.

Demande Invoke

Les demandes Invoke ne sont pas prises en charge. Pour des informations complémentaires, voir MSDN: Invoke Request.

Demande par lot

Les clients peuvent traiter par lots dans la même demande plusieurs ensembles de modifications ou plusieurs opérations de requête. Cela permet de réduire le nombre des aller-retour vers le serveur et autorise plusieurs demandes à participer à la même transaction. Pour des informations complémentaires, voir MSDN: Batch Request.

Demandes placées en tunnel

Les demandes placées en tunnel ne sont pas prises en charge. Pour des informations complémentaires, voir MSDN: Tunneled Requests.

Protocoles de demande du service de données REST

En général, les protocoles permettant d'interagir avec le service REST sont identiques à ceux décrits dans le protocole WCF Data Services AtomPub. Mais eXtreme Scale fournit des détails supplémentaires dans la perspective eXtreme Scale Entity Model perspective. Les utilisateurs doivent posséder de bonnes connaissances des protocoles WCF Data Services avant de lire cette section. Les utilisateurs peuvent également lire cette section avec la section sur le protocole WCF Data Services.

Des exemples sont fournis pour illustrer la demande et la réponse. Ces exemples s'appliquent au service de données REST d'eXtreme Scale et à WCF Data Services. Les navigateurs Web ne pouvant extraire que des données, les opérations CUD (création, mise à jour et suppression) doivent être effectuées par un autre client (Java, JavaScript™, RUBY ou PHP, par exemple).

Extraction de demandes avec le service de données REST

Une demande RetrieveEntity est utilisée par un client pour extraire une entité eXtreme Scale. La charge de la réponse contient les données d'entité au format AtomPub ou JSON. En outre, l'opérateur système $expand permet de développer les relations. Les relations sont représentées en ligne dans la réponse du service de données comme document de flux Atom (relation to-many) ou document d'entrée Atom (relation to-one).

Conseil : Pour plus de détails sur le protocole RetrieveEntity défini dans WCF Data Services, voir MSDN: RetrieveEntity Request

Extraction d'une entité

L'exemple RetrieveEntity ci-après extrait une entité Customer avec sa clé.

AtomPub

- Méthode

  GET
- URI de la demande :

  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')
- En-tête de la demande :

  Accept: application/atom+xml
- Charge de la demande :

  Aucun
- En-tête de la réponse :

  Content-Type: application/atom+xml
- Charge de la réponse :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<entry xml:base = "http://localhost:8080/wxsrestservice/
restservice" xmlns:d= "http://schemas.microsoft.com/ado/2007/
08/dataservices" xmlns:m = "http://schemas.microsoft.com/ado/2007/
08/dataservices/metadata" xmlns = "http://www.w3.org/2005/Atom">

<category term = "NorthwindGridModel.Customer" scheme = "http://
schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
<id>http://localhost:8080/wxsrestservice/restservice/
   NorthwindGrid/Customer('ACME')</id>
    <title type = "text"/>
    <updated>2009-12-16T19:52:10.593Z</updated>
    <author>
        <name/>
    </author>
    <link rel = "edit" title = "Customer" href = "Customer('ACME')"/>
    <link rel = "http://schemas.microsoft.com/ado/2007/08/
     dataservices/related/
orders" type = "application/atom+xml;type=feed" title =
  "orders" href ="Customer('ACME')/orders"/>
    <content type = "application/xml">
        <m:properties>
            <d:customerId>ACME</d:customerId>
            <d:city m:null = "true"/>
            <d:companyName>RoaderRunner</d:companyName>
            <d:contactName>ACME</d:contactName>
            <d:country m:null = "true"/>
```

```
            <d:version m:type = "Edm.Int32">3</d:version>
          </m:properties>
      </content>
  </entry>
```

- Code de la réponse :

  200 OK

JSON

- Méthode

  GET

- URI de la demande :

  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')

- En-tête de la demande :

  Accept: application/json

- Charge de la demande :

  Aucun

- En-tête de la réponse :

  Content-Type: application/json

- Charge de la réponse :

```
{"d":{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/
   restservice/NorthwindGrid/Customer('ACME')",
   "type":"NorthwindGridModel.Customer"},
   "customerId":"ACME",
   "city":null,
   "companyName":"RoaderRunner",
   "contactName":"ACME",
   "country":null,
   "version":3,
   "orders":{"__deferred":{"uri":"http://localhost:8080/
      wxsrestservice/restservice/
   NorthwindGrid/Customer('ACME')/orders"}}}}
```

- Code de la réponse :

  200 OK

Requêtes

Une requête peut également être utilisée avec une demande RetrieveEntitySet ou RetrieveEntity. Une requête est spécifiée par l'opérateur système $filter.

Pour des explications détaillées de l'opérateur $filter, voir MSDN: Filter System Query Option ($filter).

Le protocole OData prend en charge plusieurs expressions communes. Le service de données REST d'eXtreme Scale prend en charge un sous-ensemble des expressions définies dans la spécification :

- Expressions booléennes :
  - eq, ne, lt, le, gt, ge
  - negate
  - not
  - parenthèse
  - and, or
- Expressions arithmétiques :

  - add
  - sub
  - mul
  - div
- Littéraux de types primitifs
  - string
  - date-time
  - decimal
  - single
  - double
  - int16
  - int32
  - int64
  - binary
  - null
  - byte

Les expressions suivantes *ne sont pas* disponibles :
- Expressions booléennes :
  - isof
  - cast
- Expressions d'appels de méthode
- Expressions arithmétiques :
  - mod
- Littéraux de types primitifs :
  - Guid
- Expressions de membres

Vous trouverez une liste complète et la description des expressions utilisables dans Microsoft WCF Data Services à la 2.2.3.6.1.1 : Common Expression Syntax.

L'exemple suivant illustre une demande RetrieveEntity avec une requête. Dans cet exemple, tous les clients dont le nom de contact est “RoadRunner” sont extraits. Le seul client qui correspond à ce filtre est Customer('ACME'), comme indiqué dans la charge de la réponse.

Restriction : Cette requête ne fonctionne que pour les entités non partitionnées. Si Customer est partitionné, la clé appartenant au client est requise.

AtomPub
- Méthode : GET
- URI de la demande : http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer?$filter=contactName eq 'RoadRunner'
- En-tête de la demande : Accept: application/atom+xml
- Charge de l'entrée : Aucune
- En-tête de la réponse : Content-Type: application/atom+xml
- Charge de la réponse :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<feed
 xml:base="http://localhost:8080/wxsrestservice/restservice"
 xmlns:d="http://schemas.microsoft.com/ado/2007/08/
   dataservices"
 xmlns:m="http://schemas.microsoft.com/ado/2007/08/
   dataservices/metadata"
 xmlns="http://www.w3.org/2005/Atom">
 <title type="text">Customer</title>
 <id>http://localhost:8080/wxsrestservice/restservice/
   NorthwindGrid/Customer </id>
 <updated>2009-09-16T04:59:28.656Z</updated>
 <link rel="self" title="Customer" href="Customer" />
 <entry>
  <category term="NorthwindGridModel.Customer"
   scheme="http://schemas.microsoft.com/ado/2007/08/
   dataservices/scheme" />
  <id>
http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
  Customer('ACME')</id>
  <title type="text" />
  <updated>2009-09-16T04:59:28.656Z</updated>
  <author>
   <name />
  </author>
  <link rel="edit" title="Customer" href="Customer('ACME')" />
  <link
   rel="http://schemas.microsoft.com/ado/2007/08/dataservices/
      related/orders"
   type="application/atom+xml;type=feed" title="orders"
   href="Customer('ACME')/orders" />
  <content type="application/xml">
     <m:properties>
           <d:customerId>ACME</d:customerId>
           <d:city m:null = "true"/>
           <d:companyName>RoaderRunner</d:companyName>
           <d:contactName>ACME</d:contactName>
           <d:country m:null = "true"/>
           <d:version m:type = "Edm.Int32">3</d:version>
       </m:properties>
  </content>
 </entry>
</feed>
```

- Code de la réponse : 200 OK

JSON

- Méthode : GET
- URI de la demande :

  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer?$filter=contactName eq 'RoadRunner'
- En-tête de la demande : Accept: application/json
- Charge de la demande : Aucune
- En-tête de la réponse : Content-Type: application/json
- Charge de la réponse :

```
{"d":[{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/
   restservice/NorthwindGrid/Customer('ACME')",
   "type":"NorthwindGridModel.Customer"},
   "customerId":"ACME",
   "city":null,
   "companyName":"RoaderRunner",
   "contactName":"ACME",
   "country":null,
```

```
"version":3,
"orders":{"__deferred":{"uri":"http://localhost:8080/
  wxsrestservice/restservice/NorthwindGrid/
  Customer('ACME')/orders"}}}]}
```

- Code de la réponse : 200 OK

Opérateur système $expand

L'opérateur système $expand permet de développer des associations. Les associations sont représentées en ligne dans la réponse du service de données. Les associations à plusieurs valeurs (to-many) sont représentées comme document de flux Atom ou tableau JSON. Les associations à valeur unique (to-one) sont représentées comme document d'entrée Atom ou objet JSON.

Pour plus de détails sur l'opérateur $expand, voir Expand System Query Option ($expand).

Voici un exemple d'utilisation de l'opérateur système $expand. Dans cet exemple, nous extrayons l'entité Customer('IBM') associée entre autres aux commandes 5000 et 5001. La clause $expand reçoit la valeur “orders”, de sorte que la collection de commandes soit étendue comme en ligne dans la charge de la réponse. Seules les commandes 5000 et 5001 sont affichées ici.

AtomPub

- Méthode : GET
- URI de la demande : http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')?$expand=orders
- En-tête de la demande : Accept: application/atom+xml
- Charge de la demande : Aucune
- En-tête de la réponse : Content-Type: application/atom+xml
- Charge de la réponse :

```
<?xml version="1.0" encoding="utf-8"?>
<entry xml:base = "http://localhost:8080/wxsrestservice/restservice"
   xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
   xmlns:m = "http://schemas.microsoft.com/ado/2007/08/dataservices/
   metadata" xmlns = "http://www.w3.org/2005/Atom">
<category term = "NorthwindGridModel.Customer" scheme = "http://schemas.

microsoft.com/ado/2007/08/dataservices/scheme"/>
    <id>http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
    Customer('IBM')</id>
    <title type = "text"/>
    <updated>2009-12-16T22:50:18.156Z</updated>
    <author>
        <name/>
    </author><link rel = "edit" title = "Customer"

     href = "Customer('IBM')"/>
    <link rel = "http://schemas.microsoft.com/ado/2007/08/dataservices/
    related/orders" type = "application/atom+xml;type=feed" title =
    "orders" href = "Customer('IBM')/orders">
        <m:inline>
            <feed>
                <title type = "text">orders</title>
                <id>http://localhost:8080/wxsrestservice/restservice/
             NorthwindGrid/Customer('IBM')/orders</id>
                <updated>2009-12-16T22:50:18.156Z</updated>
                <link rel = "self" title = "orders" href = "Customer('IBM')/orders"/>
                <entry>
```

```
                    <category term = "NorthwindGridModel.Order" scheme =
             "http://schemas.microsoft.com/ado/2007/08/
             dataservices/scheme"/>
                    <id>http://localhost:8080/wxsrestservice/restservice/
            NorthwindGrid/Order(orderId=5000,customer_customerId=
              'IBM')</id>
                    <title type = "text"/>
                    <updated>2009-12-16T22:50:18.156Z</updated>
                    <author>
                        <name/>
                    </author>
                    <link rel = "edit" title = "Order" href =
            "Order(orderId=5000,customer_customerId='IBM')"/>
                    <link rel = "http://schemas.microsoft.com/ado/2007/08/
            dataservices/related/customer" type = "application/
            atom+xml;type=entry" title ="customer" href =
            "Order(orderId=5000,customer_customerId='IBM')/customer"/>
                    <link rel = "http://schemas.microsoft.com/ado/2007/08/
            dataservices/related/orderDetails" type = "application/
             atom+xml;type=feed" title ="orderDetails" href =
             "Order(orderId=5000,customer_customerId='IBM')/orderDetails"/>
                    <content type = "application/xml">
                        <m:properties>
                            <d:orderId m:type = "Edm.Int32">5000</d:orderId>
                            <d:customer_customerId>IBM</d:customer_customerId>
                            <d:orderDate m:type = "Edm.DateTime">
                 2009-12-16T19:46:29.562</d:orderDate>
                            <d:shipCity>Rochester</d:shipCity>
                            <d:shipCountry m:null = "true"/>
                            <d:version m:type = "Edm.Int32">0</d:version>
                        </m:properties>
                    </content>
                </entry>
                <entry>
                    <category term = "NorthwindGridModel.Order" scheme =
             "http://schemas.microsoft.com/ado/2007/08/
             dataservices/scheme"/>
                    <id>http://localhost:8080/wxsrestservice/restservice/
            NorthwindGrid/Order(orderId=5001,customer_customerId=
            'IBM')</id>
                    <title type = "text"/>
                    <updated>2009-12-16T22:50:18.156Z</updated>
                    <author>
                        <name/></author>
                    <link rel = "edit" title = "Order" href = "Order(
orderId=5001,customer_customerId='IBM')"/>
                    <link rel = "http://schemas.microsoft.com/ado/2007/
              08/dataservices/related/customer"
type = "application/atom+xml;type=entry"
title = "customer" href = "Order(orderId=5001,customer_customerId=
             'IBM')/customer"/>
                    <link rel = "http://schemas.microsoft.com/ado/2007/08/
              dataservices/related/orderDetails" type = "application/atom+xml;type=feed"
title = "orderDetails"
href = "Order(orderId=5001,customer_customerId='IBM')/orderDetails"/>
                    <content type = "application/xml">
                        <m:properties>
                            <d:orderId m:type = "Edm.Int32">5001</d:orderId>
                            <d:customer_customerId>IBM</d:customer_customerId>
                            <d:orderDate m:type = "Edm.DateTime">2009-12-16T19:
                50:11.125</d:orderDate>
                            <d:shipCity>Rochester</d:shipCity>
                            <d:shipCountry m:null = "true"/>
                            <d:version m:type = "Edm.Int32">0</d:version>
                        </m:properties>
                    </content>
                 </entry>
```

```
            </feed>
        </m:inline>
    </link>
    <content type = "application/xml">
        <m:properties>
            <d:customerId>IBM</d:customerId>
            <d:city m:null = "true"/>
            <d:companyName>IBM Corporation</d:companyName>
            <d:contactName>John Doe</d:contactName>
            <d:country m:null = "true"/>
            <d:version m:type = "Edm.Int32">4</d:version>
        </m:properties>
    </content>
</entry>
```

- Code de la réponse : 200 OK

JSON

- Méthode : GET
- URI de la demande : http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')?$expand=orders
- En-tête de la demande : Accept: application/json
- Charge de la demande : Aucune
- En-tête de la réponse : Content-Type: application/json
- Charge de la réponse :

```
{"d":{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/
   restservice/NorthwindGrid/Customer('IBM')",
 "type":"NorthwindGridModel.Customer"},
"customerId":"IBM",
"city":null,
"companyName":"IBM Corporation",
"contactName":"John Doe",
"country":null,
"version":4,
"orders":[{"__metadata":{"uri":"http://localhost:8080/
    wxsrestservice/restservice/NorthwindGrid/Order(
    orderId=5000,customer_customerId='IBM')",
"type":"NorthwindGridModel.Order"},
"orderId":5000,
"customer_customerId":"IBM",
"orderDate":"\/Date(1260992789562)\/",
"shipCity":"Rochester",
"shipCountry":null,
"version":0,
"customer":{"__deferred":{"uri":"http://localhost:8080/
   wxsrestservice/restservice/NorthwindGrid/Order(
   orderId=5000,customer_customerId='IBM')/customer"}},
"orderDetails":{"__deferred":{"uri":"http://localhost:
     8080/wxsrestservice/restservice/NorthwindGrid/
     Order(orderId=5000,customer_customerId='IBM')/
     orderDetails"}}},
{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/
     restservice/NorthwindGrid/Order(orderId=5001,
     customer_customerId='IBM')","type":
     "NorthwindGridModel.Order"},
"orderId":5001,
"customer_customerId":"IBM",
"orderDate":"\/Date(1260993011125)\/",
"shipCity":"Rochester",
"shipCountry":null,
"version":0,
"customer":{"__deferred":{"uri":"http://localhost:8080/wxsrestservice
/restservice/NorthwindGrid/Order(orderId=5001,customer_customerId='IBM')
```

```
/customer"}},"orderDetails":{"__deferred":{"uri":"http://localhost:8080/
   wxsrestservice/restservice/NorthwindGrid/Order(
   orderId=5001, customer_customerId='IBM')/
   orderDetails"}}}]}}
```

- Code de la réponse : 200 OK

Extraction d'éléments autres que des entités à l'aide des services de données REST

Le service de données REST permet d'extraire bien plus que des entités et notamment des collections d'entités, des propriétés, etc.

Extraction d'une collection d'entités

Une demande RetrieveEntitySet peut être utilisée par un client pour extraire un ensemble d'entités eXtreme Scale. Les entités sont représentées comme un document de flux Atom ou un cluster JSON dans la charge de la réponse. Pour plus de détails sur le protocole RetrieveEntitySet défini dans WCF Data Services, voir MSDN: RetrieveEntitySet Request.

L'exemple de demande RetrieveEntitySet ci-après extrait toutes les entités Order associées à l'entité Customer('IBM'). Seules les commandes 5000 et 5001 sont affichées ici.

AtomPub

- Méthode : GET
- URI de la demande : http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/orders
- En-tête de la demande : Accept: application/atom+xml
- Charge de la demande : Aucune
- En-tête de la réponse : Content-Type: application/atom+xml
- Charge de la réponse :

```
<?xml version="1.0" encoding="utf-8"?>
<feed xml:base = "http://localhost:8080/wxsrestservice/restservice"
   xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
   xmlns:m = "http://schemas.microsoft.com/ado/2007/08/dataservices/
   metadata" xmlns = "http://www.w3.org/2005/Atom">
    <title type = "text">Order</title>
    <id>http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
     Order</id>
    <updated>2009-12-16T22:53:09.062Z</updated>
    <link rel = "self" title = "Order" href = "Order"/>
    <entry>
        <category term = "NorthwindGridModel.Order" scheme = "http://
      schemas.microsoft.com/
      ado/2007/08/dataservices/scheme"/>
        <id>http://localhost:8080/wxsrestservice/restservice/
      NorthwindGrid/Order(orderId=5000,customer_customerId=
      'IBM')</id>
        <title type = "text"/>
        <updated>2009-12-16T22:53:09.062Z</updated>
        <author>
            <name/>
        </author>
        <link rel = "edit" title = "Order" href = "Order(orderId=5000,
      customer_customerId='IBM')"/>
        <link rel = "http://schemas.microsoft.com/ado/2007/08/
      dataservices/related/customer"
      type = "application/atom+xml;type=entry"
```

```
      title = "customer" href = "Order(orderId=5000,
      customer_customerId='IBM')/customer"/>
        <link rel = "http://schemas.microsoft.com/ado/2007/08/
      dataservices/related/orderDetails"
      type = "application/atom+xml;type=feed"
      title = "orderDetails"
        href = "Order(orderId=5000,customer_customerId='IBM')/
    orderDetails"/>
        <content type = "application/xml">
            <m:properties>
                <d:orderId m:type = "Edm.Int32">5000</d:orderId>
                <d:customer_customerId>IBM</d:customer_customerId>
                <d:orderDate m:type = "Edm.DateTime">2009-12-16T19:
          46:29.562</d:orderDate>
                <d:shipCity>Rochester</d:shipCity>
                <d:shipCountry m:null = "true"/>
                <d:version m:type = "Edm.Int32">0</d:version>
            </m:properties>
        </content>
    </entry>
    <entry>
        <category term = "NorthwindGridModel.Order" scheme = "http://
      schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
        <id>http://localhost:8080/wxsrestservice/restservice/
      NorthwindGrid/Order(orderId=5001, customer_customerId='IBM')
    </id>
        <title type = "text"/>
        <updated>2009-12-16T22:53:09.062Z</updated>
        <author>
            <name/>
        </author>
        <link rel = "edit" title = "Order" href = "Order(orderId=5001,
       customer_customerId='IBM')"/>
        <link rel = "http://schemas.microsoft.com/ado/2007/08/
       dataservices/related/customer"
       type = "application/atom+xml;type=entry"
       title = "customer" href = "Order(orderId=5001,
         customer_customerId='IBM')/customer"/>
        <link rel = "http://schemas.microsoft.com/ado/2007/08/
         dataservices/related/orderDetails"
       type = "application/atom+xml;type=feed"
       title = "orderDetails" href = "Order(orderId=5001,
         customer_customerId='IBM')/orderDetails"/>
        <content type = "application/xml">
            <m:properties>
                <d:orderId m:type = "Edm.Int32">5001</d:orderId>
                <d:customer_customerId>IBM</d:customer_customerId>
                <d:orderDate m:type = "Edm.DateTime">2009-12-16T19:50:
          11.125</d:orderDate>
                <d:shipCity>Rochester</d:shipCity>
                <d:shipCountry m:null = "true"/>
                <d:version m:type = "Edm.Int32">0</d:version>
            </m:properties>
        </content>
    </entry>
</feed>
```

- Code de la réponse : 200 OK

JSON

- Méthode : GET
- URI de la demande : http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/orders
- En-tête de la demande : Accept: application/json
- Charge de la demande : Aucune

- En-tête de la réponse : Content-Type: application/json
- Charge de la réponse :

```
{"d":[{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/
   restservice/NorthwindGrid/Order(orderId=5000,
   customer_customerId='IBM')",
"type":"NorthwindGridModel.Order"},
"orderId":5000,
"customer_customerId":"IBM",
"orderDate":"\/Date(1260992789562)\/",
"shipCity":"Rochester",
"shipCountry":null,
"version":0,
"customer":{"__deferred":{"uri":"http://localhost:8080/
   wxsrestservice/restservice/NorthwindGrid/Order(orderId=
   5000,customer_customerId='IBM')/customer"}},
"orderDetails":{"__deferred":{"uri":"http://localhost:8080/
   wxsrestservice/restservice/NorthwindGrid/Order(orderId=
   5000,customer_customerId='IBM')/orderDetails"}}},
{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/
     restservice/NorthwindGrid/
   Order(orderId=5001,
   customer_customerId='IBM')",
"type":"NorthwindGridModel.Order"},
"orderId":5001,
"customer_customerId":"IBM",
"orderDate":"\/Date(1260993011125)\/",
"shipCity":"Rochester",
"shipCountry":null,
"version":0,
"customer":{"__deferred":{"uri":"http://localhost:8080/
   wxsrestservice/restservice/NorthwindGrid/Order(orderId=
   5001,customer_customerId='IBM')/customer"}},
"orderDetails":{"__deferred":{"uri":"http://localhost:8080/
   wxsrestservice/restservice/NorthwindGrid/Order(orderId=
   5001,customer_customerId='IBM')/orderDetails"}}}]}
```

- Code de la réponse : 200 OK

Extraction d'une propriété

Une demande RetrievePrimitiveProperty peut être utilisée pour extraire la valeur d'une propriété d'une instance d'entité eXtreme Scale. La valeur de la propriété est représentée au format XML pour les demandes AtomPub et comme objet JSON pour les demandes JSON dans la charge de la réponse. Pour plus de détails sur la demande RetrievePrimitiveProperty, voir MSDN: RetrievePrimitiveProperty Request.

L'exemple suivant de demande RetrievePrimitiveProperty extrait la propriété contactName de l'entité Customer('IBM').

AtomPub

- Méthode : GET
- URI de la demande : http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName
- En-tête de la demande : Accept: application/xml
- Charge de la demande : Aucune
- En-tête de la réponse : Content-Type: application/atom+xml
- Charge de la réponse :

```
<contactName xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices">
    John Doe
</contactName>
```

- Code de la réponse : 200 OK

JSON

- Méthode : GET
- URI de la demande : http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName
- En-tête de la demande : Accept: application/json
- Charge de la demande : Aucune
- En-tête de la réponse : Content-Type: application/json
- Charge de la réponse : `{"d":{"contactName":"John Doe"}}`
- Code de la réponse : 200 OK

Extraction d'une valeur de propriété

Une demande RetrieveValue peut être utilisée pour extraire la valeur brute d'une propriété sur une instance d'entité eXtreme Scale. La valeur de la propriété est représentée comme valeur brute dans la charge de la réponse. Si le type d'entité est l'un des suivants, le type MIME de la réponse est “text/plain”. Sinon, le type MIME est “application/octet-stream." Ces types sont les suivants :

- types primitifs Java et leurs encapsuleurs respectifs
- java.lang.String
- byte[]
- Byte[]
- char[]
- Character[]
- enums
- java.math.BigInteger
- java.math.BigDecimal
- java.util.Date
- java.util.Calendar
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp

Pour plus de détails sur la demande RetrieveValue, voir MSDN: RetrieveValue Request.

L'exemple de demande RetrieveValue ci-après extrait la valeur brute de la propriété contactName de l'entité Customer('IBM').

- Méthode de la demande : GET
- URI de la demande : http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName/$value
- En-tête de la demande : Accept: text/plain
- Charge de la demande : Aucune
- En-tête de la réponse : Content-Type: text/plain
- Charge de la réponse : John Doe

- Code de la réponse : 200 OK

Extraction d'un lien

Une demande RetrieveLink peut être utilisée pour extraire les liens représentant une association to-one ou to-many. Pour l'association to-one, il s'agit d'un lien d'une instance d'entité eXtreme Scale vers une autre et le lien est représenté dans la charge de la réponse. Pour l'association to-many, il s'agit de liens d'une instance d'entité eXtreme Scale vers toutes les autres instances dans une collection d'entités eXtreme Scale spécifiées et la réponse est représentée sous forme de liens dans la charge de la réponse. Pour plus de détails sur la demande RetrieveLink, voir MSDN: RetrieveLink Request.

Voici un exemple de demande RetrieveLink. Dans cet exemple, nous extrayons l'association entre l'entité Order(orderId=5000,customer_customerId='IBM') et son client. La réponse indique l'URI de l'entité Customer.

AtomPub

- Méthode : GET
- URI de la demande : http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/$links/customer
- En-tête de la demande : Accept: application/xml
- Charge de la demande : Aucune
- En-tête de la réponse : Content-Type: application/xml
- Charge de la réponse :

```
<?xml version="1.0" encoding="utf-8"?>
<uri>http://localhost:8080/wxsrestservice/restservice/
   NorthwindGrid/Customer('IBM')</uri>
```

- Code de la réponse : 200 OK

JSON

- Méthode : GET
- URI de la demande : http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/$links/customer
- En-tête de la demande : Accept: application/json
- Charge de la demande : Aucune
- En-tête de la réponse : Content-Type: application/json
- Charge de la réponse : {"d":{"uri":"http:\/\/localhost:8080\/wxsrestservice\/restservice\/NorthwindGrid\/Customer('IBM')"}}

Extraction des métadonnées de service

Une demande RetrieveServiceMetadata peut être utilisée pour extraire le document CSDL (conceptual schema definition language), qui décrit le modèle de données associé au service de données REST d'eXtreme Scale. Pour plus de détails sur la demande RetrieveServiceMetadata, voir MSDN: RetrieveServiceMetadata Request.

Extraction du document de service

Une demande RetrieveServiceDocument peut être utilisée pour extraire le document de service qui décrit la collection de ressources exposée par le service de

données REST d'eXtreme Scale. Pour plus de détails sur la demande RetrieveServiceDocument, voir MSDN: RetrieveServiceDocument Request.

Demandes d'insertion avec les services de données REST

Une demande InsertEntity peut être utilisée pour insérer dans le service de données REST d'eXtreme Scale une nouvelle instance d'entité eXtreme Scale, avec éventuellement de nouvelles entités associées.

Demande d'insertion d'entité

Une demande InsertEntity peut être utilisée pour insérer dans le service de données REST d'eXtreme Scale une nouvelle instance d'entité eXtreme Scale, avec éventuellement de nouvelles entités associées. Lors de l'insertion d'une entité, le client peut indiquer si la ressource ou l'entité doit être automatiquement associée aux autres entités existantes du service de données.

Le client doit inclure les informations de liaison requises dans la représentation de la relation associée dans la charge de la demande.

En plus de prendre en charge l'insertion d'une nouvelle instance EntityType (E1), la demande InsertEntity permet également l'insertion de nouvelles entités associées à E1 (décrite par une relation d'entité) dans une même demande. Par exemple, lors de l'insertion de l'entité Customer('IBM'), nous pouvons insérer toutes les commandes avec l'entité Customer('IBM'). Cette forme de demande InsertEntity est également connue sous le nom d'*insertion profonde*. Avec une insertion profonde, les entités associées doivent être représentées à l'aide de la représentation en ligne de la relation associée à E1 qui identifie le lien vers les entités associées (à insérer).

Les propriétés de l'entité à insérer sont spécifiées dans la charge de la demande. Les propriétés sont analysées par le service de données REST, puis définies en fonction de la propriété correspondante sur l'instance d'entité. Pour le format AtomPub, la propriété est spécifiée comme un élément XML <d:NOM_PROPRIETE>. Pour JSON, la propriété est spécifiée comme une propriété d'un objet JSON.

Si une propriété est manquante dans la charge de la demande, le service de données REST spécifie comme valeur de propriété d'entité la valeur par défaut java. Toutefois, le système dorsal de la base de données peut rejeter une telle valeur par défaut si, par exemple, la colonne n'admet pas la valeur null dans la base de données. Un code de réponse 500 est renvoyé pour indiquer une erreur de serveur interne.

Si des propriétés sont spécifiées en double dans la charge, la dernière propriété est utilisée. Toutes les valeurs précédentes possédant le même nom de propriété sont ignorées par le service de données REST.

Si la charge contient une propriété inexistante, le service de données REST renvoie un code de réponse 400 (Demande incorrecte) pour indiquer que la demande envoyée par le client est syntaxiquement incorrecte.

Si les propriétés de clé sont manquantes, le service de données REST renvoie un code de réponse 400 (Demande incorrecte) pour indiquer une propriété de clé manquante.

Si la charge contient un lien vers une entité associée avec une clé inexistante, le service de données REST renvoie un code de réponse 404 (Introuvable) pour indiquer que l'entité associée est introuvable.

Si la charge contient un lien vers une entité associée avec un nom d'association incorrect, le service de données REST renvoie un code de réponse 400 (Demande incorrecte) pour indiquer que le lien est introuvable.

Si la charge contient plusieurs liens vers une relation to-one, le dernier lien est utilisé. Tous les liens précédents pour la même association sont ignorés.

Pour plus de détails sur la demande InsertEntity, voir MSDN Library: InsertEntity Request.

Une demande InsertEntity insère une entité Customer avec la clé 'IBM'.

AtomPub

- Méthode : POST
- URI de la demande : http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')
- En-tête de la demande : Accept: application/atom+xml Content-Type: application/atom+xml
- Charge de la demande :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<entry xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
 xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
 xmlns="http://www.w3.org/2005/Atom">
 <category term="NorthwindGridModel.Customer"
  scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
<content type="application/xml">
  <m:properties>
   <d:customerId>Rational</d:customerId>
   <d:city>Rochester</d:city>
   <d:companyName>Rational</d:companyName>
   <d:contactName>John Doe</d:contactName>
   <d:country>USA</d:country>
  </m:properties>
 </content>
</entry>
```

- En-tête de la réponse : Content-Type: application/atom+xml
- Charge de la réponse :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<entry xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
 xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
 xmlns="http://www.w3.org/2005/Atom">
 <category term="NorthwindGridModel.Customer"
  scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
<content type="application/xml">
  <m:properties>
   <d:customerId>Rational</d:customerId>
   <d:city>Rochester</d:city>
   <d:companyName>Rational</d:companyName>
   <d:contactName>John Doe</d:contactName>
   <d:country>USA</d:country>
  </m:properties>
 </content>
</entry>
En-tête de la réponse :
Content-Type: application/atom+xml
Charge de la réponse :
```

```
<?xml version="1.0" encoding="utf-8"?>
<entry xml:base = "http://localhost:8080/wxsrestservice/restservice" xmlns:d =
   "http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m =
     "http://schemas.microsoft.com/
   ado/2007/08/dataservices/metadata" xmlns = "http://www.w3.org/2005/Atom">
    <category term = "NorthwindGridModel.Customer" scheme = "http://schemas.
     microsoft.com/ado/2007/08/dataservices/scheme"/>
    <id>http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
     Customer('Rational')</id>
    <title type = "text"/>
    <updated>2009-12-16T23:25:50.875Z</updated>
    <author>
        <name/>
    </author>
    <link rel = "edit" title = "Customer" href = "Customer('Rational')"/>
    <link rel = "http://schemas.microsoft.com/ado/2007/08/dataservices/related/
    orders" type = "application/atom+xml;type=feed"
    title = "orders" href = "Customer('Rational')/orders"/>
    <content type = "application/xml">
        <m:properties>
            <d:customerId>Rational</d:customerId>
            <d:city>Rochester</d:city>
            <d:companyName>Rational</d:companyName>
            <d:contactName>John Doe</d:contactName>
            <d:country>USA</d:country>
            <d:version m:type = "Edm.Int32">0</d:version>
        </m:properties>
    </content>
</entry>
```

- Code de la réponse : 201 Créé

JSON

- Méthode : POST
- URI de la demande : http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer
- En-tête de la demande : Accept: application/json Content-Type: application/json
- Charge de la demande :

```
{"customerId":"Rational",
"city":null,
"companyName":"Rational",
"contactName":"John Doe",
"country": "USA",}
```

- En-tête de la réponse : Content-Type: application/json
- Charge de la réponse :

```
{"d":{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/restservice/
   NorthwindGrid/Customer('Rational')",
"type":"NorthwindGridModel.Customer"},
"customerId":"Rational",
"city":null,
"companyName":"Rational",
"contactName":"John Doe",
"country":"USA",
"version":0,
"orders":{"__deferred":{"uri":"http://localhost:8080/wxsrestservice/restservice/
   NorthwindGrid/Customer('Rational')/orders"}}}}
```

- Code de la réponse : 201 Créé

Demande d'insertion de lien

Une demande InsertLink peut être utilisée pour créer un lien entre deux instances d'entité eXtreme Scale. L'URI de la demande doit se résoudre en association eXtreme Scale to-many. La charge de la demande contient un seul lien qui pointe vers l'entité cible de l'association to-many.

Si l'URI de la demande InsertLink représente une association to-one, le service de données REST renvoie une réponse 400 (Demande incorrecte).

Si l'URI de la demande InsertLink pointe vers une association inexistante, le service de données REST renvoie une réponse 404 (Introuvable) pour indiquer que le lien est introuvable.

Si la charge contient un lien avec une clé inexistante, le service de données REST renvoie une réponse 404 (Introuvable) pour indiquer que l'entité associée est introuvable.

Si la charge contient plusieurs liens, le service de données REST d'eXtreme Scale analyse le premier lien. Les autres liens sont ignorés.

Pour plus de détails sur la demande InsertLink, voir MSDN Library: InsertLink Request.

L'exemple de demande InsertLink ci-après crée un lien entre Customer('IBM') et Order(orderId=5000,customer_customerId='IBM').

AtomPub

- Méthode : POST
- URI de la demande : http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/$link/orders
- En-tête de la demande : Content-Type: application/xml
- Charge de la demande :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<uri>http://host:1000/wxsrestservice/restservice/NorthwindGrid/Order(orderId=
   5000,customer_customerId='IBM')</uri>
```

- Charge de la réponse : Aucune
- Code de la réponse : 204 Pas de contenu

JSON

- Méthode : POST
- URI de la demande : http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/$links/orders
- En-tête de la demande : Content-Type: application/json
- Charge de la demande :

```
{"uri": "http://host:1000/wxsrestservice/restservice/NorthwindGrid/Order(orderId
    =5000,customer_customerId='IBM')"}
```

- Charge de la réponse : Aucune
- Code de la réponse : 204 Pas de contenu

Demandes de mise à jour avec les services de données REST

Le service de données REST de WebSphere eXtreme Scale prend en charge les demandes de mise à jour d'entités, de propriétés primitives d'entités, etc.

Mise à jour d'une entité

Une demande UpdateEntity permet de mettre à jour une entité eXtreme Scale existante. Le client peut utiliser une méthode HTTP PUT pour remplacer une entité eXtreme Scale existante ou utiliser une méthode HTTP MERGE pour fusionner les modifications dans une entité eXtreme Scale existante.

Lors de la mise à jour de l'entité, le client peut indiquer si cette dernière (en plus d'être mise à jour) doit être automatiquement associée aux autres entités existantes du service de données qui sont associées via des associations à valeur unique (to-one).

La propriété de l'entité à mettre à jour se trouve dans la charge de la demande. La propriété est analysée par le service de données REST, puis définie en fonction de la propriété correspondante sur l'entité. Pour le format AtomPub, la propriété est spécifiée comme un élément XML <d:NOM_PROPRIETE>. Pour JSON, la propriété est spécifiée comme une propriété d'un objet JSON.

Si une propriété est manquante dans la charge de la demande, le service de données REST spécifie comme valeur de propriété d'entité la valeur par défaut java de la méthode HTTP PUT. Toutefois, le système dorsal de la base de données peut rejeter une telle valeur par défaut si, par exemple, la colonne n'admet pas la valeur null dans la base de données. Un code de réponse 500 (Erreur de serveur interne) est renvoyé pour indiquer une erreur de serveur interne. Si une propriété est manquante dans la charge de la demande HTTP MERGE, le service de données REST ne modifie pas la valeur de propriété existante.

Si des propriétés sont spécifiées en double dans la charge, la dernière propriété est utilisée. Toutes les valeurs précédentes possédant le même nom de propriété sont ignorées par le service de données REST.

Si la charge contient une propriété inexistante, le service de données REST renvoie un code de réponse 400 (Demande incorrecte) pour indiquer que la demande envoyée par le client est syntaxiquement incorrecte.

Dans le cadre de la sérialisation d'une ressource, si la charge d'une demande de mise à jour contient l'une des propriétés de clé de l'entité, le service de données REST ignore ces valeurs de clé car les clés d'entité ne peuvent pas être modifiées.

Pour des détails sur la demande UpdateEntity, voir MSDN Library: UpdateEntity Request.

Une demande UpdateEntity met à jour le nom de la ville de Customer('IBM'), en spécifiant 'Raleigh'.

AtomPub

- Méthode : PUT
- URI de la demande : http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')
- En-tête de la demande : Content-Type: application/atom+xml
- Charge de la demande :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<entry xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
 xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
 xmlns="http://www.w3.org/2005/Atom">
```

```
<category term="NorthwindGridModel.Customer"
 scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
<title />
<updated>2009-07-28T21:17:50.609Z</updated>
<author>
 <name />
</author>
<id />
<content type="application/xml">
 <m:properties>
  <d:customerId>IBM</d:customerId>
  <d:city>Raleigh</d:city>
  <d:companyName>IBM Corporation</d:companyName>
  <d:contactName>Big Blue</d:contactName>
  <d:country>USA</d:country>
 </m:properties>
</content>
</entry>
```

- Charge de la réponse : Aucune
- Code de la réponse : 204 Pas de contenu

JSON

- Méthode : PUT
- URI de la demande : http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')
- En-tête de la demande : Content-Type: application/json
- Charge de la demande :

```
{"customerId":"IBM",
"city":"Raleigh",
"companyName":"IBM Corporation",
"contactName":"Big Blue",
"country":"USA",}
```

- Charge de la réponse : Aucune
- Code de la réponse : 204 Pas de contenu

Mise à jour d'une propriété primitive d'entité

La demande UpdatePrimitiveProperty peut mettre à jour la valeur d'une propriété d'entité eXtreme Scale. La propriété et la valeur à mettre à jour se trouvent dans la charge de la demande. La propriété ne peut pas être une propriété de clé car eXtreme Scale ne permet pas aux clients de modifier les clés d'entité.

Pour plus de détails sur la demande UpdatePrimitiveProperty, voir MSDN Library: UpdatePrimitiveProperty Request.

Voici un exemple de demande UpdatePrimitiveProperty. Dans cet exemple, nous mettons à jour le nom de la ville du Customer('IBM') en spécifiant 'Raleigh'.

AtomPub

- Méthode : PUT
- URI de la demande : http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/city
- En-tête de la demande : Content-Type: application/xml
- Charge de la demande :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<city xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices">
 Raleigh
</city>
```

- Charge de la réponse : Aucune
- Code de la réponse : 204 Pas de contenu

JSON

- Méthode : PUT
- URI de la demande : http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/city
- En-tête de la demande : Content-Type: application/json
- Charge de la demande : {"city":"Raleigh"}
- Charge de la réponse : Aucune
- Code de la réponse : 204 Pas de contenu

Mise à jour d'une valeur de propriété de primitive d'entité

La demande UpdateValue peut mettre à jour une valeur de propriété brute d'une entité eXtreme Scale. La valeur à mettre à jour est représentée comme valeur brute dans la charge de la demande. La propriété ne peut pas être une propriété de clé car eXtreme Scale ne permet pas aux clients de modifier les clés d'entité.

Le type de contenu de la demande peut être “text/plain” ou “application/octet-stream” suivant le type de propriété. Pour plus d'informations, voir la section 6.3.1.4.

Pour plus de détails sur la demande UpdateValue, voir MSDN Library: UpdateValue Request

Voici un exemple de demande UpdateValue. Dans cet exemple, nous mettons à jour le nom de la ville de Customer('IBM') en spécifiant 'Raleigh'.

- Méthode : PUT
- URI de la demande : http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/city/$value
- En-tête de la demande : Content-Type: text/plain
- Charge de la demande : Raleigh
- Charge de la réponse : Aucune
- Code de la réponse : 204 Pas de contenu

Mise à jour d'un lien

La demande UpdateLink permet d'établir une association entre deux eXtreme Scale instances d'entités. L'association peut être une relation à valeur unique (to-one) ou à plusieurs valeurs (to-many).

La mise à jour d'un lien entre deux instances d'entité eXtreme Scale peut non seulement établir des associations, mais également en supprimer. Par example, si le client établit une association to-one entre une entité Order(orderId=5000,customer_customerId='IBM') et une entité et une instance Customer('ALFKI'), il doit dissocier l'entité Order(orderId=5000,customer_customerId='IBM') et l'entité de l'instance Customer actuellement associée.

Si l'une des instances d'entité spécifiées dans la demande UpdateLink est introuvable, le service de données REST renvoie une réponse 404 (Introuvable).

Si l'URI de la demande UpdateLink spécifie une association inexistante, le service de données REST renvoie une réponse 404 (Introuvable) pour indiquer que le lien est introuvable.

Si l'URI spécifié dans la charge de la demande UpdateLink ne se résout pas dans la même entité ou dans la même clé que celle spécifiée dans l'URI, si elle existe, le service de données REST d'eXtreme Scale renvoie une réponse 400 (bad request).

Si la charge de la demande UpdateLink contient plusieurs liens, le service de données REST n'analyse que le premier. Les autres liens sont ignorés.

Pour plus de détails sur la demande UpdateLink, voir MSDN Library: UpdateLink Request.

Voici un exemple de demande UpdateLink. Dans cet exemple, nous mettons à jour la relation client de l'entité Order(orderId=5000,customer_customerId='IBM') et de Customer('IBM') à Customer('IBM').

A faire : L'exemple précédent n'est fourni qu'à des fins d'illustration. Toutes les associations étant généralement des associations de clé pour une grille partitionnée, le lien ne peut pas être modifié.

AtomPub

- Méthode : PUT
- URI de la demande : http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(101)/$links/customer
- En-tête de la demande : Content-Type: application/xml
- Charge de la demande :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<uri>
 http://host:1000/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')
</uri>
```

- Charge de la réponse : Aucune
- Code de la réponse : 204 Pas de contenu

JSON

- Méthode : PUT
- URI de la demande : http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/$links/customer
- En-tête de la demande : Content-Type: application/xml
- Charge de la demande : {"uri": "http://host:1000/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')"}
- Charge de la réponse : Aucune
- Code de la réponse : 204 Pas de contenu

Suppression de demandes avec les services de données REST

Le service de données REST de WebSphere eXtreme Scale peut supprimer des entités, des valeurs de propriété et des liens.

Suppression d'une entité

La demande DeleteEntity peut supprimer une entité eXtreme Scale du service de données REST.

Si la suppression en cascade est définie pour une relation avec l'entité à supprimer, le service de données REST d'eXtreme Scale supprimera la ou les entités associées. Pour plus de détails sur la demande DeleteEntity, voir MSDN Library: DeleteEntity Request.

La demande DeleteEntity ci-après supprime le client dont la clé est 'IBM'.

- Méthode : DELETE
- URI de la demande : http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')
- Charge de la demande : Aucune
- Charge de la réponse : Aucune
- Code de la réponse : 204 Pas de contenu

Suppression d'une valeur de propriété

La demande DeleteValue définit une valeur null pour une propriété d'entité eXtreme Scale.

Toute propriété d'entité eXtreme Scale peut être définie comme null avec une demande DeleteValue. Pour affecter la valeur null à une propriété, vérifiez les points suivants :

- Pour tout type de numéro de primitive et son encapsuleur, BigInteger ou BigDecimal, la valeur de la propriété est 0.
- Pour Boolean ou le type booléen, la valeur de la propriété est false.
- Pour char ou le type Caractère, la valeur de la propriété correspond au caractère #X1 (NIL).
- Pour le type énumération, la valeur de la propriété correspond à la valeur de l'énumération avec l'ordinal 0.
- Pour tous les autres types, la valeur de la propriété est null.

Toutefois, une telle demande de suppression peut être rejetée par le système dorsal de la base de données si, par exemple, la propriété n'accepte pas la valeur null dans la base de données. Dans ce cas, le service de données REST renvoie une réponse 500 (Erreur de serveur interne). Pour plus de détails sur la demande DeleteValue, voir MSDN Library: DeleteValue Request.

Voici un exemple de demande DeleteValue. Dans cet exemple, nous affectons la valeur null au nom de contact Customer('IBM').

- Méthode : DELETE
- URI de la demande : http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName
- Charge de la demande : Aucune

- Charge de la réponse : Aucune
- Code de la réponse : 204 Pas de contenu

Suppression d'un lien

La demande DeleteLink permet de supprimer une association entre deux instances d'entité eXtreme Scale. L'association peut être une relation to-one ou to-many. Toutefois, une telle demande de suppression peut être rejetée par le système dorsal de la base de données si, par exemple, la contrainte de clé externe est définie. Dans ce cas, le service de données REST renvoie une réponse 500 (Erreur de serveur interne). Pour plus de détails sur la demande DeleteLink, voir MSDN Library: DeleteLink Request.

La demande DeleteLink ci-après supprime l'association entre Order(101) et le client associé.

- Méthode : DELETE
- URI de la demande : http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(101)/$links/customer
- Charge de la demande : Aucune
- Charge de la réponse : Aucune
- Code de la réponse : 204 Pas de contenu

Contrôle d'accès simultanés

Le service de données REST d'eXtreme Scale utilise un modèle de verrouillage optimiste à l'aide d'en-têtes HTTP natifs : If-Match, If-None-Match et ETag. Ces en-têtes sont envoyés dans les messages de demande et de réponse pour relayer les informations de version d'une entité du serveur au client et du client au serveur.

Pour plus de détails sur les accès simultanés optimistes, voir MSDN Library: Optimistic Concurrency (ADO.NET).

Le service de données REST d'eXtreme Scale permet les accès simultanés optimiste pour une entité si un attribut de version est défini dans le schéma de cette entité. Une propriété de version peut être définie dans le schéma d'entité par une annotation @Version pour les classes Java ou un attribut <version/> pour les entités définies à l'aide d'un fichier XML de descripteur d'entité. Le service de données REST d'eXtreme Scale propage automatiquement vers le client la valeur de la propriété de version dans l'en-tête Etag des réponses d'entité uniques à l'aide d'un attribut m:etag (réponses XML d'entités multiples) ou etag (réponses JSON d'entités multiples).

Pour plus de détails sur la définition d'un schéma d'entité eXtreme Scale, voir «Définition d'un schéma d'entité», à la page 62.

Chapitre 5. Programmation avec les API système et les plug-in

Un plug-in est un composant qui fournit des fonctions aux composants connectables, par exemple ObjectGrid et BackingMap. Pour utiliser au mieux eXtreme Scale en tant qu'espace de traitement de grille de données ou base de données en mémoire, vous devez soigneusement déterminer la meilleure façon d'optimiser les performances à l'aide des plug-in disponibles.

Introduction aux plug-in

Un plug-in WebSphere eXtreme Scale est un composant qui offre un certain type de fonction aux composants connectables intégrant ObjectGrid et BackingMap. WebSphere eXtreme Scale offre plusieurs points de connexion pour permettre aux applications et aux fournisseurs de cache de s'intégrer aux divers magasins de données, aux autres API client et d'améliorer les performances générales du cache. Le produit est livré avec plusieurs plug-in par défaut préconstruits mais vous pouvez aussi créer des plug-in personnalisés avec l'application.

Tous les plug-in sont des classes concrètes qui implémentent une ou plusieurs interfaces de plug-in eXtreme Scale. Ces classes sont ensuite instanciées et appelées par l'ObjectGrid aux moments appropriés. L'ObjectGrid et les mappes de sauvegarde permettent l'enregistrement des plug-in personnalisés.

Plug-in ObjectGrid

Les plug-in suivants sont disponibles pour une instance ObjectGrid :

- **TransactionCallback** : un plug-in TransactionCallback offre des événements de cycle de vie de transaction.
- **ObjectGridEventListener** : un plug-in ObjectGridEventListener offre des événements de cycle de vie ObjectGrid pour l'ObjectGrid, les fragments et les transactions.
- **SubjectSource, ObjectGridAuthorization, SubjectValidation** : eXtreme Scale offre différents points de contact de sécurité pour permettre l'intégration des mécanismes d'authentification personnalisés à eXtreme Scale. (côté serveur uniquement)
- **MapAuthorization** (côté serveur uniquement)
- **JPATxCallback** (côté serveur uniquement)
- **Sous-classes de JPATxCallback**

Exigences communes de plug-in ObjectGrid

L'ObjectGrid instancie et initialise les instances de plug-in à l'aide de conventionsJavaBeans. Toutes les implémentations de plug-in précédentes présentent les exigences suivantes :

- La classe de plug-in doit être une classe publique de niveau supérieur.
- La classe de plug-in doit fournit un constructeur public non argument.
- La classe de plug-in doit être disponible dans le chemin d'accès aux classes pour les serveurs et les clients (le cas échéant).

- Les attributs doivent être définis à l'aide des méthodes de propriété de style JavaBeans.
- Les plug-in, sauf indication contraire, sont enregistrés avant l'initialisation de l'ObjectGrid et ne peuvent pas être modifiés après cette initialisation.

Plug-in BackingMap

Les plug-in suivants sont disponibles pour une instance BackingMap :

- **Evictor** : ce plug-in fournit un mécanisme par défaut pour expulser les entrées du cache et un plug-in permettant la création d'expulseurs personnalisés.
- **ObjectTransformer** : ce plug-in permet de sérialiser, désérialiser et copier des objets du cache.
- **OptimisticCallback** : ce plug-in permet de personnaliser les opérations de vérification et de comparaison des versions des objets du cache lorsqu'on utilise la stratégie de verrouillage optimiste.
- **MapEventListener** : ce plug-in fournit les notifications de rappel et les modifications importantes de l'état du cache qui se produisent pour une instance BackingMap.
- **Indexing** : utilisez cette fonction représentée par le plug-in MapIndex pour créer un ou plusieurs index sur une mappe BackingMap pour prendre en charge l'accès aux données non clé.
- **Loader** : ce plug-in sur une mappe ObjectGrid peut être utilisé comme cache pour les données généralement conservées dans un stockage de persistance sur le même système ou un autre système (côté serveur uniquement)

Cycle de vie des plug-in

La plupart des plug-in comportent les méthodes initialize et destroy ou des méthodes équivalentes en plus des méthodes pour lesquelles ils sont conçus. Ces méthodes spécialisées pour chaque plug-in sont disponibles pour être appelées à des points fonctionnels déterminés. Les méthodes initialize et destroy définissent le cycle de vie des plug-in qui sont contrôlés par leurs objets « propriétaires ». Un objet propriétaire est l'objet qui utilise le plug-in donné. Un propriétaire peut être un client de grille, un serveur ou une mappe de sauvegarde.

Lorsque les objets propriétaires sont initialisés, ils appellent la méthode initialize des plug-in qu'ils possèdent. Lors du cycle de destruction des objets propriétaires, la méthode destroy des plug-in est également appelée en conséquence. Pour plus de détails sur les spécificités des méthodes initialize et destroy, ainsi que d'autres méthodes adaptées à chaque plug-in, reportez-vous aux rubriques correspondant à chaque plug-in.

A titre d'exemple, prenons un environnement réparti. Les ObjectGrids côté client et les ObjectGrids côté serveur comportent leurs propres plug-in. Le cycle de vie d'une ObjectGrid côté client et, par conséquent, de ses instances de plug-in est indépendant de l'ObjectGrid côté serveur et des instances de plug-in associées.

Dans une telle topologie répartie, mettons que vous avez une ObjectGrid myGrid définie dans le fichier objectGrid.xml et configurée avec un ObjectGridEventListener personnalisé myObjectGridEventListener. Le fichier objectGridDeployment.xml définit la règle de déploiement pour l'ObjectGrid myGrid. Les fichiers objectGrid.xml et objectGridDeployment.xml sont utilisés pour démarrer les serveurs conteneurs. Lors du démarrage du serveur de conteneur, l'instance de l'ObjectGrid myGrid côté serveur est initialisée et la méthode initialize

de l'instance myObjectGridEventListener possédées par l'instance myObjectGrid est appelée. Après le démarrage du serveur de conteneur, l'application peut se connecter à l'instance de l'ObjectGrid myGrid et obtenir une instance côté client.

Lors de l'obtention de l'instance d'ObjectGrid myGrid côté client, cette dernière traverse son propre cycle d'initialisation et appelle l'instance myObjectGridEventListener côté client. Cette instance myObjectGridEventListener côté client est indépendante de l'instance myObjectGridEventListener côté serveur. Son cycle de vie est contrôlée par son propriétaire qui est l'instance d'ObjectGrid myGrid côté client.

Si votre application se déconnecte ou détruit l'instance ObjectGrid myGrid côté client, la méthode destroy de l'instance myObjectGridEventListener côté client possédée est appelée automatiquement. Toutefois, cela n'a pas d'impact sur l'instance myObjectGridEventListener côté serveur. La méthode destroy de l'instance myObjectGridEventListener côté serveur n'est appelée que lors du cycle de destruction de l'instance d'ObjectGrid myGrid côté serveur lors de l'arrêt d'un serveur de conteneur. Cela signifie que lors de l'arrêt d'un serveur de conteneur, les instances ObjectGrid contenues sont détruites et la méthode destroy de tous les plug-in possédés est appelée.

Bien que l'exemple précédent s'applique spécifiquement au cas d'un client et d'une instance de serveur d'une ObjectGrid, le propriétaire d'un plug-in peut également être une mappe de sauvegarde et vous devez prendre soin de déterminer vos configurations pour les plug-in que vous pouvez écrire en fonction de ces remarques sur le cycle de vie.

Plug-in d'expulsion d'objets du cache

WebSphere eXtreme Scale fournit un mécanisme par défaut pour expulser les entrées du cache et un plug-in permettant la création d'expulseurs personnalisés. Un expulseur contrôle l'appartenance des entrées dans chaque instance de BackingMap. L'expulseur par défaut utilise une stratégie d'expulsion basée sur la durée de vie pour chaque instance de BackingMap. Si vous fournissez un mécanisme d'expulsion connectable, il utilise généralement une stratégie d'expulsion basée sur le nombre d'entrées au lieu de la durée de vie.

Propriété TimeToLive

Un expulseur basé sur la durée de vie est créé par défaut avec chaque mappe de sauvegarde. L'expulseur par défaut supprime les entrées en se basant sur un concept de durée de vie. Ce comportement est défini par l'attribut ttlType, dont les types sont les suivants.

Aucun

Spécifie que les entrées n'expirent jamais et par conséquent ne sont jamais supprimées de la mappe.

Heure de création

Spécifie que les entrées sont expulsées en fonction de la date et de l'heure auxquelles elles ont été créées.

Si vous utilisez l'attribut ttlType CREATION_TIME, l'expulseur expulsera une entrée lorsque elle aura été créée à la date et à l'heure spécifiées par la valeur de l'attribut, laquelle est définie en millisecondes dans la

configuration de l'application. Si vous définissez à 10 secondes la valeur de l'attribut TimeToLive, l'entrée est automatiquement expulsée dix secondes après son insertion.

Il est important de faire attention lors de la définition de cette valeur. Cet expulseur s'avère utile dans les cas de quantités raisonnablement élevées d'ajouts au cache utilisés pendant un temps donné. Avec cette stratégie, tout ce qui est créé est supprimé à la fin de la durée définie.

Le type TTL CREATION_TIME est utile dans des scénarios où l'on doit actualiser des cours boursiers toutes les 20 minutes, voire moins. Supposons qu'une application Web récupère les cours de la bourse, sans que l'obtention des cours les plus récents soit cruciale. Dans ce cas, les cours sont mis en cache dans un ObjectGrid pendant 20 minutes. Après 20 minutes, les entrées de la mappe ObjectGrid expirent et sont expulsées. Toutes les vingt minutes environ, la mappe ObjectGrid utilise le plug-in Loadere pour actualiser ses données avec des données neuves provenant de la base. Celle-ci est actualisée toutes les 20 minutes avec les cours les plus récents.

Heure du dernier accès

Spécifie que les entrées sont expulsées en fonction de l'heure de leur dernier accès, qu'elles aient été lues ou actualisées.

Heure de la dernière modification

Spécifie que les entrées sont expulsées en fonction de l'heure de leur dernière modification.

Si vous utilisez l'attribut ttlType LAST_ACCESS_TIME ou LAST_UPDATE_TIME, donnez une valeur plus faible à TimeToLive que si vous utilisiez l'attribut CREATION_TIME, car les entrées d'attribut TimeToLive sont réinitialisées lors de chaque accès. En d'autres termes, si l'attribut TimeToLive est égal à 15 et qu'une entrée a existé 14 secondes mais qu'on y accède, elle n'expire pas pendant les 15 prochaines secondes. Si vous définissez TimeToLive sur une valeur relativement élevée, il est possible que de nombreuses entrées ne soient pas expulsées. Cependant, si vous définissez la valeur sur quelque chose comme 15 secondes, les entrées avec peu d'accès risquent d'être supprimées.

Le type TTL LAST_ACCESS_TIME ou LAST_UPDATE_TIME sont utiles dans des scénarios où l'on doit conserver les données de session d'un client dans une mappe ObjectGrid. Les données de session doivent être détruites si le client ne les utilise pas pendant un certain laps de temps. Par exemple, après 30 minutes d'inactivité du client. Dans ce cas, l'utilisation d'un type TTL LAST_ACCESS_TIME ou LAST_UPDATE_TIME avec un attribut TimeToLive défini à 30 minutes convient tout à fait à cette application.

L'exemple suivant crée une mappe de sauvegarde, définit son attribut ttlType d'expulseur par défaut et définit sa propriété TimeToLive.

```
ObjectGrid objGrid = new ObjectGrid;
BackingMap bMap = objGrid.defineMap("SomeMap");
bMap.setTtlEvictorType(TTLType.LAST_ACCESS_TIME);
bMap.setTimeToLive(1800);
```

La plupart des paramètres d'expulseur doivent être définis avant l'initialisation d'ObjectGrid.

Vous pouvez également écrire vos propres expulseurs : pour plus d'informations, voir les informations concernant l'écriture d'un expulseur personnalisé dans *Guide de programmation*.

Expulseurs facultatifs

L'expulseur TTL par défaut utilise une stratégie d'expulsion basée sur le temps, et le nombre d'entrées dans la mappe de sauvegarde n'a pas d'effet sur le délai d'expiration d'une entrée. Vous pouvez utiliser un expulseur connectable facultatif pour expulser des entrées en fonction du nombre d'entrées existantes au lieu du temps.

Les expulseurs connectables optionnels qui suivent fournissent des algorithmes communément utilisés pour décider quelles entrées expulser dès qu'une mappe de sauvegarde dépasse une certaine taille :

- l'expulseur LRUEvictor utilise un algorithme moins récent (LRU)
- l'expulseur LFUEvictor utilise un algorithme moins fréquent (LFU)

La mappe de sauvegarde informe un expulseur quand des entrées sont créées, modifiées ou supprimées dans une transaction. La mappe suit ces entrées et choisit quand en expulser une ou plusieurs.

Une mappe de sauvegarde ne dispose pas d'informations de configuration pour une taille maximale. A la place, les propriétés de l'expulseur sont définies pour contrôler le comportement de ce dernier. Le LRUEvictor et le LFUEvictor ont une taille maximale utilisée pour déclencher l'expulsion des entrées quand une certaine taille est dépassée. Comme l'expulseur TTL, il est possible que les expulseurs LRU et LFU n'expulsent pas immédiatement une entrée quand le nombre maximal d'entrées est atteint, pour minimiser l'impact sur les performances.

Si l'algorithme d'expulsion LRU ou LFU se révèle inadéquat pour une application particulière, vous pouvez écrire vos propres expulseurs pour créer votre stratégie d'expulsion.

Expulsion basée sur la mémoire

Important : L'expulsion basée sur la mémoire est prise en charge uniquement par Java Platform, Enterprise Edition Version 5 ou ultérieure.

Tous les expulseurs pré-intégrés prennent en charge l'expulsion basée sur la mémoire, qui peut être activée sur l'interface BackingMap en définissant l'attribut evictionTriggers sur `MEMORY_USAGE_THRESHOLD`. Pour en savoir plus sur la définition de l'attribut evictionTriggers sur BackingMap, voir les informations concernant l'interface BackingMap et le Fichier XML de descripteur d'ObjectGrid dans le *Guide de programmation*.

L'expulsion basée sur la mémoire repose sur un seuil d'utilisation d'un segment de mémoire. Quand cette expulsion est activée sur BackingMap et que cette mappe de sauvegarde dispose d'un expulseur pré-intégré, le seuil d'utilisation est défini sur un pourcentage par défaut de la mémoire totale, si le seuil n'a pas été défini auparavant.

Lors de l'utilisation de l'expulsion basée sur la mémoire, vous devez configurer le seuil de récupération de place sur la même valeur que l'utilisation du segment de mémoire cible. Par exemple, si le seuil de l'expulsion basée sur la mémoire est à

50 % et que le seuil de récupération de place est à la valeur par défaut de 70 %, alors l'utilisation du segment de mémoire peut aller jusqu'à 70 %. Cette augmentation de l'utilisation du segment de mémoire se produit car l'expulsion basée sur la mémoire n'est déclenchée qu'après un cycle de récupération de place.

L'expulsion basée sur la mémoire utilisée par WebSphere eXtreme Scale est sensible au comportement de l'algorithme en cours d'utilisation. Le meilleur algorithme pour l'expulsion basée sur la mémoire est le collecteur de débit par défaut d'IBM. Les algorithmes de génération de récupération de place peuvent avoir des comportements indésirables et il est déconseillé de les utiliser avec l'expulsion basée sur la mémoire.

Pour changer le pourcentage du seuil d'utilisation, définissez la propriété memoryThresholdPercentage sur les fichiers de propriété de conteneur et de serveur pour les processus serveur eXtreme Scale.

Si pendant l'exécution, l'utilisation de la mémoire dépasse le seuil cible, les expulseurs basés sur la mémoire commencent à expulser des entrées et essaient de garder l'utilisation de la mémoire sous le seuil d'utilisation cible. Cependant, il n'existe aucune garantie que la vitesse d'expulsion soit assez grande pour éviter une erreur de dépassement de mémoire si l'exécution du système continue à consommer rapidement de la mémoire.

Expulseur TimeToLive (TTL)

WebSphere eXtreme Scale fournit un mécanisme par défaut pour expulser les entrées du cache et un plug-in permettant la création d'expulseurs personnalisés. Un expulseur contrôle l'appartenance des entrées dans chaque instance de BackingMap.

Activation de l'expulseur basé sur la durée de vie à l'aide d'un programme

Les expulseurs basés sur la durée de vie sont associés à des instances de BackingMap. L'expulseur par défaut utilise une stratégie d'expulsion basée sur la durée de vie pour chaque instance de BackingMap. Si vous fournissez un mécanisme d'expulsion connectable, il utilise généralement une stratégie d'expulsion basée sur le nombre d'entrées au lieu de la durée de vie.

Le fragment de code suivant utilise l'interface BackingMap pour définir un délai d'expiration de 10 minutes pour chaque entrée après sa création.

```
expulseur basé sur la durée de vie à l'aide d'un programme
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.TTLType;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid" );
BackingMap bm = og.defineMap( "myMap" );
bm.setTtlEvictorType( TTLType.CREATION_TIME );
bm.setTimeToLive( 600 );
```

L'argument de la méthode setTimeToLive est 600 car cela indique la valeur de durée de vie en secondes. Le code précédent doit être exécuté avant l'appel de la méthode d'initialisation sur l'instance ObjectGrid. Ces attributs BackingMap ne peuvent pas être modifiés une fois que l'instance ObjectGrid est initialisée. Une fois

que le code est exécuté, une entrée insérée dans BackingMap myMap possède un délai d'expiration. Une fois ce délai expiré, l'expulseur basé sur la durée de vie supprime l'entrée.

Pour définir un délai d'expiration à la date du dernier accès plus 10 minutes, changez l'argument qui est passé à la méthode setTtlEvictorType de TTLType.CREATION_TIME à TTLType.LAST_ACCESS_TIME. Avec cette valeur, le délai d'expiration équivaut à la date du dernier accès plus 10 minutes. Quand une entrée vient d'être créée, sa date de dernier accès est sa date de création. Pour baser l'heure de l'expiration sur la dernière *modification* au lieu de se contenter du dernier *accès* (qu'une modification ait eu effectivement lieu ou non), remplacez le paramètre TTLType.LAST_ACCESS_TIME par TTLType.LAST_UPDATE_TIME.

Lorsque vous utilisez TTLType.LAST_ACCESS_TIME ou TTLType.LAST_UPDATE_TIME, vous pouvez utiliser les interfaces ObjectMap et JavaMap pour que celles-ci substituent leur propre valeur de durée de vie de la mappe de sauvegarde. Ce mécanisme autorise une application à utiliser une valeur de durée de vie différente pour chaque entrée créée. Partons du principe que le fragment de code précédent définit l'attribut ttlType sur LAST_ACCESS_TIME et la valeur de durée de vie sur 10 minutes. Une application peut substituer sa propre valeur de durée de vie pour chacune des entrées en exécutant le code suivant avant de créer ou de modifier l'entrée :

```
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.ObjectMap;
Session session = og.getSession();
ObjectMap om = session.getMap( "myMap" );
int oldTimeToLive1 = om.setTimeToLive( 1800 );
om.insert("key1", "value1" );
int oldTimeToLive2 = om.setTimeToLive( 1200 );
om.insert("key2", "value2" );
```

Dans le fragment de code précédent, l'entrée avec la clé key1 a un délai d'expiration basé sur la date d'insertion plus 30 minutes en raison de l'appel de la méthode setTimeToLive(1800) sur l'instance ObjectMap. La variable oldTimeToLive1 est définie sur 600 car la valeur de durée de vie de BackingMap est utilisée comme variable par défaut si la méthode setTimeToLive n'a pas été appelée précédemment sur l'instance ObjectMap.

L'entrée avec la clé key2 a un délai d'expiration basé sur la date d'insertion plus 20 minutes en raison de l'appel de la méthode setTimeToLive(1200) sur l'instance ObjectMap. La variable oldTimeToLive2 est définie sur 1800 car la valeur de durée de vie de l'appel précédent de la méthode ObjectMap.setTimeToLive définit cette valeur sur 1800.

Les exemples précédents montrent l'insertion de deux entrées de mappe dans la mappe myMap pour les clés key1 et key2. Plus tard, l'application peut toujours mettre à jour ces entrées de mappe tout en conservant les valeurs de durée de vie utilisées au moment de l'insertion pour chaque entrée de mappe. L'exemple suivant illustre comment conserver les valeurs de durée de vie, en utilisant une constante définie dans l'interface ObjectMap:

```
Session session = og.getSession();
ObjectMap om = session.getMap( "myMap" );
om.setTimeToLive( ObjectMap.USE_DEFAULT );
session.begin();
om.update("key1", "updated value1" );
om.update("key2", "updated value2" );
om.insert("key3", "value3" );
session.commit();
```

Comme la valeur spéciale ObjectMap.USE_DEFAULT est utilisée sur l'appel de la méthode setTimeToLive, la clé key1 conserve sa valeur de 1800 secondes et la clé key2 conserve la sienne de 1200 secondes, car ces valeurs on été utilisées quand les entrées de mappe ont été insérées par la transaction précédente.

L'exemple précédent montre également une nouvelle entrée de mappe pour l'insertion de la clé key3. Dans ce cas, la valeur spéciale USE_DEFAULT indique d'utiliser le paramètre par défaut de valeur de durée de vie pour cette mappe. La valeur par défaut est définie par l'attribut BackingMap de durée de vie. Voir Les attributs de l'interface BackingMap pour en savoir plus sur la manière dont est redéfini l'attribut de durée de vie sur l'instance BackingMap.

Voir la documentation d'API pour la méthode setTimeToLive sur les interfaces ObjectMap et JavaMap. La documentation explique qu'une exception IllegalStateException survient si la méthode BackingMap.getTtlEvictorType renvoie autre chose que la valeur TTLType.LAST_ACCESS_TIME ou TTLType.LAST_UPDATE_TIME. Les interfaces ObjectMap et JavaMap ne peuvent substituer la valeur de durée de vie que lorsqu'on utilise le paramètre LAST_ACCESS_TIME ou TTLType.LAST_UPDATE_TIME pour le type d'expulseur TTL. La méthode setTimeToLive ne peut pas être utilisée pour substituer sa propre valeur de durée de vie lorsqu'on utilise les paramètres CREATION_TIME ou NONE.

Activation de l'expulseur basé sur la durée de vie à l'aide d'une configuration XML

Au lieu d'utiliser l'interface BackingMap pour définir à l'aide d'un programme les attributs BackingMap qui doivent être utilisés par l'expulseur basé sur la durée de vie, vous pouvez utiliser un fichier XML pour configurer chaque instance de BackingMap. Le code suivant démontre comment définir ces attributs pour trois mappes BackingMap différentes :

Activation de l'expulseur basé sur la durée de vie par XML

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
    <objectGrid name="grid1">
        <backingMap name="map1" ttlEvictorType="NONE" />
        <backingMap name="map2" ttlEvictorType="LAST_ACCESS_TIME|LAST_UPDATE_TIME"
     timeToLive="1800" />
        <backingMap name="map3" ttlEvictorType="CREATION_TIME"
     timeToLive="1200" />
    </objectgrid>
</objectGrids>
```

L'exemple précédent montre que l'instance BackingMap map1 utilise le type d'expulseur basé sur la durée de vie NONE. L'instance BackingMap map2 utilise un type d'expulseur basé sur la durée de vie LAST_ACCESS_TIME TTL ou TTLType.LAST_UPDATE_TIME (ne spécifiez que l'un ou l'autre de ces deux paramètres) et sa durée de vie est de 1800 secondes(30 minutes). L'instance BackingMap map3 est définie pour utiliser le type d'expulseur basé sur la durée de vie CREATION_TIME et a une durée de vie de 1200 secondes (20 minutes).

Connecter un expulseur

Les expulseurs étant associés à des mappes de sauvegarde, l'interface BackingMap permet de spécifier l'expulseur à connecter.

Expulseurs optionnels

L'expulseur TTL par défaut utilise une stratégie d'expulsion basée sur le temps, et le nombre d'entrées dans la mappe de sauvegarde n'a pas d'effet sur le délai d'expiration d'une entrée. Au lieu de vous baser sur le temps, vous pouvez connecter un expulseur optionnel pour expulser des entrées en fonction du nombre d'entrées existantes.

Les expulseurs optionnels fournissent des algorithmes communément utilisés pour décider quelles entrées expulser dès qu'une mappe de sauvegarde dépasse une certaine taille.

- L'expulseur LRUEvictor utilise un algorithme déterminant les entrées les moins récemment utilisées (LRU).
- L'expulseur LFUEvictor utilise un algorithme déterminant les entrées les moins fréquemment utilisées(LFU).

La mappe de sauvegarde informe un expulseur quand des entrées sont créées, modifiées ou supprimées d'une transaction. La mappe de sauvegarde suit ces entrées et choisit quand expulser de l'instance BackingMap une ou plusieurs de ces entrées.

Une instance BackingMap ne dispose pas d'informations de configuration pour une taille maximale. A la place, les propriétés d'expulseur sont définies pour contrôler le comportement de ce dernier. Le LRUEvictor et le LFUEvictor ont une taille maximale utilisée pour déclencher l'expulsion des entrées quand une certaine taille est dépassée. Comme l'expulseur TTL, il est possible que les expulseurs LRU et LFU n'expulsent pas immédiatement une entrée quand le nombre maximal d'entrées est atteint, pour minimiser l'impact sur les performances.

Si l'algorithme d'expulsion LRU ou LFU se révèle inadéquat pour une application particulière, vous pouvez écrire vos propres expulseurs pour créer votre stratégie d'expulsion.

Connecter des expulseurs optionnels

Pour ajouter des expulseurs optionnels à la configuration BackingMap, vous pouvez utiliser une configuration par programmation ou une configuration XML.

Connecter un expulseur par programmation

Les expulseurs étant associés à des mappes de sauvegarde, l'interface BackingMap permet de spécifier l'expulseur à connecter. Le fragment de code qui suit montre comment spécifier un expulseur LRUEvictor pour la mappe de sauvegarde map1 et un expulseur LFUEvictor pour l'instance BackingMap map2 :

connecter un expulseur par programmation

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor;
import com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
```

```
ObjectGrid og = ogManager.createObjectGrid( "grid" );
BackingMap bm = og.defineMap( "map1" );
LRUEvictor evictor = new LRUEvictor();
evictor.setMaxSize(1000);
evictor.setSleepTime( 15 );
evictor.setNumberOfLRUQueues( 53 );
bm.setEvictor(evictor);
bm = og.defineMap("map2");
LFUEvictor evictor2 = new LFUEvictor();
evictor2.setMaxSize(2000);
evictor2.setSleepTime( 15 );
evictor2.setNumberOfHeaps( 211 );
bm.setEvictor(evictor2);
```

Le fragment de code montrait un expulseur LRUEvictor utilisé pour la mappe de sauvegarde map1 avec un nombre maximum d'entrées d'environ 53 000 (53 * 1000). L'expulseur LFUEvictor était utilisé pour la mappe de sauvegarde map2 avec un nombre maximum d'entrées d'à peu près 422 000 (211*2000). Les deux expulseurs LRU et LFU comportent une propriété SleepTime qui indique au bout de combien de temps l'expulseur doit s'éveiller pour vérifier s'il n'y a pas lieu d'expulser des entrées. Cette durée d'inactivité est spécifiée en secondes. 15 secondes est un bon compromis entre l'impact sur les performances et le souci d'empêcher la mappe de trop grossir. L'objectif est d'utiliser le délai d'inactivité le plus long possible sans que la mappe atteigne une taille excessive.

La méthode setNumberOfLRUQueues définit la propriété LRUEvictor qui indique combien de files d'attente LRU l'expulseur utilise pour gérer les informations LRU. Un ensemble de files d'attente est en effet utilisé pour éviter que chaque entrée ne conserve les informations LRU dans la même file. Cette approche permet d'améliorer les performances en réduisant le nombre d'entrées de mappe qui ont besoin de synchronisation dans le même objet queue. Augmenter le nombre de files d'attente est un bon moyen de réduire l'impact possible de l'expulseur LRU sur les performances. 10 % du nombre maximum d'entrées est un bon point de départ pour définir le nombre des files d'attente. En règle générale, il vaut mieux utiliser un nombre premier. La méthode setMaxSize indique combien d'entrées sont autorisées dans chaque file d'attente. Lorsqu'une file atteint son nombre maximum d'entrées, la ou les entrées les moins récemment utilisées dans cette file sont expulsées lors de la prochaine vérification par l'expulseur.

La méthode setNumberOfHeaps définit la propriété LFUEvictor qui indique combien d'objets binaires de segments de mémoire LFUEvictor utilise pour gérer les informations LFU. Ici aussi, un ensemble est utilisé afin d'améliorer les performances. Et ici aussi, 10 % du nombre d'entrées et un bon point de départ et il vaut mieux utiliser un nombre premier. La méthode setMaxSize indique combien d'entrées sont autorisées dans chaque segment de mémoire. Lorsqu'un segment de mémoire atteint son nombre maximum d'entrées, la ou les entrées les moins fréquemment utilisées dans ce segment sont expulsées lors de la prochaine vérification par l'expulseur.

Connecter un expulseur par configuration XML

Au lieu d'utiliser diverses API pour programmer la connexion d'un expulseur et pour définir ses propriétés, il est possible d'utiliser un fichier XML pour configurer chacune des mappes de sauvegarde :

connecter un expulseur par XML

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
```

```
    <objectGrid name="grid">
        <backingMap name="map1" ttlEvictorType="NONE" pluginCollectionRef="LRU" />
        <backingMap name="map2" ttlEvictorType="NONE" pluginCollectionRef="LFU" />
    </objectGrid>
</objectGrids>
<backingMapPluginCollections>
    <backingMapPlugincollection id="LRU">
        <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
            <property name="maxSize" type="int" value="1000" description="set max size
       for each LRU queue" />
            <property name="sleepTime" type="int" value="15" description="evictor
       thread sleep time" />
            <property name="numberOfLRUQueues" type="int" value="53" description="set number
       of LRU queues" />
        </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="LFU">
        <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor">
            <property name="maxSize" type="int" value="2000" description="set max size for each LFU heap" />
            <property name="sleepTime" type="int" value="15" description="evictor thread sleep time" />
            <property name="numberOfHeaps" type="int" value="211" description="set number of LFU heaps" />
        </bean>
    </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Expulsion basée sur la mémoire

Tous les expulseurs pré-intégrés prennent en charge l'expulsion basée sur la mémoire, laquelle peut être activée dans l'interface BackingMap en donnant à l'attribut evictionTriggers la valeur MEMORY_USAGE_THRESHOLD. Pour plus d'informations sur la configuration de l'attribut evictionTriggers dans BackingMap, voir les informations de référence concernant l'interface BackingMap et la configuration d'eXtreme Scale.

L'expulsion basée sur la mémoire repose sur un seuil d'utilisation d'un segment de mémoire. Quand cette expulsion est activée dans la mappe de sauvegarde et que cette dernière dispose d'un expulseur pré-intégré, le seuil d'utilisation est défini en un pourcentage par défaut de la mémoire totale, si le seuil n'a pas été défini auparavant.

Pour modifier le pourcentage du seuil d'utilisation, définissez la propriété memoryThresholdPercentage dans les fichiers de propriétés de conteneur et de serveur pour les processus de serveur eXtreme Scale. Pour définir le seuil d'utilisation cible dans un processus de client d'eXtreme Scale, vous pouvez utiliser le MemoryPoolMXBean. Voir aussi : le fichier containerServer.props et Démarrage de processus serveur eXtreme Scale.

Si, pendant l'exécution, l'utilisation de la mémoire dépasse le seuil cible, les expulseurs basés sur la mémoire commencent à expulser des entrées et essaient de conserver l'utilisation de la mémoire en dessous du seuil cible. Cependant, il n'existe aucune garantie que la vitesse d'expulsion soit assez grande pour éviter une erreur de dépassement de mémoire si l'exécution du système continue à rapidement consommer de la mémoire.

Ecriture d'un expulseur personnalisé

WebSphere eXtreme Scale permet l'écriture d'une implémentation d'expulseur personnalisé.

Vous devez créer un expulseur personnalisé qui implémente l'interface d'expulseur et suit les conventions courantes de plug-in eXtreme Scale. L'interface se présente comme suit :

```
public interface Evictor
{
    void initialize(BackingMap map, EvictionEventCallback callback);
```

```
    void activate();
    void apply(LogSequence sequence);
    void deactivate();
    void destroy();
}
```

- La méthode initialize est appelée lors de l'initialisation de l'objet BackingMap. Cette méthode initialise un plug-in Evictor avec une référence à la mappe de sauvegarde et une référence à un objet qui implémente l'interface com.ibm.websphere.objectgrid.plugins.EvictionEventCallback.
- La méthode activate est appelée pour activer l'expulseur. Après l'appel de cette méthode, l'expulseur peut utiliser l'interface EvictionEventCallback pour expulser les entrées de mappe. Si l'expulseur tente d'utiliser l'interface EvictionEventCallback pour expulser les entrées de mappe avant l'appel de la méthode activate, une exception IllegalStateException est émise.
- La méthode apply est appelée lors de la validation de transactions qui accèdent à une ou plusieurs entrées de la mappe de sauvegarde. La méthode apply reçoit une référence à un objet qui implémente l'interface com.ibm.websphere.objectgrid.plugins.LogSequence. L'interface LogSequence permet à un plug-in Evictor de déterminer les entrées de mappe de sauvegarde créées, modifiées ou supprimées par la transaction. Un expulseur utilise ces informations pour décider quelles entrées doivent être expulsées et quand l'expulsion doit avoir lieu.
- La méthode deactivate est appelée pour désactiver l'expulseur. Après l'appel de cette méthode, l'expulseur doit cesser d'utiliser l'interface EvictionEventCallback pour expulser les entrées de mappe. Si l'expulseur utilise l'interface EvictionEventcallback après l'appel de cette méthode, une exception IllegalStateException est émise.
- La méthode destroy est appelée lors de la destruction de la mappe de sauvegarde. Cette méthode permet à un expulseur de mettre fin à toute unité d'exécution éventuellement créée.

L'interface EvictionEventCallback comporte les méthodes suivantes :

```
public interface EvictionEventCallback
{
    void evictMapEntries(List evictorDataList) throws ObjectGridException;
    void evictEntries(List keysToEvictList) throws ObjectGridException;
    void setEvictorData(Object key, Object data);
    Object getEvictorData(Object key);
}
```

Les méthodes EvictionEventCallback sont utilisées par un plug-in Evictor pour rappeler l'infrastructure eXtreme Scale comme suit :

- La méthode setEvictorData est utilisée par un expulseur pour demander l'infrastructure de stockage et pour associer certains objets evictor créés avec l'entrée indiquée par l'argument de clé. Les données sont spécifiques à l'expulseur et sont déterminées par les informations requises par l'expulseur pour implémenter l'algorithme qu'il utilise. Par exemple, dans un algorithme le moins fréquemment utilisé, l'expulseur maintient un compte dans l'objet de données pour suivre la fréquence d'appel de la méthode apply avec un objet LogElement qui se réfère à une entrée pour une clé donnée.
- La méthode getEvictorData est utilisée par un expulseur pour extraire les données transmises à la méthode setEvictorData lors d'une invocation de la méthode apply antérieure. Si les données de l'expulseur pour l'argument de clé spécifié ne sont pas trouvées, un objet spécial KEY_NOT_FOUND défini sur l'interface EvictorCallback est renvoyé.

- La méthode evictMapEntries est utilisée par un expulseur pour demander l'expulsion d'une ou de plusieurs entrées de mappe. Chaque objet dans le paramètre evictorDataList doit implémenter l'interface com.ibm.websphere.objectgrid.plugins.EvictorData. De plus, la même instance EvictorData transmise à la méthode setEvictorData doit être incluse dans le paramètre de liste de données de l'expulseur pour cette méthode. La méthode getKey de l'interface EvictorData est utilisée pour déterminer l'entrée de mappe à expulser. L'entrée de mappe est expulsée si l'entrée de cache contient la même instance EvictorData inclue dans la liste de données de l'expulseur pour cette entrée de cache.
- La méthode evictEntries est utilisée par un expulseur pour demander l'expulsion d'une ou plusieurs entrées de mappe. Cette méthode est uniquement utilisée si l'objet transmis à la méthode setEvictorData n'implémente pas l'interface com.ibm.websphere.objectgrid.plugins.EvictorData.

A la fin d'une transaction, eXtreme Scale appelle la méthode apply de l'interface Evictor. Tous les verrous de transaction obtenus à la fin de la transaction ne sont plus maintenus. Eventuellement, plusieurs unités d'exécution peuvent appeler la méthode apply simultanément et chaque unité d'exécution peut effectuer sa propre transaction. Etant donné que les verrous de transaction sot déjà libérés à la fin de la transaction, la méthode apply doit offrir sa propre synchronisation pour faire en sorte que la méthode apply soit sécurisée pour les unités d'exécution.

L'implémentation de l'interface EvictorData et l'utilisation de la méthode evictMapEntries à la place de la méthode evictEntries se justifient par la fermeture d'une fenêtre de temps potentielle. Examinez la séquence d'événements suivante :

1. La transaction 1 se termine et appelle la méthode apply avec un objet LogSequence qui supprime l'entrée de mappe pour la clé 1.
2. La transaction 2 se termine et appelle la méthode apply avec un objet LogSequence qui insère une nouvelle entrée de mappe pour la clé 1. En d'autres termes, la transaction 2 recrée l'entrée de mappe supprimée par la transaction 1.

L'expulseur s'exécutant de façon asynchrone depuis les unités d'exécution qui exécutent les transactions, il est possible que lorsque l'expulseur décide d'expulser la clé 1, il expulse l'entrée de mappe qui existait avant la fin de la transaction 1 ou l'entrée de mappe qui a été recréée par la transaction 2. Pour éviter les fenêtres de temps et s'assurer de la version de l'entrée de mappe de la clé 1 que l'expulseur a voulu expulser, implémentez l'interface EvictorData par l'objet transmis à la méthode setEvictorData. Utilisez la même instance EvictorData pour la durée de vie d'une entrée de mappe. Lorsque cette entrée de mappe est supprimées et est ensuite recréée par une autre transaction, l'expulseur doit utiliser une nouvelle instance de l'implémentation EvictorData. En utilisant l'implémentation EvictorData et la méthode evictMapEntries, l'expulseur peut vérifier que l'entrée de mappe est expulsée à condition uniquement que l'entrée de cache associée à l'entrée de mappe contienne l'instance EvictorData correcte.

Les interfaces Evictor et EvictonEventCallback permettent à une application de connecter un expulseur qui implémente un algorithme défini par l'utilisateur pour l'expulsion. Le fragment de code ci-dessous illustre comment implémenter la méthode initialize de l'interface Evictor :

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.EvictionEventCallback;
import com.ibm.websphere.objectgrid.plugins.Evictor;
import com.ibm.websphere.objectgrid.plugins.LogElement;
import com.ibm.websphere.objectgrid.plugins.LogSequence;
import java.util.LinkedList;
```

```
// Variables d'instance
private BackingMap bm;
private EvictionEventCallback evictorCallback;
private LinkedList queue;
private Thread evictorThread;
public void initialize(BackingMap map, EvictionEventCallback callback)
{
    bm = map;
    evictorCallback = callback;
    queue = new LinkedList();
    // générez une unité d'exécution d'expulseur
    evictorThread = new Thread( this );
    String threadName = "MyEvictorForMap-" + bm.getName();
    evictorThread.setName( threadName );
    evictorThread.start();
}
```

Le code précédent enregistre les références à la mappe et aux objets de rappel dans des variables d'instance de façon à les rendre disponibles pour les méthodes apply et destroy. Dans cet exemple, une liste liée est créée qui est utilisée comme file d'attente premier entré dernier sorti pour l'implémentation d'un algorithme LRU. Une unité d'exécution est générée et une référence à l'unité d'exécution est conservée en tant que variable d'instance. En conservant cette référence, la méthode destroy peut interrompre et mettre fin à l'unité d'exécution générée.

En ignorant les exigences de synchronisation afin de sécuriser les unités d'exécution de code, le fragment de code ci-dessous illustre comment la méthode apply de l'interface Evictor peut être implémentée :

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.EvictionEventCallback;
import com.ibm.websphere.objectgrid.plugins.Evictor;
import com.ibm.websphere.objectgrid.plugins.EvictorData;
import com.ibm.websphere.objectgrid.plugins.LogElement;
import com.ibm.websphere.objectgrid.plugins.LogSequence;

public void apply(LogSequence sequence)
{
    Iterator iter = sequence.getAllChanges();
    while(iter.hasNext())
    {
        LogElement elem = (LogElement)iter.next();
        Object key = elem.getKey();
        LogElement.Type type = elem.getType();
        if ( type == LogElement.INSERT )
        {
            // n'insérez pas le traitement ici par l'ajout en tête de file d'attente LRU.
            EvictorData data = new EvictorData(key);
            evictorCallback.setEvictorData(key, data);
            queue.addFirst( data );
        }
        else if ( type == LogElement.UPDATE || type == LogElement.FETCH || type == LogElement.TOUCH )
        {
            // ne mettez pas à jour le traitement ici en déplaçant l'objet EvictorData au début
            // de la file d'attente.
            EvictorData data = evictorCallback.getEvictorData(key);
            queue.remove(data);
            queue.addFirst(data);
        }
        else if ( type == LogElement.DELETE || type == LogElement.EVICT )
        {
            // ne supprimez pas le traitement ici en supprimant l'objet EvictorData
            // de la file d'attente.
            EvictorData data = evictorCallback.getEvictorData(key);
            if ( data == EvictionEventCallback.KEY_NOT_FOUND )
            {
                // Hypothèse ici que votre unité d'exécution d'expulseur asynchrone
                // a expulsé l'entrée de mappe avant que cette unité d'exécution
                // traite la demande de LogElement. Vous devez probablement
                // n'effectuer aucune action lorsque cela se produit.
            }
            else
            {
                // La clé a été trouvée. Traitez les données de l'expulseur.
                if ( data != null )
                {
                    // Ignorez les valeurs null renvoyées par la méthode remove car l'unité
                    // d'exécution de l'expulseur générée l'a peut-être déjà supprimé de la file d'attente.
                    // Mais nous avons besoin de ce code au cas où cet objet LogElement n'a pas été
                    // provoqué par l'unité d'exécution.
                    queue.remove( data );
                }
                else
                {
```

```
                    // En fonction de l'écriture de l'expulseur, cette possibilité peut ne pas
                    // exister ou peut signaler une erreur de l'expulseur en raison d'une
                    // logique de synchronisation d'unités d'exécution inadaptée.
                }
            }
        }
    }
}
```

Le traitement des insertions de la méthode apply permet généralement de gérer la création d'un objet de données d'expulseur transmis à la méthode setEvictorData de l'interface EvictionEventCallback. Etant donné que cet expulseur illustre une implémentation LRU, les données de l'expulseur sont également ajoutées au début de la file d'attente qui a été créée par la méthode initialize. Le traitement des mises à jour de la méthode apply permet généralement de mettre à jour l'objet de données de l'expulseur créé par une invocation antérieure de la méthode apply (par exemple par le traitement d'une insertion de la méthode apply). Etant donné que cet expulseur illustre une implémentation LRU, il doit déplacer l'objet EvictorData de sa position actuelle dans la file d'attente au début de la file d'attente. L'unité d'exécution d'expulseur générée supprime le dernier objet EvictorData dans la file d'attente car ce dernier élément représente l'entrée la moins récemment utilisée. Il est supposé que l'objet EvictorData est associé à une méthode getKey de sorte que l'unité d'exécution d'expulseur connaît les clés des entrées à expulser. Gardez à l'esprit que cet exemple ignore les exigences de synchronisation afin de sécuriser l'unité d'exécution de code. Un expulseur personnalisé réel est plus complexe car il doit gérer la synchronisation et les goulots d'étranglement qui résultent des points de synchronisation.

Les fragments de code ci-dessous illustrent la méthode destroy et la méthode run de l'unité d'exécution générée par la méthode initialize :

```
// La méthode destroy interrompt simplement l'unité d'exécution générée par la méthode initialize.
public void destroy() {
    evictorThread.interrupt();
}

// Voici la méthode run de l'unité d'exécution générée par la méthode initialize.
public void run()
{
    // Exécutez en boucle jusqu'à ce que la méthode destroy interrompe cette unité d'exécution.
    boolean continueToRun = true;
    while ( continueToRun )
    {
        try
        {
            // Mettez en veille quelques instants avant analyse de la file d'attente.
            // sleepTime est une bonne pratique pour la définition
            // de la propriété d'un expulseur.
            Thread.sleep( sleepTime );
            int queueSize = queue.size();
            // Expulsez les entrées si la taille de la file d'attente dépasse
            // la taille maximale autorisée. Apparemment, la taille maximale
            // correspond à une autre propriété de l'expulseur.
            int numToEvict = queueSize - maxSize;
            if ( numToEvict > 0 )
            {
                // Supprimez de la fin de la file d'attente car la fin correspond
                // à l'entrée la moins récemment utilisée.
                List evictList = new ArrayList( numToEvict );
                while( queueSize > ivMaxSize )
                {
                    EvictorData data = null;
                    try
                    {
                        EvictorData data = (EvictorData) queue.removeLast();
                        evictList.add( data );
                        queueSize = queue.size();
                    }
                    catch ( NoSuchElementException nse )
                    {
                        // La file d'attente est vide.
                        queueSize = 0;
                    }
                }
                // Demandez l'expulsion si la liste de clés n'est pas vide.
                if ( ! evictList.isEmpty() )
                {
                    evictorCallback.evictMapEntries( evictList );
```

```
                }
            }
        }
        catch (InterruptedException e)
        {
            continueToRun = false;
        }
    } // mettez fin à la boucle while
} // mettez fin à la méthode run.
```

Interface RollBackEvictor facultative

L'interface com.ibm.websphere.objectgrid.plugins.RollbackEvictor peut être implémentée en option par un plug-in Evictor. En implémentant cette interface, un expulseur peut être appelé non seulement lors de la validation des transactions mais aussi lors de l'annulation des transactions.

```
public interface RollbackEvictor
{
    void rollingBack( LogSequence ls );
}
```

La méthode apply est appelée uniquement si une transaction est validée. Si une transaction est annulée et si l'interface RollbackEvictor est implémentée par l'expulseur, la méthode rollingBack est appelée. Si l'interface RollbackEvictor n'est pas implémentée et si la transaction s'annule, la méthode apply et la méthode rollingBack ne sont pas appelées.

Meilleures pratiques pour les performances de l'expulseur de plug-in

Si vous utilisez les expulseurs de plug-in, ces derniers ne sont actifs que si vous les créez et les associez à une mappe de sauvegarde. Les meilleures pratiques suivantes contribuent à renforcer les performances pour les expulseurs les moins fréquemment utilisés (LFU) et les expulseurs les moins récemment utilisés (LRU).

Expulseur le moins fréquemment utilisé (LFU)

Le concept d'un expulseur LFU consiste à supprimer les entrées d'une mappe qui ne sont pas utilisées fréquemment. Les entrées de la mappe sont réparties sur une quantité définie de segments de mémoire binaires. Lorsqu'une entrée de cache est de plus en plus sollicitée, elle est placée plus haut dans le segment de mémoire. Lorsque l'expulseur tente d'effectuer un ensemble d'expulsions, il supprime uniquement les entrées de cache situées en dessous d'un point spécifique du segment de mémoire binaire. Par conséquent, les entrées les moins fréquemment utilisées sont expulsées.

Expulseur le moins récemment utilisé (LRU)

L'expulseur LRU suit les mêmes concepts que l'expulseur LFU à quelques différentes près. Il diffère principalement en ce qu'il utilise une file d'attente premier entré, premier sorti et non un ensemble de segments de mémoire binaires. A chaque accès à une entrée de cache, il remonte en tête de la file d'attente. Par conséquent, le début de la file d'attente contient les entrées de mappe les plus récemment utilisées et la fin de la file d'attente contient les entrées de mappe les moins récemment utilisés. Par exemple, l'entrée de cache A est utilisée 50 fois et l'entrée de cache B est utilisée une seule fois juste après l'entrée de cache A. Dans ce cas, l'entrée de cache B est en tête de la file d'attente car elle a été utilisée en dernier alors que l'entrée de cache A se trouve à la fin de la file d'attente. L'expulseur LRU expulse les entrées de cache situées à la fin de la file d'attente car ce sont les entrées les moins récemment utilisées.

Propriétés LFU et LRU et meilleures pratiques pour améliorer les performances

Nombre de segments de mémoire

Lors de l'utilisation de l'expulseur LFU, toutes les entrées de cache d'une mappe donnée sont organisées en fonction du nombre de segments de mémoire spécifié, ce qui contribue à améliorer considérablement les performances et à réduire toutes les expulsions résultant de la synchronisation sur un segment de mémoire binaire qui contient toutes les organisations de la mappe. Une quantité supérieure de segments de mémoire accélère le temps requis pour réorganiser les segments de mémoire car chaque segment de mémoire dispose de moins d'entrées. Le nombre de segments de mémoire doit représenter 10 % du nombre total d'entrées de votre mappe de base.

Nombre de files d'attente

Lors de l'utilisation de l'expulseur LRU, toutes les entrées de cache d'une mappe donnée sont organisées en fonction du nombre de files d'attente LRU, ce qui contribue à améliorer considérablement les performances et à réduire toutes les expulsions résultant de la synchronisation sur une file d'attente qui contient toutes les organisations de la mappe. Le nombre de files d'attente doit représenter 10 % du nombre total d'entrées de votre mappe de base.

Propriété MaxSize

Lorsqu'un expulseur LFU ou LRU commence à expulser des entrées, il utilise la propriété MaxSize pour déterminer le nombre de segments de mémoire binaires ou de files d'attente LRU à expulser. Par exemple, supposons que les segments de mémoire ou files d'attente comportent 10 entrées de mappe dans chaque file d'attente de mappe. Si la propriété MaxSize est définie sur 7, l'expulseur expulse 3 entrées de chaque segment de mémoire ou de file d'attente pour ramener le nombre de segments de mémoire ou de files d'attente à 7. L'expulseur expulse uniquement les entrées de mappe d'un segment de mémoire ou d'une file d'attente lorsque ce dernier ou cette dernière contient un nombre d'éléments supérieur à la valeur de la propriété MaxSize. La valeur de la propriété MaxSize doit représenter 70 % de la taille de votre segment de mémoire ou de votre file d'attente. Pour cet exemple, la valeur est définie sur 7. Vous pouvez calculer la taille approximative de chaque segment de mémoire ou file d'attente en divisant le nombre d'entrées de la mappe de base par le nombre de segments de mémoire ou files d'attente utilisés.

Propriété SleepTime

Un expulseur ne supprime pas constamment les entrées de la mappe. En revanche, il est en veille pendant un intervalle de temps déterminé et ne vérifie la mappe que toutes les n secondes où n se réfère à la propriété SleepTime. Cette propriété influe également sur les performances de manière positive : une analyse d'expulsion trop fréquente réduit les performances en raison de l'utilisation des ressources nécessaires à cette opération. Toutefois, une utilisation trop rare de l'expulseur peut entraîner la présence d'entrées superflues dans la mappe. Une mappe saturée d'entrées inutiles peut nuire à la configuration requise pour la mémoire et aux ressources de traitement nécessaires pour la mappe. Il est recommandé de définir un intervalle d'analyse d'expulsion de quinze secondes pour la plupart des mappes. Si l'écriture de la mappe est trop fréquente et si le taux de transactions est trop élevé, pensez à réduire cette valeur. En revanche, si l'accès à la mappe n'est pas fréquent, vous pouvez augmenter la valeur.

Exemple

L'exemple suivant définit une mappe, crée un expulseur LFU, définit les propriétés de l'expulseur et définit la mappe pour utiliser l'expulseur :

```
//Utilisez ObjectGridManager pour créer/obtenir la grille d'objets. Voir
// la section ObjectGridManager
ObjectGrid objGrid = ObjectGridManager.create.....
BackingMap bMap = objGrid.defineMap("SomeMap");

//Définissez les propriétés sur la base de 50 000 entrées de mappe
LFUEvictor someEvictor = new LFUEvictor();
someEvictor.setNumberOfHeaps(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

L'utilisation de l'expulseur LRU s'apparente à celle d'un expulseur LFU. Voici un exemple :

```
ObjectGrid objGrid = new ObjectGrid;
BackingMap bMap = objGrid.defineMap("SomeMap");

//Définissez les propriétés sur la base de 50 000 entrées de mappe
LRUEvictor someEvictor = new LRUEvictor();
someEvictor.setNumberOfLRUQueues(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

Notez que seulement deux lignes sont différentes de l'exemple LFUEvictor.

Plug-in de transformation des objets mis en cache

La transformation des objets mis en cache peut améliorer les performances de ce dernier. Vous pouvez utiliser le plug-in ObjectTransformer dans lorsque votre processeur est très sollicité : jusqu'à 60-70 % du temps processeur total est consacré à sérialiser et à copier des entrées. L'implémentation du plug-in ObjectTransformer permet de sérialiser et de désérialiser grâce à votre propre implémentation. Vous pouvez utiliser un plug-in CollisionArbiter pour définir la manière dont doivent être gérées sur vos domaines les collisions entre modifications.

Développement d'arbitres personnalisés pour la réplication multi-maître

Des collisions entre des modifications peuvent se produire s'il est possible à des enregistrements identiques d'être modifiés simultanément en deux endroits différents. Dans une topologie de réplication multi-maître, les collisions sont automatiquement détectées par les domaines. Lorsqu'un domaine détecte une collision, il fait appel à un arbitre. Normalement, les collisions sont résolues à l'aide de l'arbitre par défaut. Mais une application peut très bien fournir un arbitre personnalisé.

Pourquoi et quand exécuter cette tâche

Si un domaine reçoit une entrée répliquée qui entre en collision avec un enregistrement local, l'arbitre par défaut tranche en faveur des modifications du domaine dont le nom vient en premier par ordre alphabétique. Supposons par exemple que les domaines A et B génèrent un conflit pour un enregistrement, dans ce cas, la modification du domaine B sera ignorée. Le domaine A conserve sa

version et l'enregistrement dans le domaine B est modifié de manière à correspondre à celui du domaine A. Les noms de domaines sont convertis en majuscules pour les besoins de la comparaison.

Une autre possibilité est que la topologie de réplication multi-maître fasse appel à un plug-in de collisions personnalisé pour décider de l'issue à donner. Les instructions qui suivent expliquent comment développer un arbitre personnalisé et configurer son utilisation par une topologie de réplication multi-maître.

Procédure

1. Développez un arbitre personnalisé et intégrez-le à votre application.

   La classe doit implémenter l'interface

   `com.ibm.websphere.objectgrid.revision.CollisionArbiter`

   Pour décider de l'issue d'une collision, un plug-in peut faire trois choix : il peut choisir la copie locale, la copie distante ou de fournir une version révisée de l'entrée. Un domaine fournit à un arbitre personnalisé les informations suivantes :

   - la version externe de l'enregistrement
   - la version locale de l'enregistrement
   - un objet Session qui doit servir à créer la version révisée de l'entrée en collision

   La méthode du plug-in retourne un objet indiquant sa décision. La méthode invoquée par le domaine pour faire appel au plug-in doit retourner true ou false, false signifiant que la collision doit être ignorée, que la version locale demeure inchangée et qu'eXtreme Scale oubliera jusqu'à la version externe. En revanche, la méthode retourne une valeur true si elle a utilisé la session fournie pour créer une nouvelle version fusionnée de l'enregistrement, réconciliant les modifications.
2. Dans le fichier `objectgrid.xml`, spécifiez d'utiliser le plug-in d'arbitre personnalisé.

   L'ID doit être "CollisionArbiter".

```
<dgc:objectGrid name="revisionGrid" txTimeout="10">
     <dgc:bean className="com.you.your_application.
      CustomArbiter" id="CollisionArbiter">
      <dgc:property name="property" type="java.lang.String"
      value="propertyValue"/>
     </dgc:bean>
</dgc:objectGrid>
```

Plug-in ObjectTransformer

Le plug-in ObjectTransformer permet de sérialiser, désérialiser et copier des objets du cache afin d'améliorer les performances.

Si vous constatez des problèmes de performances dans l'utilisation des processeurs, ajoutez à chaque mappe un plug-in ObjectTransformer. Sans ce plug-in ObjectTransformer, jusqu'à 60-70 % du temps processeur sera consacré à la sérialisation et à la copie des entrées.

Utilité

Le plug-in ObjectTransformer permet à vos applications de fournir des méthodes personnalisées pour les opérations suivantes :

- sérialisation ou désérialisation de la clé d'une entrée

- sérialisation ou désérialisation de la valeur d'une entrée
- copie de la clé ou de la valeur d'une entrée

Si aucun plug-in ObjectTransformer n'est fourni, vous devrez savoir sérialiser vous-mêmes les clés et les valeurs car l'ObjectGrid utilise une séquence de sérialisation/désérialisation pour copier les objets. Cette méthode est onéreuse ; c'est pourquoi il convient d'utiliser un plug-in ObjectTransformer lorsque les performances sont en jeu. La copie ne se produit que lorsqu'une application recherche pour la première fois un objet dans une transaction. Vous pouvez éviter la copie en donnant au mode copy de la mappe la valeur NO_COPY ou réduisant la copie en donnant à ce mode la valeur COPY_ON_READ. Optimisez l'opération de copie lorsque l'application a besoin d'en effectuer une en fournissant une méthode personnalisée de copie dans ce plug-in. Ce plug-in peut faire tomber le temps système consacré à la copie de 65–70 % à 2-3 % du temps processeur total.

La première fois, les implémentations par défaut des méthodes copyKey et copyValue tentent d'utiliser la méthode clone si cette méthode est fournie. Si aucune implémentation de clone n'est fournie, par défaut, l'implémentation passe à la sérialisation.

La sérialisation des objets est également utilisée directement lorsque eXtreme Scale s'exécute en mode réparti. LogSequence utilise le plug-in ObjectTransformer pour sérialiser les clés et les valeurs avant de transmettre les modifications aux homologues présents dans l'ObjectGrid. Vous devez prendre un certain nombre de précautions lorsque vous fournissez une méthode personnalisée de sérialisation au lieu d'utiliser la sérialisation pré-intégrée du kit de développement Java. La vérification des versions d'objets est en effet un problème complexe et vous risquez de rencontrer des problèmes de compatibilité de versions si vos méthodes personnalisées ne sont pas conçues pour gérer cette vérification.

La liste qui suit décrit comment eXtreme Scale s'y prend pour sérialiser les clés et les valeurs :

- Si un plug-in ObjectTransformer personnalisé est écrit et connecté, eXtreme Scale appelle les méthodes présentes dans l'interface ObjectTransformer pour sérialiser les clés et les valeurs et pour obtenir des copies de ces clés et de ces valeurs.
- S'il n'est pas fait usage d'un plug-in ObjectTransformer personnalisé, eXtreme Scale sérialise et désérialise les valeurs conformément à la méthode par défaut. Si c'est le Plug-in par défaut qui est utilisé, chaque objet est implémenté comme externalisable ou comme sérialisable.
  - Si l'objet prend en charge l'interface Externalizable, c'est la méthode writeExternal qui est appelée. Les objets implémentés comme externalisables donnent de meilleures performances.
  - Si l'objet ne prend pas en charge l'interface Externalizable et qu'il implémente l'interface Serializable, il est enregistré à l'aide de la méthode ObjectOutputStream.

Utiliser l'interface ObjectTransformer

Un objet ObjectTransformer doit implémenter l'interface ObjectTransformer et se conformer aux conventions communes des plug-in ObjectGrid.

Comme toujours, deux approches sont possibles pour ajouter un objet ObjectTransformer à la configuration BackingMap : la configuration par programmation et la configuration XML.

Configuration par XML du plug-in ObjectTransformer

Supposons que le nom de la classe de l'implémentation d'ObjectTransformer soit com.company.org.MyObjectTransformer. Cette classe implémente l'interface ObjectTransformer. Le code XML suivant permet de configurer une implémentation d'ObjectTransformer :

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
    xmlns="http://ibm.com/ws/objectgrid/config">
    <objectGrids>
        <objectGrid name="myGrid">
     <backingMap name="myMap" pluginCollectionRef="myMap" />
        </objectGrid>
    </objectGrids>

    <backingMapPluginCollections>
 <backingMapPluginCollection id="myMap">
      <bean id="ObjectTransformer" className="com.company.org.MyObjectTransformer" />
 </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

Configuration par programmation du plug-in ObjectTransformer

Le fragment de code suivant crée l'objet ObjectTransformer personnalisé et l'ajoute à une BackingMap :

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
BackingMap backingMap = myGrid.getMap("myMap");
MyObjectTransformer myObjectTransformer = new MyObjectTransformer();
backingMap.setObjectTransformer(myObjectTransformer);
```

Scénarios d'utilisation d'ObjectTransformer

Vous pouvez utiliser le plug-in ObjectTransformer dans les situations suivantes :

- objet non sérialisable
- objet sérialisable mais nécessité d'améliorer les performances de la sérialisation
- copie de clés ou de valeurs

Dans l'exemple qui suit, l'ObjectGrid sert à stocker la classe Stock :

```
/**
* Objet Stock pour la démo ObjectGrid
*
*
*/
public class Stock implements Cloneable {
    String ticket;
    double price;
    String company;
    String description;
    int serialNumber;
    long lastTransactionTime;
    /**
    * @return retourne la description.
    */
    public String getDescription() {
        return description;
    }
    /**
    * @param description La description à définir.
    */
    public void setDescription(String description) {
        this.description = description;
    }
    /**
    * @return Retourne le lastTransactionTime.
    */
    public long getLastTransactionTime() {
        return lastTransactionTime;
```

```
    }
    /**
    * @param lastTransactionTime Le lastTransactionTime à définir.
    */
    public void setLastTransactionTime(long lastTransactionTime) {
        this.lastTransactionTime = lastTransactionTime;
    }
    /**
    * @return Retourne le prix.
    */
    public double getPrice() {
        return price;
    }
    /**
    * @param price Le prix à définir.
    */
    public void setPrice(double price) {
        this.price = price;
    }
    /**
    * @return Retourne le serialNumber.
    */
    public int getSerialNumber() {
        return serialNumber;
    }
    /**
    * @param serialNumber Le serialNumber à définir.
    */
    public void setSerialNumber(int serialNumber) {
        this.serialNumber = serialNumber;
    }
    /**
    * @return Retourne le ticket.
    */
    public String getTicket() {
        return ticket;
    }
    /**
    * @param ticket Le ticket à définir.
    */
    public void setTicket(String ticket) {
        this.ticket = ticket;
    }
    /**
    * @return Retourne la Company.
    */
    public String getCompany() {
        return company;
    }
    /**
    * @param company La Company à définir.
    */
    public void setCompany(String company) {
        this.company = company;
    }
    //clone
    public Object clone() throws CloneNotSupportedException
    {
        return super.clone();
    }
}
```

Vous pouvez écrire une classe ObjectTransformer personnalisée pour la classe Stock :

```
/**
* Implémentation personnalisée d'ObjectGrid ObjectTransformer pour l'objet Stock
*
*/
public class MyStockObjectTransformer implements ObjectTransformer {
/* (non-Javadoc)
* @see
* com.ibm.websphere.objectgrid.plugins.ObjectTransformer#serializeKey
* (java.lang.Object,
* java.io.ObjectOutputStream)
*/
public void serializeKey(Object key, ObjectOutputStream stream) throws IOException {
    String ticket= (String) key;
    stream.writeUTF(ticket);
}
```

```
/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#serializeValue(java.lang.Object,
java.io.ObjectOutputStream)
*/
public void serializeValue(Object value, ObjectOutputStream stream) throws IOException {
    Stock stock= (Stock) value;
    stream.writeUTF(stock.getTicket());
    stream.writeUTF(stock.getCompany());
    stream.writeUTF(stock.getDescription());
    stream.writeDouble(stock.getPrice());
    stream.writeLong(stock.getLastTransactionTime());
    stream.writeInt(stock.getSerialNumber());
}

/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#inflateKey(java.io.ObjectInputStream)
*/
public Object inflateKey(ObjectInputStream stream) throws IOException, ClassNotFoundException {
    String ticket=stream.readUTF();
    return ticket;
}

/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#inflateValue(java.io.ObjectInputStream)
*/

public Object inflateValue(ObjectInputStream stream) throws IOException, ClassNotFoundException {
    Stock stock=new Stock();
    stock.setTicket(stream.readUTF());
    stock.setCompany(stream.readUTF());
    stock.setDescription(stream.readUTF());
    stock.setPrice(stream.readDouble());
    stock.setLastTransactionTime(stream.readLong());
    stock.setSerialNumber(stream.readInt());
    return stock;
}

/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#copyValue(java.lang.Object)
*/
public Object copyValue(Object value) {
    Stock stock = (Stock) value;
    try {
        return stock.clone();
    }
    catch (CloneNotSupportedException e)
    {
        // affichage du message d'exception     }
}

/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#copyKey(java.lang.Object)
*/
public Object copyKey(Object key) {
    String ticket=(String) key;
    String ticketCopy= new String (ticket);
    return ticketCopy;
}
}
```

Vous pouvez alors connecter cette classe personnalisée MyStockObjectTransformer dans la BackingMap :

```
ObjectGridManager ogf=ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogf.getObjectGrid("NYSE");
BackingMap bm = og.defineMap("NYSEStocks");
MyStockObjectTransformer ot = new MyStockObjectTransformer();
bm.setObjectTransformer(ot);
```

Performances de sérialisation

WebSphere eXtreme Scale utilise plusieurs processus Java pour héberger les données. Ces processus sérialisent les données, c'est-à-dire les convertissent (sous la forme d'instances d'objets Java) en octets, puis si nécessaire de nouveau en objets pour permettre le déplacement des données entre les processus client et serveur. La conversion des paramètres de données est l'opération la plus coûteuse et doit être effectuée par le développeur d'applications lors de la conception du schéma, la configuration de la grille et l'interaction avec les API d'accès aux données.

La sérialisation Java par défaut et les routines de copie sont relativement lentes et peuvent consommer 60 à 70 % de la capacité du processeur dans une configuration classique. Les sections ci-dessous présentent des options d'amélioration des performances de la sérialisation.

Ecriture d'un ObjectTransformer pour chaque mappe de sauvegarde

Un ObjectTransformer peut être associé à une mappe de sauvegarde. Votre application peut comporter une classe qui implémente l'interface ObjectTransformer et offre des implémentations pour les opérations suivantes :

- Copie des valeurs
- Sérialisation et inflation des clés vers et à partir des flux
- Sérialisation et inflation des valeurs vers et à partir des flux

L'application ne doit pas copier des clés car celles-ci sont considérées comme étant non modifiables.

Pour plus d'informations, voir Plug-in de sérialisation et de copie d'objets mis en cache et Pratiques recommandées pour l'interface ObjectTransformer.

Remarque : L'interface ObjectTransformer est uniquement appelée lorsque la grille d'objets connaît les données en cours de transformation. Par exemple, les agents de l'API DataGrid sont utilisés, les agents et les données d'instance des agents ou les données renvoyées par l'agent doivent être optimisées à l'aide de techniques de sérialisation personnalisées. L'interface ObjectTransformer n'est pas appelée pour les agent de l'API DataGrid.

Utilisation d'entités

Lors de l'utilisation de l'API EntityManager avec les entités, la grille ne stocke pas les objets d'entité directement dans les mappes de sauvegarde. L'API EntityManager convertit l'objet d'entité en objets de bloc de données. Pour plus d'informations, voir Pour plus d'informations, consultez la rubrique relative à l'utilisation d'un chargeur avec les mappes d'entité et les blocs de données dans le *Guide de programmation*. Les mappes d'entité sont automatiquement associées à un ObjectTransformer hautement optimisé. Dès que l'API ObjectMap ou EntityManager est utilisée pour interagir avec les mappes d'entité, l'entité ObjectTransformer est appelée.

Sérialisation personnalisée

Dans certains cas, les objets doivent être modifiés de façon à utiliser la sérialisation personnalisée, telle que l'implémentation de l'interface java.io.Externalizable ou l'implémentation des méthodes writeObject et readObject pour les classes qui mettent en oeuvre l'interface java.io.Serializable. Les techniques de sérialisation personnalisée doivent être employées lorsque les objets sont sérialisés à l'aide de mécanismes autres que les méthodes de l'API ObjectGrid ou EntityManager.

Par exemple, lorsque les objets ou entités sont stockés en tant que données d'instance dans un agent d'API DataGrid ou lorsque l'agent renvoie des objets ou des entités, ces objets ne sont transformés à l'aide d'un ObjectTransformer. Toutefois, l'agent fait automatiquement appel à l'ObjectTransformer lors de l'utilisation de l'interface `EntityMixin`. Pour plus de détails, voir Agents DataGrid et mappes basées sur les entités.

Tableaux d'octets

Lors de l'utilisation des API ObjectMap ou DataGrid, les objets de clé et de valeur sont sérialisés dès que le client interagit avec la grille et lorsque les objets sont répliqués. Pour limiter les frais liés à la sérialisation, utilisez des tableaux d'octets et non les objets Java. Le stockage en mémoire des tableaux d'octets revient nettement moins cher car le kit JDK doit rechercher moins d'objets lors de la récupération de place et les tableaux d'octets peuvent être agrandis à la demande. Les tableaux d'octets doivent être uniquement utilisés si vous ne devez pas accéder aux objets utilisant des requêtes ou des index. Les données étant stockées sous la forme d'octets, leur accès n'est autorisé qu'avec la clé correspondante.

WebSphere eXtreme Scale peut automatiquement stocker les données sous la forme de tableaux d'octets à l'aide l'option de configuration de mappes CopyMode.COPY_TO_BYTES ou ce mode de stockage peut être traité manuellement par le client. Cette option permet de stocker les données en mémoire de façon efficace et peut, à la demande, agrandir automatiquement les objets au sein du tableau d'octets pour une utilisation des requêtes et des index.

Meilleures pratiques concernant l'interface ObjectTransformer

L'interface ObjectTransformer envoie les rappels à l'application pour permettre l'implémentation personnalisée d'opérations courantes et coûteuses telles de la sérialisation ou la copie complète sur des objets.

Présentation

Pour des informations détaillées sur l'interface ObjectTransformer, voir «Plug-in ObjectTransformer», à la page 227. Du point de vue des performances et à partir des informations sur la méthode CopyMode (sous la rubrique des meilleures pratiques concernant la méthode CopyMode), eXtreme Scale copie clairement les valeurs dans tous les cas, sauf lorsque le mode NO_COPY est utilisé. Le mécanisme de copie par défaut utilisé dans eXtreme Scale est la sérialisation, qui est connue pour être une opération coûteuse. L'interface ObjectTransformer est utilisée dans cette situation. L'interface ObjectTransformer envoie des rappels à l'application pour permettre l'implémentation personnalisée d'opérations courantes et coûteuses, telles que la sérialisation d'objets et la copie complète sur des objets.

Une application permet l'implémentation de l'interface ObjectTransformer sur une mappe. eXtreme Scale délègue ensuite aux méthodes de cet objet et dépend de l'application pour fournir une version optimisée de chaque méthode de l'interface. Voici l'interface ObjectTransformer :

```
public interface ObjectTransformer {
    void serializeKey(Object key, ObjectOutputStream stream) throws IOException;
    void serializeValue(Object value, ObjectOutputStream stream) throws IOException;
    Object inflateKey(ObjectInputStream stream) throws IOException, ClassNotFoundException;
    Object inflateValue(ObjectInputStream stream) throws IOException, ClassNotFoundException;
    Object copyValue(Object value);
    Object copyKey(Object key);
}
```

Vous pouvez associer l'interface ObjectTransformer à une BackingMap en utilisant le code de l'exemple suivant :

```
ObjectGrid g = ...;
BackingMap bm = g.defineMap("PERSON");
MyObjectTransformer ot = new MyObjectTransformer();
bm.setObjectTransformer(ot);
```

Ajustement de la sérialisation d'objets et inflation

La sérialisation d'objets est généralement l'opération la plus coûteuse en termes de performances, de même qu'eXtreme Scale, qui utilise le mécanisme de sérialisation par défaut si un plug-in ObjectTransformer n'est pas fourni par l'application. Soit l'application fournit des implémentations des objets sérialisables readObject et writeObject, soit les objets implémentent l'interface externalisable, ce qui est environ dix fois plus rapide. Si les objets dans la mappe ne peuvent pas être vérifiés, une application peut associer une interface ObjectTransformer avec ObjectMap. Les méthodes de sérialisation et d'inflation sont proposées pour permettre à l'application de fournir un code personnalisé permettant d'optimiser ces opérations, compte tenu de leur impact important sur les performances du système. La méthode de sérialisation permet de sérialiser l'objet sur le flux fourni. La méthode d'inflation fournit le flux d'entrée et attend que l'application crée l'objet, augmente la taille de ce dernier en utilisant les données du flux et le renvoie. Les implémentations des méthodes de sérialisation et d'inflation doivent être symétriques.

Ajustement des opérations de copie complète

Après qu'une application a reçu un objet d'une ObjectMap, eXtreme Scale effectue une copie complète de la valeur de l'objet afin de s'assurer que la copie de la mappe BaseMap maintient l'intégrité des données. L'application peut ensuite modifier la valeur de l'objet en toute sécurité. Lorsque la transaction est validée, la copie de la valeur de l'objet dans la mappe BaseMap est mise à jour avec la nouvelle valeur modifiée et l'application arrête d'utiliser la valeur. Vous pouvez copier de nouveau l'objet au moment de la validation afin de disposer d'une copie privée. Cependant, dans ce cas, le coût des performances de cette action est compensé par le fait que le programmeur de l'application ne peut pas utiliser cette valeur une fois la transaction validée. L'ObjectTransformer par défaut tente d'utiliser soit un clone soit une paire sérialisation-inflation pour générer une copie. La paire sérialisation-inflation est le pire scénario imaginable en termes de performances. Si le profilage révèle que la sérialisation et l'inflation présentent un problème pour votre application, écrivez une méthode de clonage appropriée pour créer une copie complète. Si vous ne pouvez pas modifier la classe, créez un plug-in ObjectTransformer personnalisé et implémentez des méthodes copyValue et copyKey plus efficaces.

Plug-in de vérification et de comparaison des versions des objets mis en cache

Le plug-in OptimisticCallback permet de personnaliser les opérations de vérification et de comparaison des versions des objets du cache lorsqu'on utilise la stratégie de verrouillage optimiste.

Il est possible de fournir un objet connectable de rappel optimiste qui implémente l'interface com.ibm.websphere.objectgrid.plugins.OptimisticCallback. Pour les mappes d'entités, un plug-in OptimisticCallback hautes performances est automatiquement configuré.

Utilité

L'interface OptimisticCallback permet de fournir des opérations de comparaison optimiste entre les valeurs d'une mappe. Un plug-in OptimisticCallback est nécessaire lorsqu'on utilise la stratégie de verrouillage optimiste. Le produit fournit

une implémentation par défaut d'OptimisticCallback. Mais, en principe, c'est à l'application de connecter sa propre implémentation de cette interface.

Implémentation par défaut

La structure eXtreme Scale fournit une implémentation par défaut de l'interface OptimisticCallback qui est utilisée si l'application ne connecte pas d'objet OptimisticCallback fourni par ses soins. L'implémentation par défaut retourne toujours la valeur spéciale NULL_OPTIMISTIC_VERSION comme objet version de la valeur et elle n'actualise jamais cet objet version. Cette action fait de la comparaison optimiste une fonction "no operation". Dans la plupart des cas, l'on ne souhaite pas que se produise une fonction "no operation" lorsqu'on utilise la stratégie de verrouillage optimiste. Vos applications doivent implémenter l'interface OptimisticCallback et connecter leurs propres implémentations OptimisticCallback afin de ne pas utiliser l'implémentation par défaut. Mais il existe au moins un scénario où l'implémentation OptimisticCallback fournie par défaut a toute son utilité. Prenons le cas de figure suivant :

- Un loader est connecté pour la mappe de sauvegarde.
- Le loader sait comment effectuer la comparaison optimiste sans l'assistance d'un plug-in OptimisticCallback.

Comment le loader peut-il effectuer une vérification optimiste des versions sans l'assistance d'un objet OptimisticCallback ? Le loader connaît l'objet de classe value et il sait quel champ de l'objet value est utilisé comme valeur optimiste de version. Supposons, par exemple, que l'interface suivante soit utilisée pour l'objet value de la mappe Employee :

```
public interface Employee
{
    // Sequential sequence number used for optimistic versioning.
    public long getSequenceNumber();
    public void setSequenceNumber(long newSequenceNumber);
    // Other get/set methods for other fields of Employee object.
}
```

Dans cet exemple, le loader sait qu'il peut utiliser la méthode getSequenceNumber pour obtenir la version actuelle d'un objet value Employee. Le loader incrémente la valeur retournée pour générer un nouveau numéro de version avant d'actualiser le stockage de persistance avec la nouvelle valeur d'Employee. Dans le cas d'un loader JDBC (Java Database Connectivity), c'est le numéro actuel de séquence dans la clause WHERE d'une instruction SQL UPDATE surqualifiée qui est utilisé et le loader utilise le nouveau numéro de séquence qui est généré pour définir la colonne des numéros de séquence. Autre possibilité : le loader recourt à une certaine fonction fournie en dorsal, qui actualise automatiquement une colonne masquée utilisables pour la vérification optimiste des versions.

Dans certains cas, une procédure stockée ou déclencheur peut être utilisée pour aider à maintenir une colonne contenant les informations de version. Si le loader utilise l'une de ces techniques pour maintenir des informations optimistes de version, l'application n'aura pas besoin de fournir d'implémentation d'OptimisticCallback. L'implémentation par défaut d'OptimisticCallback est utilisable dans ce scénario car le loader peut vérifier les versions de manière optimiste sans l'assistance d'un objet OptimisticCallback.

Implémentation par défaut des entités

Les entités sont stockées dans l'ObjectGrid à l'aide d'objets tuple. Le comportement de l'implémentation par défaut d'OptimisticCallback est semblable à celui des

mappes de non-entités. Mais le champ version dans l'entité est identifié à l'aide de l'annotation @Version ou de l'attribut version dans le fichier XML de descripteur d'entités.

L'attribut version peut être de l'un des types suivants : int, Integer, short, Short, long, Long ou java.sql.Timestamp. Une entité ne doit avoir qu'un seul attribut version défini. L'attribut version ne doit être défini que pendant la construction. Une fois l'entité persistante, la valeur de l'attribut version ne doit pas être modifiée.

Si un attribut version n'est pas configuré et que l'on utilise la stratégie de verrouillage optimiste, le tuple tout entier est versionné en utilisant son état tout entier, ce qui est beaucoup plus onéreux.

Dans l'exemple qui suit, l'entité Employee a un attribut long de version nommé SequenceNumber :

```
@Entity
public class Employee {
private long sequence;
    // Sequential sequence number used for optimistic versioning.
    @Version
    public long getSequenceNumber() {
        return sequence;
    }
    public void setSequenceNumber(long newSequenceNumber) {
        this.sequence = newSequenceNumber;
    }
    // Other get/set methods for other fields of Employee object.
}
```

Ecrire un plug-in OptimisticCallback

Un plug-in OptimisticCallback doit implémenter l'interface OptimisticCallback et se conformer aux conventions habituelles des plug-in ObjectGrid. Pour plus d'informations, voir l'interface OptimisticCallback dans la documentation de l'API .

La liste suivante décrit ou explique chacune des méthodes de l'interface OptimisticCallback :

NULL_OPTIMISTIC_VERSION

Cette valeur spéciale est retournée par la méthode getVersionedObjectForValue si l'implémentation d'OptimisticCallback ne requiert pas de vérification des versions. L'implémentation pré-intégrée de la classe com.ibm.websphere.objectgrid.plugins.builtins.NoVersioningOptimisticCallback utilise cette valeur car la vérification des versions est désactivée lorsque c'est cette implémentation qui est spécifiée pour le plug-in.

Méthode getVersionedObjectForValue

La méthode getVersionedObjectForValue peut retourner une copie de la valeur ou d'un attribut de la valeur qui peut être utilisée à des fins de vérification des versions. Cette méthode est appelée chaque fois qu'un objet est associé à une transaction. Lorsqu'aucune API Loader n'est connectée à une mappe de sauvegarde, cette dernière utilise cette valeur au moment de la validation afin d'effectuer une comparaison optimiste des versions. La comparaison optimiste des versions est utilisée par la mappe de sauvegarde pour s'assurer que la version n'a pas changé après le premier accès de cette transaction à l'entrée de mappe qui a

été modifiée par cette transaction. Si entre-temps une autre transaction a modifié la version de cette entrée de mappe, la comparaison échoue et la mappe de sauvegarde affiche une exception OptimisticCollisionException pour forcer la transaction à s'annuler. Si une API Loader est connectée, la mappe de sauvegarde n'utilise pas les informations de vérification optimiste des versions. Car c'est à l'API Loader d'effectuer cette comparaison optimiste et d'actualiser les informations de version quand c'est nécessaire. Normalement, l'API Loader obtient l'objet de version initial du LogElement transmis à sa méthode batchUpdate, laquelle méthode est appelée lorsqu'une opération de vidage se produit ou lorsqu'une transaction est validée.

Le code suivant montre comment l'objet EmployeeOptimisticCallbackImpl utilise l'implémentation :

```
public Object getVersionedObjectForValue(Object value)
{
    if (value == null)
    {
        return null;
    }
    else
    {
        Employee emp = (Employee) value;
        return new Long( emp.getSequenceNumber() );
    }
}
```

Comme le montre l'exemple ci-dessus, l'attribut sequenceNumber est retourné dans un objet java.lang.Long object, attendu par l'API Loader, ce qui implique que la personne qui a écrit l'API soit a écrit l'implémentation EmployeeOptimisticCallbackImpl, soit a collaboré étroitement avec celle qui a implémenté EmployeeOptimisticCallbackImpl, notamment en se mettant d'accord sur la valeur retournée par la méthode getVersionedObjectForValue. Le plug-in OptimisticCallback par défaut retourne la valeur spéciale NULL_OPTIMISTIC_VERSION en tant qu'objet version.

Méthode updateVersionedObjectForValue

Cette méthode est appelée chaque fois qu'une transaction a actualisé une valeur et que l'on a besoin d'un nouvel objet versionné. Si la méthode getVersionedObjectForValue retourne un attribut de la valeur, cette méthode actualise normalement la valeur de l'attribut avec un nouvel objet version. Si la méthode getVersionedObjectForValue retourne une copie de la valeur, en principe, cette méthode n'effectue aucune action. Avec cette méthode, le plug-in OptimisticCallback par défaut n'effectue aucune action car l'implémentation par défaut de getVersionedObjectForValue retourne la valeur spéciale NULL_OPTIMISTIC_VERSION en tant qu'objet version. L'exemple qui suit montre l'implémentation utilisée par l'objet EmployeeOptimisticCallbackImpl qui est utilisé dans la section OptimisticCallback :

```
public void updateVersionedObjectForValue(Object value)
{
    if ( value != null )
    {
        Employee emp = (Employee) value;
        long next = emp.getSequenceNumber() + 1;
        emp.updateSequenceNumber( next );
    }
}
```

Comme le montre l'exemple ci-dessus, l'attribut sequenceNumber s'incrémente de un afin que, lors du prochain appel de la méthode getVersionedObjectForValue, la valeur java.lang.Long qui est retournée soit une valeur de type long, égale à celle du numéro de séquence d'origine. Plus un, par exemple, sera la version suivante de cette instance d'Employee. Il ressort de cet exemple que la personne qui a écrit l'API Loader soit a écrit l'implémentation EmployeeOptimisticCallbackImpl, soit a collaboré étroitement avec celle qui a implémenté EmployeeOptimisticCallbackImpl.

Méthode serializeVersionedValue

Cette méthode écrit dans le flux spécifié la valeur versionnée. Selon l'implémentation, la valeur versionnée peut être utilisée pour identifier les collisions de mises à jour optimistes. Dans certaines implémentations, la valeur versionnée est une copie de la valeur d'origine. D'autres implémentations peuvent avoir un numéro de séquence ou un autre objet pur indiquer la version de la valeur. Comme l'implémentation effective est inconnue, cette méthode est fournie pour effectuer la sérialisation appropriée. L'implémentation par défaut appelle la méthode writeObject.

Méthode inflateVersionedValue

Cette méthode prend la version sérialisée de la valeur versionnée et elle retourne l'objet effectif de la valeur versionnée. Selon l'implémentation, la valeur versionnée peut être utilisée pour identifier les collisions de mises à jour optimistes. Dans certaines implémentations, la valeur versionnée est une copie de la valeur d'origine. D'autres implémentations peuvent avoir un numéro de séquence ou un autre objet pur indiquer la version de la valeur. Comme l'implémentation effective est inconnue, cette méthode est fournie pour effectuer la désérialisation appropriée. L'implémentation par défaut appelle la méthode readObject.

Utiliser l'objet OptimisticCallback fourni par l'application

Deux approches permettent d'ajouter un plug-in OptimisticCallback dans la configuration BackingMap : la configuration XML et la configuration par programmation.

Configuration par XML du plug-in OptimisticCallback

L'application peut utiliser un fichier XML pour connecter son objet OptimisticCallback :

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
    <objectGrid name="grid1">
        <backingMap name="employees" pluginCollectionRef="employees" lockStrategy="OPTIMISTIC" />
    </objectGrid>
</objectGrids>

<backingMapPluginCollections>
    <backingMapPluginCollection id="employees">
        <bean id="OptimisticCallback" className="com.xyz.EmployeeOptimisticCallbackImpl" />
    </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Configuration par programmation du plug-in OptimisticCallback

L'exemple qui suit montre comment une application peut par programmation connecter un objet OptimisticCallback pour la mappe de sauvegarde Employee dans l'instance ObjectGrid locale grid1 :

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid1" );
BackingMap bm = dg.defineMap("employees");
EmployeeOptimisticCallbackImpl cb = new EmployeeOptimisticCallbackImpl();
bm.setOptimisticCallback( cb );
```

Plug-in d'écouteurs d'événements

Vous pouvez utiliser les plug-in ObjectGridEventListener et MapEventListener afin de configurer des notifications pour divers événements se produisant dans le cache d'eXtreme Scale. Les plug-in du programme d'écoute sont enregistrés avec une instance ObjectGrid ou BackingMap, comme les autres plug-in eXtreme Scale, et ajoutent des points d'intégration et de personnalisation pour les applications et les fournisseurs de cache.

Plug-in ObjectGridEventListener

Un plug-in ObjectGridEventListener fournit des événements du cycle de vie eXtreme Scale pour l'instance ObjectGrid, les fragments et les transactions. Utilisez le plug-in ObjectGridEventListener pour recevoir des notifications lorsque des événements importants se produisent sur une interface ObjectGrid. Ces événements incluent l'initialisation d'ObjectGrid, le début d'une transaction, la fin d'une transaction et la destruction d'un ObjectGrid. Pour écouter ces événements, créez une classe qui implémente l'interface ObjectGridEventListener et ajoutez-la à eXtreme Scale.

Pour plus d'informations sur la création d'un plug-in ObjectGridEventListener, voir «Plug-in ObjectGridEventListener», à la page 241. Pour plus d'informations, vous pouvez également vous reporter à la documentation de l'API.

Ajout et suppression d'instances ObjectGridEventListener

Un ObjectGrid peut posséder plusieurs programmes d'écoute ObjectGridEventListener. Ajoutez et supprimez les programmes d'écoute à l'aide des méthodes addEventListener, setEventListeners et removeEventListener sur l'interface ObjectGrid. Vous pouvez également enregistrer de manière déclarative les plug-in ObjectGridEventListener avec le fichier descripteur d'ObjectGrid. Pour des exemples, voir «Plug-in ObjectGridEventListener», à la page 241.

Plug-in MapEventListener

Un plug-in MapEventListener fournit les notifications de rappel et les modifications importantes de l'état du cache qui se produisent pour une instance BackingMap. Pour plus de détails sur l'enregistrement d'un plug-in MapEventListener, voir «Plug-in MapEventListener», à la page 240. Pour plus d'informations, vous pouvez également vous reporter à la documentation de l'API.

Ajout et suppression d'instances MapEventListener

Un système eXtreme Scale peut posséder plusieurs programmes d'écoute MapEventListener. Ajoutez et supprimez des programmes d'écoute avec les méthodes addMapEventListener, setMapEventListeners et removeMapEventListener sur l'interface BackingMap. Vous pouvez également enregistrer de manière déclarative les programmes d'écoute MapEventListener avec le fichier descripteur d'ObjectGrid. Pour des exemples, voir «Plug-in MapEventListener».

Plug-in MapEventListener

Un plug-in MapEventListener fournit les notifications de rappel et les modifications importantes de l'état du cache qui se produisent pour un objet BackingMap : lorsque le préchargement d'une mappe est terminé ou qu'une entrée est expulsée de la mappe. Un plug-in MapEventListener particulier est une classe personnalisée que vous écrivez lors de l'implémentation de l'interface MapEventListener.

Conventions du plug-in MapEventListener

Lorsque vous développez un plug-in MapEventListener, vous devez suivre les conventions de plug-in communes. Pour plus d'informations sur ces conventions de plug-in, voir «Introduction aux plug-in», à la page 209. Pour les autres types de plug-in de programme d'écoute, voir «Plug-in d'écouteurs d'événements», à la page 239.

Une fois que vous avez écrit une implémentation MapEventListener, vous pouvez l'intégrer à la configuration BackingMap à l'aide d'un programme ou d'une configuration XML.

Ecriture d'une implémentation MapEventListener

Votre application peut inclure une implémentation du plug-in MapEventListener. Le plug-in doit implémenter l'interface MapEventListener pour recevoir les événements importants sur une mappe. Des événements sont envoyés au plug-in MapEventListener lorsqu'une entrée est expulsée de la mappe et à la fin du préchargement d'une mappe.

Intégration d'une implémentation MapEventListener à l'aide d'un XML

Une implémentation MapEventListner peut être configurée à l'aide d'un XML. Le XML suivant doit se trouver dans le fichier `myGrid.xml` :

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridconfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
    <objectGrids>
        <objectGrid name="myGrid">
            <backingMap name="myMap" pluginCollectionRef="myPlugins" />
        </objectGrid>
    </objectGrids>
    <backingMapPluginCollections>
        <backingMapPluginCollection id="myPlugins">
            <bean id="MapEventListener" className=
       "com.company.org.MyMapEventListener" />
        </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

Si ce fichier est fourni à l'instance ObjectGridManager, la création de cette configuration est facilitée. Le fragment de code suivant indique comment créer une instance ObjectGrid à l'aide de ce fichier XML. Pour l'instance ObjectGrid nouvellement créée, un MapEventListener est défini sur la mappe de sauvegarde myMap.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid =
 objectGridManager.createObjectGrid("myGrid", new URL("file:etc/test/myGrid.xml"),
  true, false);
```

Intégration d'une implémentation MapEventListener à l'aide d'un programme

Le nom de classe du MapEventListener personnalisé est com.company.org.MyMapEventListener. Cette classe implémente l'interface MapEventListener. Le fragment de code suivant crée l'objet MapEventListener personnalisé et l'ajoute à un objet BackingMap :

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
BackingMap myMap = myGrid.defineMap("myMap");
MyMapEventListener myListener = new MyMapEventListener();
myMap.addMapEventListener(myListener);
```

Plug-in ObjectGridEventListener

Un plug-in ObjectGridEventListener fournit des événements du cycle de vie WebSphere eXtreme Scale pour l'ObjectGrid, les fragments et les transactions. Un plug-in ObjectGridEventListener fournit des notifications lorsqu'un ObjectGrid est initialisé ou détruit et lorsqu'une transaction est démarrée ou terminée. Les plug-in ObjectGridEventListener sont des classes personnalisées que vous créez lorsque vous implémentez l'interface ObjectGridEventListener. L'implémentation peut éventuellement inclure des sous-interfaces ObjectGridEventGroup et suivre les conventions communes aux plug-in eXtreme Scale.

Présentation

Un plug-in ObjectGridEventListener est utile si un plug-in Loader est disponible et que vous devez initialiser des connexions JDBC (Java Database Connectivity) ou des connexions à un système dorsal lorsque des transactions démarrent ou s'arrêtent. Généralement un plug-in ObjectGridEventListener et un plug-in Loader sont écrits ensembles.

Ecriture d'un plug-in ObjectGridEventListener

Un plug-in ObjectGridEventListener doit implémenter l'interface ObjectGridEventListener pour recevoir des notifications sur les événements eXtreme Scale importants. Pour recevoir des notifications d'événement supplémentaires, vous pouvez implémenter les interfaces ci-après. Ces sous-interfaces sont incluses dans l'interface ObjectGridEventGroup :

- Interface ShardEvents
- Interface ShardLifecycle
- Interface TransactionEvents

Pour plus d'informations sur ces interfaces, voir la documentation de l'API.

Evénements de fragment

Lorsque le service de catalogue place des fragments primaires de partition ou des fragments répliques dans une machine virtuelle Java, une instance ObjectGrid est créée dans cette machine virtuelle Java pour héberger ce fragment. Certaines applications qui doivent démarrer des unités d'exécution sur la machine virtuelle Java qui héberge le fragment primaire doivent être notifiées de ces événements. L'interface ObjectGridEventGroup.ShardEvents déclare les méthodes shardActivate et shardDeactivate. Ces méthodes ne sont appelées que si un fragment est activé comme fragment primaire ou qu'un fragment est désactivé d'un serveur primaire. Ces deux événements permettent à l'application de démarrer des unités d'exécution supplémentaires si le fragment est un fragment primaire et d'arrêter les unités d'exécution si le fragment redevient une réplique ou est mis hors service.

Une application peut déterminer quelle partition a été activée en recherchant une mappe de sauvegarde spécifique dans la référence ObjectGrid fournie à la méthode shardActivate à l'aide de la méthode ObjectGrid#getMap. L'application peut alors déterminer le numéro de partition à l'aide de la méthode BackingMap#getPartitionId(). Les partitions sont numérotées de 0 au nombre de partitions dans le descripteur de déploiement moins un.

Evénements de cycle de vie du fragment

Les événements des méthodes ObjectGridEventListener.initialize et ObjectGridEventListener.destroy sont distribués à l'aide de l'interface ObjectGridEventGroup.ShardLifecycle.

Evénements de transaction

Les méthodes ObjectGridEventListener.transactionBegin et ObjectGridEventListener.transactionEnd sont distribuées via l'interface ObjectGridEventGroup.TransactionEvents.

Avantages de cette approche

Si un plug-in ObjectGridEventListener implémente les interfaces ObjectGridEventListener et ShardLifecycle, les événements de cycle de vie du fragment sont les seuls à être distribués au programme d'écoute. Une fois que vous avez implémenté l'une des nouvelles interfaces ObjectGridEventGroup internes, eXtreme Scale ne distribue que les événements spécifiques des nouvelles interfaces. Avec cette implémentation, le code offre une compatibilité amont. Si vous utilisez les nouvelles interfaces internes, il peut désormais ne recevoir que les événements nécessaires.

Utilisation du plug-in ObjectGridEventListener

Pour utiliser un plug-in ObjectGridEventListener personnalisé, créez d'abord une classe qui implémente l'interface ObjectGridEventListener et les éventuelles sous-interfaces ObjectGridEventGroup facultatives. Ajoutez le programme d'écoute personnalisé à un ObjectGrid pour recevoir la notification d'événements importants. Deux approches permettent d'ajouter un plug-in ObjectGridEventListener dans la configuration d'eXtreme Scale : la configuration à l'aide d'un programme et la configuration XML.

Configuration d'un plug-in ObjectGridEventListener à l'aide d'un programme

Supposons que le nom de classe du programme d'écoute d'événement d'eXtreme Scale correspond à la classe com.company.org.MyObjectGridEventListener. Cette classe implémente l'interface ObjectGridEventListener. Le fragment de code suivant crée l'interface ObjectGridEventListener personnalisée et l'ajoute à un ObjectGrid.

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
MyObjectGridEventListener myListener = new MyObjectGridEventListener();
myGrid.addEventListener(myListener);
```

Configuration d'un plug-in ObjectGridEventListener avec XML

Vous pouvez également configurer un plug-in ObjectGridEventListener à l'aide de XML. Le XML ci-après crée une configuration équivalente au programme d'écoute d'événements créé à l'aide d'un programme décrit. Le texte suivant doit se trouver dans le fichier `myGrid.xml` :

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
    <objectGrids>
        <objectGrid name="myGrid">
            <bean id="ObjectGridEventListener"
       className="com.company.org.MyObjectGridEventListener" />
            <backingMap name="Book"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

Notez que les déclarations de bean précèdent les déclarations de mappe de sauvegarde. Fournissez ce fichier au plug-in ObjectGridManager pour faciliter la création de cette configuration. Le fragment de code suivant indique comment créer une instance ObjectGrid à l'aide de ce fichier XML. L'instance ObjectGrid créée possède un programme d'écoute ObjectGridEventListener défini sur l'ObjectGrid myGrid.

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid",
 new URL("file:etc/test/myGrid.xml"), true, false);
```

Plug-in d'indexation personnalisée des objets en cache

Avec un plug-in MapIndexPlugin (index), vous pouvez écrire des stratégies d'indexation personnalisées dont les possibilités dépassent celles des index pré-intégrés fournis par eXtreme Scale.

Pour des informations générales sur l'indexation, voir Indexation.

Pour plus d'informations sur l'utilisation de l'indexation, voir «Utilisation de l'indexation pour l'accès aux données ne correspondant pas à une clé», à la page 249.

Les implémentations de MapIndexPlugin doivent utiliser l'interface MapIndexPlugin et suivre les conventions communes de plug-in d'eXtreme Scale.

Les sections suivantes comprennent certaines méthodes importantes de l'interface d'index.

Méthode setProperties

Utilisez la méthode setProperties pour initialiser le plug-in d'index automatiquement. Le paramètre d'objet Properties qui est passé dans la méthode

devrait contenir les informations de configuration requise pour initialiser correctement le plug-in d'index. L'implémentation de la méthode setProperties et celle de la méthode getProperties sont nécessaires dans un environnement réparti car la configuration du plug-in d'index se déplace entre les processus client et serveur. Voici un exemple d'implémentation de cette méthode.

```
setProperties(Properties properties)

// exemple de code de la méthode setProperties
    public void setProperties(Properties properties) {
        ivIndexProperties = properties;

        String ivRangeIndexString = properties.getProperty("rangeIndex");
        if (ivRangeIndexString != null && ivRangeIndexString.equals("true")) {
            setRangeIndex(true);
        }
        setName(properties.getProperty("indexName"));
        setAttributeName(properties.getProperty("attributeName"));

        String ivFieldAccessAttributeString = properties.getProperty("fieldAccessAttribute");
        if (ivFieldAccessAttributeString != null && ivFieldAccessAttributeString.equals("true")) {
            setFieldAccessAttribute(true);
        }

        String ivPOJOKeyIndexString = properties.getProperty("POJOKeyIndex");
        if (ivPOJOKeyIndexString != null && ivPOJOKeyIndexString.equals("true")) {
            setPOJOKeyIndex(true);
        }
    }
```

Méthode getProperties

La méthode getProperties extrait la configuration du plug-in d'index depuis une instance MapIndexPlugin. Vous pouvez utiliser les propriétés extraites pour initialiser une autre instance de MapIndexPlugin avec les mêmes états internes. L'implémentation des méthodes getProperties et setProperties est nécessaire pour un environnement réparti. Voici un exemple d'implémentation de la méthode getProperties.

```
getProperties()

// exemple de code de la méthode getProperties
    public Properties getProperties() {
        Properties p = new Properties();
        p.put("indexName", indexName);
        p.put("attributeName", attributeName);
        p.put("rangeIndex", ivRangeIndex ? "true" : "false");
        p.put("fieldAccessAttribute", ivFieldAccessAttribute ? "true" : "false");
        p.put("POJOKeyIndex", ivPOJOKeyIndex ? "true" : "false");
        return p;
    }
```

Méthode setEntityMetadata

La méthode setEntityMetadata est appelée par l'exécution de WebSphere eXtreme Scale au cours de l'initialisation pour définir l'EntityMetadata du BackingMap associé sur l'instance de MapIndexPlugin. L'EntityMetadata est nécessaire pour la prise en charge de l'indexation des objets bloc de données. Un bloc de données est un ensemble de données qui représente un objet entité ou sa clé. Si le BackingMap est pour une entité, vous devez implémenter cette méthode.

L'échantillon de code suivant implémente la méthode setEntityMetadata.

```
setEntityMetadata(EntityMetadata entityMetadata)

// exemple de code de la méthode setEntityMetadata
    public void setEntityMetadata(EntityMetadata entityMetadata) {
        ivEntityMetadata = entityMetadata;
        if (ivEntityMetadata != null) {
            // ceci est une mappe de blocs de données
            TupleMetadata valueMetadata = ivEntityMetadata.getValueMetadata();
            int numAttributes = valueMetadata.getNumAttributes();
            for (int i = 0; i < numAttributes; i++) {
```

```
                String tupleAttributeName = valueMetadata.getAttribute(i).getName();
                if (attributeName.equals(tupleAttributeName)) {
                    ivTupleValueIndex = i;
                    break;
                }
            }

            if (ivTupleValueIndex == -1) {
                // attribut non trouvé dans le bloc de données valeur, essaie de le trouver sur le bloc
                // de données clé.
                // si trouvé dans le bloc de données clé, implique l'indexation clé sur un des attributs

                //  clés de bloc de données.
                TupleMetadata keyMetadata = ivEntityMetadata.getKeyMetadata();
                numAttributes = keyMetadata.getNumAttributes();
                for (int i = 0; i < numAttributes; i++) {
                    String tupleAttributeName = keyMetadata.getAttribute(i).getName();
                    if (attributeName.equals(tupleAttributeName)) {
                        ivTupleValueIndex = i;
                        ivKeyTupleAttributeIndex = true;
                        break;
                    }
                }
            }

            if (ivTupleValueIndex == -1) {
                // si entityMetadata n'est pas nul et que nous n'avons pas trouvé
       // attributeName dans entityMetadata, il s'agit d'une
                // erreur
                throw new ObjectGridRuntimeException("Invalid attributeName.  Entity: " +
         ivEntityMetadata.getName());
            }
        }
    }
```

Méthode de nom d'attribut

La méthode setAttributeName définit le nom de l'attribut à indexer. La classe d'objets mise en cache doit fournir la méthode get pour l'attribut indexé. Par exemple, si l'objet possède un attribut employeeName ou EmployeeName, l'index appelle la méthode getEmployeeName sur l'objet pour extraire la valeur de l'attribut. Le nom d'attribut doit être le même que celui qui se trouve dans la méthode get, et l'attribut doit implémenter l'interface Comparable. Si l'attribut est de type booléen, vous pouvez également utiliser le modèle de la méthode isAttributeName.

La méthode getAttributeName renvoie le nom de l'attribut indexé.

Méthode getAttribute

La méthode getAttribute renvoie la valeur de l'attribut indexé à partir de l'objet spécifié. Par exemple, si un objet Employee a un attribut appelé employeeName indexé, vous pouvez utiliser la méthode getAttribute pour extraire la valeur d'attribut employeeName depuis un objet Employee spécifié. Cette méthode est requise dans un environnement WebSphere eXtreme Scale réparti.

```
getAttribute(Object value)

// exemple de code de la méthode getAttribute
    public Object getAttribute(Object value) throws ObjectGridRuntimeException {
        if (ivPOJOKeyIndex) {
            // Dans le cas de l'indexation d'une clé d'objet Java simple, il n'est pas nécessaire
            // d'obtenir l'attribut à partir de l'objet valeur.
            // La clé elle-même est la valeur d'attribut utilisée pour construire l'index.
            return null;
        }

        try {
            Object attribute = null;
            if (value != null) {
                // indiquer la valeur Tuple si ivTupleValueIndex != -1
                if (ivTupleValueIndex == -1) {
                    // valeur normale
                    if (ivFieldAccessAttribute) {
                        attribute = this.getAttributeField(value).get(value);
                    } else {
                        attribute = getAttributeMethod(value).invoke(value, emptyArray);
                    }
```

```
            } else {
                // valeur Tuple
                attribute = extractValueFromTuple(value);
            }
        }
        return attribute;
    } catch (InvocationTargetException e) {
        throw new ObjectGridRuntimeException(
                "Caught unexpected Throwable during index update processing,
       index name = " + indexName + ": " + t,
                t);
    } catch (Throwable t) {
        throw new ObjectGridRuntimeException(
                "Caught unexpected Throwable during index update processing,
      index name = " + indexName + ": " + t,
                t);
    }
}
```

Index HashIndex composite

L'index HashIndex composite permet d'améliorer les performances des requêtes et d'éviter des recherches dans les mappes, consommatrices en ressources. Il facilite également les recherches d'objets mis en cache effectuées par l'API HashIndex lorsque les critères de recherche contiennent plusieurs attributs.

Amélioration des performances

L'index HashIndex de hachage constitue un outil rapide et pratique pour rechercher des objets mis en cache lorsque plusieurs attributs sont indiqués dans les critères de recherche. Il prend en charge les recherches de correspondance d'attribut complète, mais pas les recherches de plages.

Remarque : Les index composites ne prennent pas en charge l'opérateur BETWEEN dans le langage de requête ObjectGrid car la prise en charge des plages est obligatoire pour cet opérateur. De même, les opérateurs conditionnels "supérieur à" (>) et "inférieur à" (<) ne fonctionnent pas, pour la même raison.

L'index composite permet d'améliorer les performances des requêtes lorsque l'index composite approprié est disponible pour la condition WHERE. Ses attributs sont alors exactement les mêmes que ceux indiqués dans la condition WHERE lorsque les attributs correspondent complètement.

Une requête, peut contenir plusieurs attributs au sein d'une même condition, comme dans l'exemple ci-dessous :

```
SELECT a FROM Address a WHERE a.city='Rochester' AND a.state='MN' AND
a.zipcode='55901'
```

L'index composite améliore les performances des requêtes car il permet d'éviter les recherches dans les mappes ou de joindre plusieurs résultats d'index contenant un seul attribut. Dans cet exemple, si un index composite est défini avec des attributs (city,state,zipcode), le moteur de recherche peut utiliser l'index composite pour rechercher l'entrée associée à `city='Rochester'`, `state='MN'` et `zipcode='55901'`. Sans l'index composite et l'index d'attribut pour les attributs city, state et zipcode, le moteur de recherche doit effectuer la recherche dans la mappe ou joindre plusieurs recherches contenant un seul attribut, ce qui entraîne généralement une augmentation de la consommation des ressources. Par ailleurs, l'exécution d'une requête pour l'index composite prend uniquement en charge le modèle de correspondance complète.

Configuration d'un index composite

Vous pouvez configurer une indexation composite de trois manières différentes : avec XML, à l'aide d'un programme et avec des annotations d'entités (uniquement pour les mappes d'entités).

Utilisation de XML

Pour configurer un index composite avec XML, entrez des lignes de code semblables à ce qui suit dans l'élément backingMapPluginCollections du fichier de configuration.

Index composite - configuration de type XML

```
<bean id="MapIndexPlugin"  className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="Address.CityStateZip"/>
<property name="AttributeName" type="java.lang.String" value="city,state,zipcode"/>
</bean>
```

Configuration par programmation

Les lignes de code représentées ci-dessous permettent de créer le même index composite que dans l'exemple précédent.

```
HashIndex mapIndex = new HashIndex();
 mapIndex.setName("Address.CityStateZip");
 mapIndex.setAttributeName(("city,state,zipcode"));
 mapIndex.setRangeIndex(true);

 BackingMap bm = objectGrid.defineMap("mymap");
 bm.addMapIndexPlugin(mapIndex);
```

Notez que la configuration d'un index composite est identique à la configuration d'un index standard avec XML, à l'exception de la valeur de la propriété attributeName. Dans un index composite, cette valeur est une liste d'attributs séparés par des virgules. Par exemple, la classe de valeur Address a trois attributs : city, state et zipcode. Un index composite peut être défini avec la valeur de propriété attributeName `"city,state,zipcode"` indiquant que la ville, l'état et le code postal sont inclus dans l'index composite.

Notez également qu'un index HashIndexes composite ne prend pas en charge les recherches de plages et que sa propriété RangeIndex ne peut pas être associée à true.

Avec annotations d'entités

Dans le cas d'une mappe d'entités, il est possible d'utiliser des annotations pour définir un index composite. Vous pouvez définir une liste de CompositeIndex dans l'annotation CompositeIndexes au niveau de la classe d'entités. Le CompositeIndex a un nom et une propriété attributeNames. Chaque CompositeIndex est associé à une instance HashIndex appliquée à la mappe de sauvegarde associée de l'entité. L'index HashIndex est configuré en tant qu'index ne contenant pas de plage.

```
@Entity
@CompositeIndexes({
    @CompositeIndex(name="CityStateZip", attributeNames="city,state,zipcode"),
    @CompositeIndex(name="lastnameBirthday", attributeNames="lastname,birthday")
 })
public class Address {
    @Id int id;
    String street;
    String city;
    String state;
    String zipcode;
    String lastname;
    Date birthday;
}
```

La propriété de nom de chaque index composite doit être unique dans l'entité et dans la mappe de sauvegarde. Si aucun nom n'est indiqué, un nom est généré. La propriété attributeNames permet de remplir le nom d'attribut HashIndex avec la liste d'attributs séparés par des virgules. Les noms d'attribut coïncident avec les noms de champ persistant lorsque les entités sont configurées pour utiliser l'accès par champ, ou le nom de propriété tel qu'il est défini pour les conventions de dénomination JavaBeans pour les entités d'accès par propriété. Par exemple : si le nom d'attribut est "street", la méthode d'accès get de la propriété se nommera getStreet.

Execution de recherches avec un index composite

Une fois l'index composite configuré, l'application peut utiliser la méthode findAll(Object) de l'interface MapIndex pour exécuter des recherches, comme ci-dessous.

```
Session sess = objectgrid.getSession();
ObjectMap map = sess.getMap("MAP_NAME");
MapIndex codeIndex = (MapIndex) map.getIndex("INDEX_NAME");
Object[] compositeValue = new Object[]{ MapIndex.EMPTY_VALUE,
   "MN", "55901"};
Iterator iter = mapIndex.findAll(compositeValue);
```

La valeur MapIndex.EMPTY_VALUE est affectée à compositeValue[0] qui indique que l'attribut city est exclu de l'évaluation. Seuls les objets dont l'attribut state est égal à "MN" et l'attribut zipcode est égal à "55901" figureront dans le résultat.

Les requêtes suivantes bénéficient de la configuration de l'index composite précédent :

```
SELECT a FROM Address a WHERE a.city='Rochester' AND a.state='MN' AND
a.zipcode='55901'

SELECT a FROM Address a WHERE a.state='MN' AND a.zipcode='55901'
```

Le moteur de requête cherche l'index composite approprié et l'utilise pour améliorer les performances des requêtes dans les recherches à correspondance complète.

Dans certains scénarios, l'application peut avoir besoin de définir plusieurs index composites dont les attributs se chevauchent pour satisfaire toutes les requêtes à correspondance complète. L'augmentation du nombre d'index risque de ralentir les performances des opérations relatives aux mappes.

Migration et interopérabilité

La seule contrainte liée à l'utilisation d'un index composite est qu'une application ne peut pas le configurer dans un environnement réparti contenant des conteneurs hétérogènes. Il est impossible de faire cohabiter des conteneurs anciens et nouveaux, car les anciens ne reconnaissent pas la configuration d'index composite. Celui-ci est semblable à un index d'attribut normal, mais il permet en outre l'indexation de plusieurs attributs. En cas d'utilisation d'un index d'attribut standard, un environnement composé de plusieurs types de conteneurs est viable.

Utilisation de l'indexation pour l'accès aux données ne correspondant pas à une clé

L'utilisation de l'indexation comme alternative à l'accès aux clés de données peut s'avérer plus efficace.

Etapes requises

1. Ajoutez des plug-in d'indexation statique ou dynamique à la mappe de sauvegarde.
2. Obtenez l'objet proxy de l'index d'application en émettant la méthode getIndex de l'ObjectMap.
3. Transtypez l'objet proxy d'index vers une interface d'index d'application appropriée, MapIndex, MapRangeIndex ou une interface d'index personnalisée, par exemple.
4. Utilisez les méthodes définies dans l'interface d'index de l'application pour trouver des objets mis en cache.

La classe HashIndex est l'implémentation du plug-in d'indexation pré-intégré capable de prendre en charge les deux interfaces pré-intégrées d'API d'indexation : MapIndex et MapRangeIndex. Vous pouvez également créer vos propres index. Vous pouvez ajouter HashIndex à la BackingMap en tant qu'index statique ou dynamique, obtenir un objet proxy d'index MapIndex ou MapRangeIndex et utiliser cet objet proxy pour chercher des objets mis en cache.

Remarque : Dans un environnement réparti, si l'objet d'index est obtenu à partir d'une ObjectGrid client, il est de type client et toutes les opérations d'index sont exécutées dans une ObjectGrid de serveur distant. Si la mappe est partitionnée, l'opération d'index est exécutée à distance sur chaque partition et les résultats de chaque partition sont fusionnés avant d'être renvoyés à l'application. Les performances seront déterminées par le nombre de partitions et la taille des résultats renvoyés par chaque partition. De faibles performances peuvent être générées si les deux facteurs sont élevés.

Concernant la configuration de HashIndex, voir Configuration de HashIndex.

Si vous voulez écrire votre propre plug-in d'index, reportez-vous à la rubrique«Plug-in d'indexation personnalisée des objets en cache», à la page 243.

Concernant l'indexation, voir Indexation et «Index HashIndex composite», à la page 246.

Ajout de plug-in d'index statique

Deux approches permettent d'ajouter un plug-in d'index statique dans la configuration de mappe de sauvegarde : la configuration XML et la configuration par programmation. L'exemple suivant illustre l'approche de la configuration XML.

Ajout de plug-in d'index statique : approche de la configuration XML

```
<backingMapPluginCollection id="person">
     <bean id="MapIndexplugin"
   className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
         <property name="Name" type="java.lang.String" value="CODE"
      description="index name" />
         <property name="RangeIndex" type="boolean" value="true"
      description="true for MapRangeIndex" />
         <property name="AttributeName" type="java.lang.String" value="employeeCode"
      description="attribute name" />
     </bean>
</backingMapPluginCollection>
```

Dans cet exemple de configuration XML, la classe pré-intégrée HashIndex est utilisée comme plug-in d'indexation. HashIndex prend en charge des propriétés que l'utilisateur peut configurer telles que Name, RangeIndex et AttributeName dans l'exemple précédent.

- La propriété Name est configuré en tant que CODE, une chaîne identifiant ce plug-in d'indexation. La valeur de la propriété Name doit être unique dans l'étendue de la mappe de sauvegarde et peut servir à extraire l'objet d'index de l'instance ObjectMap pour la mappe de sauvegarde.
- La propriété RangeIndex est configurée avec la valeur true, ce qui signifie que l'application peut transtyper vers l'interface MapRangeIndex l'objet d'index extrait. Si la propriété RangeIndex est configuré avec la valeur false, l'application ne peut que transtyper l'objet d'index extrait que vers l'interface MapIndex. Une interface MapRangeIndex prend en charge des fonctions de recherche de données à l'aide des fonctions de plage (range) telles que greater than, less than ou les deux, alors qu'un index MapIndex prend uniquement en charge les fonctions d'égalité (equals). Si l'index est utilisé par la requête, la propriété RangeIndex doit être définie sur true pour les index à attribut unique. Pour un index de relation ou composite, la propriété RangeIndex doit être définie sur false.
- La propriété AttributeName est configurée avec la valeur employeeCode, ce qui signifie que l'attribut employeeCode de l'objet mis en cache est utilisé pour générer un index à attribut unique. Si une application doit rechercher des objets mis en cache pour plusieurs attributs, la propriété AttributeName peut être définie sur une liste d'attributs délimitée par des virgules, produisant un index composite.

Consultez les informations relatives à la configuration de l'index HashIndex dans le *Guide d'administration* pour en savoir plus.

L'interface BackingMap comporte deux méthodes que vous pouvez utiliser pour ajouter des plug-in d'index statique : addMapIndexplugin et setMapIndexplugins. Pour plus d'informations, consultez la documentation d'API.

L'exemple de code suivant illustre l'approche de la configuration par programmation :

Ajout de plug-in d'index statique : approche de la configuration par programmation

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;

    ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
    ObjectGrid ivObjectGrid = ogManager.createObjectGrid( "grid" );
    BackingMap personBackingMap = ivObjectGrid.getMap("person");

    // utilisez la classe prê-intégrée HashIndex comme classe de plug-in d'indexation.
    HashIndex mapIndexplugin = new HashIndex();
    mapIndexplugin.setName("CODE");
    mapIndexplugin.setAttributeName("EmployeeCode");
    mapIndexplugin.setRangeIndex(true);
    personBackingMap.addMapIndexplugin(mapIndexplugin);
```

Utilisation des index statiques

Après l'ajout d'un plug-in d'index statique à une configuration de mappe de sauvegarde et l'initialisation de l'instance ObjectGrid contenante, les applications peuvent extraire l'objet d'index par nom à partir de l'instance ObjectMap pour la

mappe de sauvegarde. Distribuez l'objet d'index vers l'interface d'index de l'application. Les opérations prises en charge par l'interface d'index de l'application peuvent désormais avoir lieu.

L'exemple de code suivant montre comment extraire et utiliser les index statiques.

Exemple d'utilisation des index statiques

```
Session session = ivObjectGrid.getSession();
    ObjectMap map = session.getMap("person ");
    MapRangeIndex codeIndex = (MapRangeIndex) m.getIndex("CODE");
    Iterator iter = codeIndex.findLessEqual(new Integer(15));
    while (iter.hasNext()) {
        Object key = iter.next();
        Object value = map.get(key);
    }
```

Ajout, suppression et utilisation des index dynamiques

Vous pouvez créer et supprimer à tout moment des index dynamiques d'une instance BackingMap. Un index dynamique diffère d'un index statique en ce qu'il peut être créé même après l'initialisation de l'instance ObjectGrid contenante. Contrairement à l'indexation statique, l'indexation dynamique est un processus asynchrone et doit être à l'état ready avant d'être utilisée. Cette méthode utilise la même approche pour extraire et utiliser les index dynamiques que pour les index statiques. Vous pouvez supprimer un index dynamique s'il n'est plus utile. L'interface BackingMap comprend deux méthodes pour créer et supprimer les index dynamiques.

Pour plus d'informations sur les méthodes createDynamicIndex et removeDynamicIndex, reportez-vous à l'API BackingMap.

Exemple d'utilisation des index dynamiques

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;

        ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
        ObjectGrid og = ogManager.createObjectGrid("grid");
        BackingMap bm = og.getMap("person");
        og.initialize();

        // créez l'index après l'initialisation de l'ObjectGrid sans DynamicIndexCallback.
        bm.createDynamicIndex("CODE", true, "employeeCode", null);

        try {
            // Si DynamicIndexCallback n'est pas utilisé, attendre que l'index soit prêt.
            // Le délai d'attente dépend de la taille actuelle de la mappe
            Thread.sleep(3000);
        } catch (Throwable t) {
            // ...
        }

        // Lorsque l'index est prêt, les applications peuvent tenter d'obtenir l'instance d'interface
        // de l'index d'application.
        // Les applications doivent trouver un moyen de vérifier que l'index est prêt à être utilisé,
        // faute de quoi elles utilisent l'interface DynamicIndexCallback.
        // L'exemple ci-dessous illustre comment attendre que l'index soit prêt
        // Prenez en compte la taille de la mappe dans le délai d'attente total.

        Session session = og.getSession();
        ObjectMap m = session.getMap("person");
        MapRangeIndex codeIndex = null;

        int counter = 0;
        int maxCounter = 10;
        boolean ready = false;
        while (!ready && counter < maxCounter) {
            try {
                counter++;
                codeIndex = (MapRangeIndex) m.getIndex("CODE");
                ready = true;
            } catch (IndexNotReadyException e) {
                // implique que l'index n'est pas prêt, ...
                System.out.println("Index pas prêt. continuez de patienter.");
                try {
                    Thread.sleep(3000);
                } catch (Throwable tt) {
                    // ...
```

```
            }
        } catch (Throwable t) {
            // exception inattendue
            t.printStackTrace();
        }
    }

    if (!ready) {
        System.out.println("Index pas prêt.  Remédiez à cette situation.");
    }

    // Utilisez l'index pour exécuter des requêtes
    // Reportez-vous à l'interface MapIndex ou MapRangeIndex pour les opérations prises en charge.
    // L'attribut d'objet sur lequel repose l'index est EmployeeCode.
    // Supposons que l'attribut EmployeeCode est de type entier : le
    // paramètre transmis aux opérations d'index présente ce type de données.

    Iterator iter = codeIndex.findLessEqual(new Integer(15));

    // supprimez l'index dynamique lorsqu'il n'est plus nécessaire

    bm.removeDynamicIndex("CODE");
```

Interface DynamicIndexCallback

L'interface DynamicIndexCallback est conçue pour les applications qui visent à recevoir des notifications des événements d'indexation ready, error ou destroy. L'interface DynamicIndexCallback est un paramètre facultatif pour la méthode createDynamicIndex de la mappe de sauvegarde. Avec une instance DynamicIndexCallback enregistrée, les applications peuvent exécuter la logique applicative en recevant une notification d'un événement d'indexation. Par exemple, l'événement ready signifie que l'index est prêt à être utilisé. Lorsqu'une notification est reçue pour cet événement, une application peut tenter d'extraire et d'utiliser l'instance d'interface de l'index d'application. Pour plus d'informations, reportez-vous à l'API DynamicIndexCallback dans la documentation d'API.

L'exemple de code suivant illustre l'utilisation de l'interface DynamicIndexCallback :

Utilisation de l'interface DynamicIndexCallback

```
BackingMap personBackingMap = ivObjectGrid.getMap("person");
    DynamicIndexCallback callback = new DynamicIndexCallbackImpl();
    personBackingMap.createDynamicIndex("CODE", true, "employeeCode", callback);


    class DynamicIndexCallbackImpl implements DynamicIndexCallback {
        public DynamicIndexCallbackImpl() {
        }

        public void ready(String indexName) {
            System.out.println("DynamicIndexCallbackImpl.ready() -> indexName = " + indexName);

            // Simulez le comportement d'une application lors de la notification ready.
            // Normalement, l'application doit patienter jusqu'à l'obtention de l'état ready, puis doit mettre en oeuvre
            // la logique d'utilisation de l'index.
            if("CODE".equals(indexName)) {
                ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
                ObjectGrid og = ogManager.createObjectGrid( "grid" );
                Session session = og.getSession();
                ObjectMap map = session.getMap("person");
                MapIndex codeIndex = (MapIndex) map.getIndex("CODE");
                Iterator iter = codeIndex.findAll(codeValue);
            }
        }

        public void error(String indexName, Throwable t) {
            System.out.println("DynamicIndexCallbackImpl.error() -> indexName = " + indexName);
            t.printStackTrace();
        }

        public void destroy(String indexName) {
            System.out.println("DynamicIndexCallbackImpl.destroy() -> indexName = " + indexName);
        }
    }
```

Plug-in de communication avec les stockages de persistance

Avec un plug-in Loader d'eXtreme Scale, une mappe ObjectGrid peut être utilisée comme cache pour les données généralement conservées dans un stockage de persistance sur le même système ou un autre système. Généralement, une base de données ou un système de fichiers est utilisé comme stockage de persistance. Une machine virtuelle Java (JVM) distante peut également être utilisée comme source

des données, ce qui permet de créer des caches basés sur un concentrateur à l'aide d'ObjectGrid. Un chargeur peut lire et écrire des données vers un stockage persistant ou à partir de celui-ci.

Les Loaders sont des plug-in de mappe de sauvegarde appelés lorsque des modifications sont apportées à la mappe de sauvegarde ou lorsque cette dernière est dans l'impossibilité de répondre à une demande de données (absence dans le cache).

Pour plus d'informations, reportez-vous aux informations sur les concepts de cache dans la *Présentation du produit*.

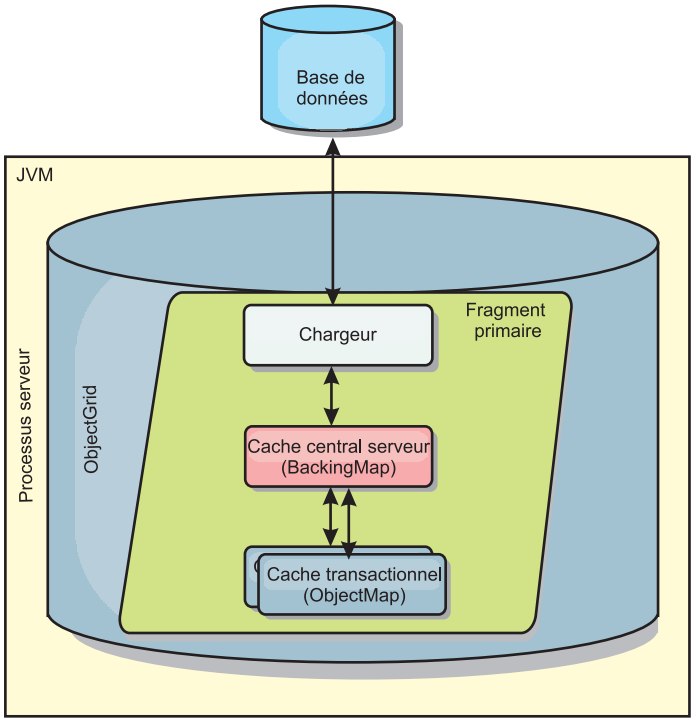


Figure 3. Chargeur

WebSphere eXtreme Scale est livré avec deux chargeurs permettant de s'intégrer aux dorsaux de bases de données relationnelles. Les chargeurs Java Persistence API (JPA) utilisent les fonctions du mappage objet-relationnel (ORM) des implémentations OpenJPA et Hibernate de la spécification JPA.

Utilisation d'un chargeur

Pour ajouter un chargeur à la configuration BackingMap, vous pouvez utiliser la configuration programmatique ou la configuration XML. Un chargeur possède la relation suivante avec une mappe de sauvegarde :

- Une mappe de sauvegarde peut avoir un seul chargeur.
- Une mappe de sauvegarde client (cache proche) ne peut pas avoir de chargeur.
- Une définition de chargeur peut être appliquée à plusieurs mappes de sauvegarde, mais chaque mappe de sauvegarde dispose de sa propre instance de chargeur.

Ajout d'un Loader à l'aide d'un programme

L'extrait de code suivant illustre comment introduire un Loader fourni par l'application dans la mappe de sauvegarde pour map1, par le biais de l'API ObjectGrid :

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid" );
BackingMap bm = og.defineMap( "map1" );
MyLoader loader = new MyLoader();
loader.setDataBaseName("testdb");
loader.setIsolationLevel("read committed");
bm.setLoader( loader );
```

Cet extrait de code fait la supposition que la classe MyLoader est la classe fournie par l'application qui implémente l'interface com.ibm.websphere.objectgrid.plugins.Loader. Etant donné que l'association d'un Loader avec une mappe de sauvegarde ne peut pas être modifiée postérieurement à l'initialisation d'ObjectGrid, le code doit être exécuté avant d'appeler la méthode initialize de l'interface ObjectGrid appelée. Une exception IllegalStateException se produit suite à l'appel de la méthode setLoader si cet appel est postérieur à l'initialisation.

Le Loader fourni par l'application peut avoir des propriétés définies. Dans l'exemple donné, le chargeur MyLoader est utilisé pour lire et écrire des données à partir de la table d'une base de données relationnelle. Le chargeur doit spécifier le nom de la base de données et le niveau d'isolement SQL. Le chargeur MyLoader dispose des méthodes setDataBaseName et setIsolationLevel, qui permettent à l'application de définir ces deux propriétés de Loader.

Approche XML de la connexion d'un Loader

Un Loader fourni par l'application peut être également connecté par le biais d'un fichier XML. L'exemple suivant illustre la façon de connecter le chargeur MyLoader dans la mappe de sauvegarde map1 avec le même nom de base de données et le même niveau d'isolement :

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
    <objectGrid name="grid">
        <backingMap name="map1" pluginCollectionRef="map1" lockStrategy="OPTIMISTIC" />
    </objectGrid>
</objectGrids>
<backingMapPluginCollections>
    <backingMapPluginCollection id="map1">
        <bean id="Loader" className="com.myapplication.MyLoader">
            <property name="dataBaseName"
                      type="java.lang.String"
                      value="testdb"
                      description="database name" />
            <property name="isolationLevel"
                      type="java.lang.String"
                      value="read committed"
                      description="iso level" />
        </bean>
    </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Ecriture d'un chargeur

Vous pouvez écrire votre implémentation de plug-in de chargeur dans vos applications qui doit suivre les conventions de plug-in WebSphere eXtreme Scale communes.

Ajout d'un plug-in de chargeur

L'interface du chargeur comporte la définition suivante :

```
public interface Loader
{
    static final SpecialValue KEY_NOT_FOUND;
    List get(TxID txid, List keyList, boolean forUpdate) throws LoaderException;
    void batchUpdate(TxID txid, LogSequence sequence) throws
    LoaderException, OptimisticCollisionException;
    void preloadMap(Session session, BackingMap backingMap) throws LoaderException;
}
```

Pour plus d'informations, voir les explications sur les chargeurs dans le *Présentation du produit*.

Méthode get

La mappe de sauvegarde appelle la méthode get du chargeur pour obtenir les valeurs associées à une liste de clés transmise en tant qu'argument keyList. La méthode get est requise pour renvoyer une liste de valeurs java.lang.util.List, une valeur pour chaque clé contenue dans la liste de clés. La première valeur renvoyée dans la liste de valeurs correspond à la première clé de la liste de clés, la deuxième valeur renvoyée dans la liste de valeurs correspond à la deuxième clé de la liste de clés et ainsi de suite. Si le chargeur ne trouve pas la valeur pour une clé de la liste de clés, il est requis pour renvoyer l'objet de valeur spécial KEY_NOT_FOUND défini dans l'interface du chargeur. Etant donné qu'une mappe de sauvegarde peut être configurée pour autoriser la valeur `NULL` comme étant une valeur valide, il est crucial pour le chargeur de renvoyer l'objet spécial KEY_NOT_FOUND lorsque la clé ne peut pas être détectée. Cette valeur spéciale permet à la mappe de sauvegarde de distinguer entre une valeur NULL et une valeur inexistante lorsque la clé est introuvable. Si une mappe de sauvegarde ne prend pas en charge les valeurs NULL, un chargeur renvoyant une valeur NULL et non l'objet KEY_NOT_FOUND pour une clé inexistante déclenche une exception.

L'argument forUpdate informe le chargeur si l'application a appelé une méthode get sur la mappe ou une méthode getForUpdate sur la mappe. Pour plus d'informations, reportez-vous à l' interface ObjectMap dans la documentation d'API. Le chargeur est responsable de l'implémentation d'une stratégie de contrôle des accès simultanés au stockage de persistance. Par exemple, un grand nombre de systèmes de gestion de base de données relationnelle prennent en charge la syntaxe for update sur l'instruction SQL select permettant de lire les données à partir d'une table relationnelle. Le chargeur peut choisir d'utiliser la syntaxe for update sur l'instruction SQL select en fonction de la transmission ou non de la valeur `boolean true` en que valeur d'argument pour le paramètre forUpdate de cette méthode. Le chargeur utilise généralement la syntaxe for update uniquement lorsque la stratégie de contrôle des accès simultanés pessimiste est mise en oeuvre. Dans le cas d'une stratégie de contrôle des accès simultanés optimiste, le chargeur n'utilise jamais la syntaxe `for update` sur l'instruction SQL select. Le chargeur décide d'utiliser l'argument forUpdate en fonction de la stratégie de contrôle des accès simultanés mise en oeuvre par le chargeur.

Pour obtenir une explication du paramètre txid, reportez-vous à la section «Plug-in de gestion des événements du cycle de vie des transactions», à la page 273.

Méthode batchUpdate

La méthode batchUpdate est importante dans l'interface Loader. Cette méthode est appelée dès que eXtreme Scale doit appliquer toutes les modifications en cours au chargeur. Le chargeur obtient une liste des modifications pour la mappe sélectionnée. Les modifications sont itérées et appliquées au programme d'arrière plan. La méthode reçoit la valeur TxID et les modifications à appliquer. L'exemple ci-dessous parcourt l'ensemble des modifications et traite par lots trois instructions JDBC (Java Database Connectivity), une avec insert, l'autre avec update et la dernière avec delete.

```
import java.util.Collection;
import java.util.Map;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
import com.ibm.websphere.objectgrid.plugins.Loader;
import com.ibm.websphere.objectgrid.plugins.LoaderException;
import com.ibm.websphere.objectgrid.plugins.LogElement;
import com.ibm.websphere.objectgrid.plugins.LogSequence;

    public void batchUpdate(TxID tx, LogSequence sequence) throws LoaderException {
        // Etablissez une connexion SQL à utiliser.
        Connection conn = getConnection(tx);
        try {
            // Traitez la liste des modifications et créez un ensemble d'instructions
            // préparées pour l'exécution d'une opération SQL par lots update, insert
            // ou delete.
            Iterator iter = sequence.getPendingChanges();
            while (iter.hasNext()) {
                LogElement logElement = (LogElement) iter.next();
                Object key = logElement.getKey();
                Object value = logElement.getCurrentValue();
                switch (logElement.getType().getCode()) {
                case LogElement.CODE_INSERT:
                    buildBatchSQLInsert(tx, key, value, conn);
                    break;
                case LogElement.CODE_UPDATE:
                    buildBatchSQLUpdate(tx, key, value, conn);
                    break;
                case LogElement.CODE_DELETE:
                    buildBatchSQLDelete(tx, key, conn);
                    break;
                }
            }
            // Exécutez les instructions par lots créées par la boucle au-dessus.
            Collection statements = getPreparedStatementCollection(tx, conn);
            iter = statements.iterator();
            while (iter.hasNext()) {
                PreparedStatement pstmt = (PreparedStatement) iter.next();
                pstmt.executeBatch();
            }
        } catch (SQLException e) {
            LoaderException ex = new LoaderException(e);
            throw ex;
        }
    }
```

L'exemple précédent illustre la logique de niveau supérieur du traitement de l'argument LogSequence mais les détails de la création d'une instruction SQL insert, update ou delete ne sont pas illustrés. Les principaux points illustrés sont les suivants :

- La méthode getPendingChanges est appelée sur l'argument LogSequence pour obtenir un itérateur sur la liste de LogElements que le chargeur doit traiter.
- La méthode LogElement.getType().getCode() est utilisée pour déterminer si le LogElement correspond à une opération SQL insert, update ou delete.
- Une exception SQLException est émise et est chaînée à une exception LoaderException qui signale qu'une exception est survenue lors de la mise à jour par lots.
- La prise en charge de la mise à jour par lots JDBC permet de réduire le nombre de requêtes devant être adressées au serveur principal.

Méthode preloadMap

Pendant l'initialisation d'eXtreme Scale, chaque mappe de sauvegarde définie est initialisée. Si un chargeur est relié à une mappe de sauvegarde, cette dernière appelle la méthode preloadMap dans l'interface Loader pour permettre au chargeur de préextraire des données à partir du serveur principal et de les charger dans la mappe. D'après l'exemple suivant, les 100 premières lignes de la table Employee sont lues depuis la base de données et sont chargées dans la mappe. La classe EmployeeRecord est une classe fournie par l'application contenant les données relatives à l'employé lues dans la table employee.

Remarque : Cet exemple extrait toutes les données de la base de données, puis les insère dans la mappe de base d'une partition. Dans un scénario de déploiement réparti eXtreme Scale concret, les données doivent être réparties entre toutes les partitions. Pour plus d'informations, voir «Programmation de l'utilitaire de préchargement client JPA», à la page 309.

```
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.TxID;
import com.ibm.websphere.objectgrid.plugins.Loader;
import com.ibm.websphere.objectgrid.plugins.LoaderException

    public void preloadMap(Session session, BackingMap backingMap) throws LoaderException {
        boolean tranActive = false;
        ResultSet results = null;
        Statement stmt = null;
        Connection conn = null;
        try {
            session.beginNoWriteThrough();
            tranActive = true;
            ObjectMap map = session.getMap(backingMap.getName());
            TxID tx = session.getTxID();
            // Etablissez une connexion de validation automatique définie sur
            // un niveau d'isolement lecture validée.
            conn = getAutoCommitConnection(tx);
            // Préchargez la mappe de l'employé avec les objets
            // EmployeeRecord. Lisez tous les employés de la table mais
            // limitez le préchargement aux 100 premières lignes.
            stmt = conn.createStatement();
            results = stmt.executeQuery(SELECT_ALL);
            int rows = 0;
            while (results.next() && rows < 100) {
                int key = results.getInt(EMPNO_INDEX);
                EmployeeRecord emp = new EmployeeRecord(key);
                emp.setLastName(results.getString(LASTNAME_INDEX));
                emp.setFirstName(results.getString(FIRSTNAME_INDEX));
                emp.setDepartmentName(results.getString(DEPTNAME_INDEX));
                emp.updateSequenceNumber(results.getLong(SEQNO_INDEX));
                emp.setManagerNumber(results.getInt(MGRNO_INDEX));
                map.put(new Integer(key), emp);
                ++rows;
            }
            // Validez la transaction.
            session.commit();
            tranActive = false;
        } catch (Throwable t) {
            throw new LoaderException("preload failure: " + t, t);
        } finally {
            if (tranActive) {
                try {
                    session.rollback();
                } catch (Throwable t2) {
                    // Tolérez tous les échecs d'annulation et
                    // autorisez l'émission de la classe throwable originale.
                }
            }
            // Assurez-vous de nettoyer ici les ressources des autres bases de données
            // telles que les instructions de clôture, les ensembles de résultats, etc.
        }
    }
```

Ce modèle illustre les principaux points suivants :

- La mappe de sauvegarde preloadMap utilise l'objet Session qui lui est transmis en tant qu'argument session.

- La méthode Session.beginNoWriteThrough sert à commencer la transaction à la place de la méthode begin.
- Le chargeur ne peut pas être appelé pour chaque opération put qui se produit dans cette méthode pour le chargement de la mappe.
- Le chargeur peut mapper des colonnes de la table de l'employé sur un champ de l'objet Java EmployeeRecord. Il intercepte toutes les exceptions de la classe throwable déclenchées et émet une exception LoaderException chaînée avec l'exception de la classe throwable interceptée.
- Le bloc finally veille à ce que que toutes les exceptions de la classe throwable qui sont émises entre l'appel de la méthode beginNoWriteThrough et celui de la méthode commit déclenchent l'annulation de la transaction active. Cette action s'avère cruciale pour que toutes les transactions démarrées par la méthode preloadMap soient terminées avant d'être envoyées au demandeur. Le bloc finally est l'emplacement idéal pour effectuer d'autres actions de nettoyage, notamment la fermeture de la connexion JDBC (Java Database Connectivity) et des autres objets JDBC.

L'exemple preloadMap utilise une instruction SQL Select qui sélectionne toutes les lignes de la table. A partir du chargeur de l'application, il est conseillé de définir une ou plusieurs propriétés du chargeur pour contrôler le degré de préchargement de la table dans la mappe.

Etant donné que la méthode preloadMap n'est appelée qu'une seule fois pendant l'initialisation de la mappe de sauvegarde, il s'agit également d'un bon emplacement pour exécuter le code d'initialisation unique du chargeur. Même si un chargeur choisit de ne pas préextraire de données du serveur principal et de charger les données dans la mappe, il doit probablement procéder à certaines initialisations uniques pour renforcer l'efficacité d'autres méthodes du chargeur. L'exemple suivant illustre la mise en cache des objets TransactionCallback et OptimisticCallback en tant que variables d'instance du chargeur de façon à dispenser les autres méthodes du chargeur d'effectuer des appels de méthode pour accéder à ces objets. Cette mise en cache des valeurs du plug-in ObjectGrid peut être réalisée car les objets TransactionCallback et OptimisticCallback ne peuvent être ni modifiés ni remplacés après l'initialisation de la mappe de sauvegarde. Il est acceptable de mettre en cache ces références d'objet en tant que variables d'instance du chargeur.

```
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.OptimisticCallback;
import com.ibm.websphere.objectgrid.plugins.TransactionCallback;

    // Variables d'instance du chargeur.
    MyTransactionCallback ivTcb; // MyTransactionCallback

    // étend TransactionCallback
    MyOptimisticCallback ivOcb; // MyOptimisticCallback

    // implémente OptimisticCallback
    // ...
    public void preloadMap(Session session, BackingMap backingMap) throws LoaderException
   [Replication programming]
        // Mettre en cache les objets TransactionCallback et OptimisticCallback
        // dans les variables d'instance de ce chargeur.
        ivTcb = (MyTransactionCallback) session.getObjectGrid().getTransactionCallback();
        ivOcb = (MyOptimisticCallback) backingMap.getOptimisticCallback();
        // Le reste du code preloadMap (tel qu'indiqué dans l'exemple précédent).
    }
```

Pour plus d'informations sur le préchargement et le préchargement récupérable dans le cadre du basculement de réplication, voir les informations relatives à la réplication dans la *Présentation du produit*.

Chargeurs avec mappes d'entités

Si le chargeur est relié à une mappe d'entité, il doit traiter des objets de bloc de données. Les objets de bloc de données correspondent à un format de données d'entité spécial. Le chargeur doit convertir les données entre le format de bloc de données et les autres formats de données. Par exemple, la méthode get renvoie une liste de valeurs qui correspond à l'ensemble des clés qui sont transmises à la méthode. Les clés transmises sont de type Bloc de données, à savoir des bloc de données de clés. En supposant que le chargeur conserve la mappe avec une base de données JDBC, la méthode get doit convertir chaque bloc de données de clés en une liste de valeurs d'attribut qui correspondent aux colonnes de la clé primaire de la table mappée sur la mappe d'entité, exécuter l'instruction SELECT avec la clause WHERE qui utilise les valeurs d'attribut converties comme critère d'extraction des données de la base de données, puis convertir les données renvoyées en blocs de données de valeurs. La méthode get extrait les données de la base de données et les convertit en blocs de données de valeurs pour les blocs de données de clés transmis, puis renvoie une liste des blocs de données de valeurs correspondant à l'ensemble des clés de bloc de données transmises au demandeur. La méthode get peut exécuter une instruction SELECT pour extraire toutes les données en même temps ou pour chaque bloc de données de clés. Pour obtenir des détails de programmation illustrant l'utilisation du chargeur lors du stockage des données à l'aide d'un gestionnaire d'entités, reportez-vous à la rubrique «Utilisation d'un chargeur avec des mappes d'entité et des tuples», à la page 263.

Remarques sur la programmation du chargeur JPA

Un chargeur Java Persistence API (JPA) est une implémentation de plug-in de chargeur qui utilise JPA pour interagir avec la base de données. Utilisez les considérations ci-après lorsque vous développez une application qui utilise un chargeur JPA.

Entité eXtreme Scale et entité JPA

Vous pouvez désigner une classe POJO comme une entité eXtreme Scale à l'aide d'annotations d'entité eXtreme Scale et/ou d'une configuration XML. Vous pouvez également désigner la même classe POJO comme une entité JPA à l'aide d'annotations d'entité JPA et/ou d'une configuration XML.

Entité eXtreme Scale : Une entité eXtreme Scale représente les données persistantes stockées dans des mappes ObjectGrid. Un objet d'entité est converti en un nuplet de clé et un nuplet de valeur, qui sont ensuite stockés sous forme de paires de clé/valeur dans les mappes. Un nuplet est un tableau d'attributs de primitive.

Entité JPA : Une entité JPA représente les données persistantes stockées automatiquement dans une base de données relationnelle à l'aide de la persistance gérée par conteneur. Les données sont conservées sous forme de système de stockage de données au format approprié, tel que des nuplets de base de données dans une base de données.

Lorsqu'une entité eXtreme Scale est conservée, ses relations sont stockées dans d'autres mappes d'entités. Par exemple, si vous stockez une entité Consumer avec une relation one-to-many dans une entité ShippingAddress et que l'élément cascade-persist est activé, l'entité ShippingAddress est stockée dans la mappe shippingAddress au format de nuplet. Si vous stockez une entité JPA, les entités JPA associées le sont également dans des tables de base de données lorsque

l'élément cascade-persist est activé. Lorsqu'une classe d'objet Java simple est désignée à la fois comme une entité eXtreme Scale et une entité JPA, les données peuvent être stockées à la fois dans des bases de données et des mappes d'entités ObjectGrid. Voici les scénarios les plus courants :

- **Scénario de préchargement** : Une entité est chargée à partir d'une base de données à l'aide d'un fournisseur JPA et conservée dans des mappes d'entités ObjectGrid.
- **Scénario de chargeur** : Une implémentation de chargeur est intégrée pour les mappes d'entités ObjectGrid de sorte qu'une entité stockée dans des mappes d'entités ObjectGrid puisse être stockée dans une base de données ou chargée à partir de cette dernière, à l'aide de fournisseurs JPA.

Il est également courant qu'une classe POJO soit désignée comme une entité JPA uniquement. Dans ce cas, les mappes ObjectGrid contiennent les instances d'objet Java simple au lieu des nuplets d'entité, dans le cas des entités ObjectGrid.

Considérations liées à la conception d'applications pour les mappes d'entités

Lorsque vous intégrez une implémentation dans une interface JPALoader, les instances d'objet sont directement stockées dans les mappes ObjectGrid.

Toutefois, lorsque vous intégrez une implémentation dans un plug-in JPAEntityLoader, la classe d'entité est désignée à la fois comme entité eXtreme Scale et entité JPA. Dans ce cas, traitez cette entité comme si elle possédait deux stockages de persistance : les mappes d'entités ObjectGrid et le stockage de persistance JPA. L'architecture devient plus compliquée que dans le cas de l'interface JPALoader.

Pour plus d'informations sur le plug-in JPAEntityLoader et des remarques sur la conception d'application, voir les informations sur le plug-in JPAEntityLoader, dans le *Guide d'administration*. Ces informations peuvent également vous aider si vous prévoyez d'implémenter votre propre chargeur pour les mappes d'entités.

Considérations sur les performances

Assurez-vous de bien définir le type d'extraction accélérée ou différée approprié pour les relations. Par exemple, une relation bidirectionnelle one-to-many entre Consumer et ShippingAddress, avec OpenJPA pour expliquer les différences de performances. Dans cet exemple, une requête JPA tente `select o from Consumer o where . . .` d'effectuer un chargement en bloc et charge tous les objets ShippingAddress associés. Voici une relation one-to-many définie dans la classe Consumer :

```
@Entity
public class Consumer implements Serializable {

    @OneToMany(mappedBy="consumer",cascade=CascadeType.ALL, fetch =FetchType.EAGER)
    ArrayList <ShippingAddress> addresses;
```

Voici la relation many-to-one consumer définie dans la classe ShippingAddress :

```
@Entity
public class ShippingAddress implements Serializable{

    @ManyToOne(fetch=FetchType.EAGER)
    Consumer consumer;
}
```

Si les types d'extraction des deux relations sont configurés avec la valeur `eager`, OpenJPA utilise N+1+1 requêtes pour extraire tous les objets Consumer et ShippingAddress, N représentant le nombre d'objets ShippingAddress. Toutefois, si l'adresse de livraison est modifiée pour utiliser une extraction de type différé comme ci-après, il ne faut que deux requêtes pour extraire toutes les données.

```
@Entity
public class ShippingAddress implements Serializable{

    @ManyToOne(fetch=FetchType.LAZY)
    Consumer consumer;
}
```

La requête renvoie les mêmes résultats, mais le nombre inférieur de requêtes permet de réduire considérablement l'interaction avec la base de données, ce qui peut améliorer les performances de l'application.

Plug-in JPAEntityLoader

Le plug-in JPAEntityLoader est une implémentation pré-intégrée de chargeur qui repose sur Java Persistence API (JPA) pour communiquer avec la base de données lors de l'utilisation de l'API EntityManager. Lorsque vous utilisez l'API ObjectMap, faites appel au chargeur JPALoader.

Détails du chargeur

Utilisez le plug-in JPALoader lorsque vous stockez des données à l'aide de l'API ObjectMap. Utilisez le plug-in JPAEntityLoader lorsque vous stockez des données à l'aide de l'API EntityManager.

Les chargeurs présentent deux fonctions principales :

1. **get** : dans la méthode get, le plug-in JPAEntityLoader appelle en premier la méthode javax.persistence.EntityManager.find(Class entityClass, Object key) pour trouver l'entité JPA. Le plug-in projette ensuite cette entité JPA dans les tuples d'entité. Pendant la projection, les attributs de tuple et les clés associées sont stockés dans le tuple de valeur. Après le traitement de chaque clé, la méthode get renvoie une liste de tuples de valeur d'entité.
2. **batchUpdate** : la méthode batchUpdate concerne un objet LogSequence qui contient une liste d'objets LogElement. Chaque objet LogElement contient un tuple de clé et un tuple de valeur. Pour interagir avec le fournisseur JPA, vous devez tout d'abord trouver l'entité eXtreme Scale en fonction du tuple de clé. Vous exécutez en fonction du type LogElement les appels JPA suivants :
   - **insert**: javax.persistence.EntityManager.persist(Object o)
   - **update**: javax.persistence.EntityManager.merge(Object o)
   - **remove**: javax.persistence.EntityManager.remove(Object o)

Un objet LogElement de type **update** provoque l'appel de la méthode javax.persistence.EntityManager.merge(Object o) par le plug-in JPAEntityLoader pour fusionner l'entité. Toutefois, un objet LogElement de type **update** peut résulter d'un appel com.ibm.websphere.objectgrid.em.EntityManager.merge(object o) ou d'une modification d'attribut de l'instance eXtreme Scale gérée par l'API EntityManager. Examinez l'exemple suivant :

```
com.ibm.websphere.objectgrid.em.EntityManager em = og.getSession().getEntityManager();
em.getTransaction().begin();
Consumer c1 = (Consumer) em.find(Consumer.class, c.getConsumerId());
c1.setName("New Name");
em.getTransaction().commit();
```

Dans cet exemple, un objet LogElement de type update est envoyé au plug-in JPAEntityLoader du consommateur de mappe. La méthode javax.persistence.EntityManager.merge(Object o) est appelée dans le gestionnaire d'entité JPA au lieu d'une mise à jour d'attribut dans l'entité gérée par JPA. En raison de ce changement de comportement, l'utilisation de ce modèle de programmation est sujette à certaines limitations.

Règles de conception des applications

Des relations peuvent aussi exister entre une entité et d'autres entités. La conception d'une application avec l'implication de relations et la connexion du plug-in JPAEntityLoader entraîne d'autres considérations. L'application doit suivre les quatre règles décrites dans les sections ci-dessous.

Prise en charge d'une profondeur de relations limitée

Le plug-in JPAEntityLoader est uniquement pris en charge lors de l'utilisation d'entités sans relations ou d'entités présentant des relations à un seul niveau. Ces relations à un seul niveau, telles que `Company > Department > Employee` ne sont pas prises en charge.

Un chargeur par mappe

En utilisant l'exemple des relations d'entités Consumer-ShippingAddress lors du chargement d'un consommateur avec l'extraction hâtive activée, vous pouvez charger tous les objets liés ShippingAddress. Lorsque vous conservez ou fusionnez un objet Consumer, vous pouvez conserver ou fusionner les objets liés ShippingAddress si cascade-persist ou cascade-merge est activée.

Vous ne pouvez pas connecter un chargeur pour la mappe d'entité racine qui stocke les tuples d'entité Consumer. Vous devez configurer un chargeur pour chaque mappe d'entité.

Même type de cascade pour JPA et eXtreme Scale

Réexaminez le scénario dans lequel l'entité Consumer entretient une relation un à plusieurs avec ShippingAddress. Vous pouvez examiner le scénario dans lequel cascade-persist est activée pour cette relation. Lorsqu'un objet Consumer est conservé dans eXtreme Scale, le nombre N associé d'objets ShippingAddress est également conservé dans eXtreme Scale.

Un appel persist de l'objet Consumer présentant une relation cascade-persist à l'objet ShippingAddress est convertie vers un appel de méthode javax.persistence.EntityManager.persist(consumer) et *N* appels de méthode javax.persistence.EntityManager.persist(shippingAddress) par la couche JPAEntityLoader. Toutefois, ces N appels persist supplémentaires aux objets ShippingAddress sont superflus en raison du paramètre cascade-persist du point de vue du fournisseur JPA. Pour résoudre ce problème, eXtreme Scale fournit une nouvelle méthode isCascaded sur l'interface LogElement. La méthode isCascaded indique si l'objet LogElement résulte d'une opération en cascade eXtreme Scale EntityManager. Dans cet exemple, le plug-in JPAEntityLoader de la mappe ShippingAddress reçoit *N* objets LogElement en raison des appels en cascade. Le plug-in JPAEntityLoader détecte que la méthode isCascaded renvoie la valeur true, puis l'ignore sans effectuer d'appels JPA. Par conséquent, d'un, point de vue JPA, un seul appel de méthode javax.persistence.EntityManager.persist(consumer) est reçu.

Le même comportement se présente si vous fusionnez ou supprimez une entité en mode cascade. Les opérations en cascade sont ignorées par le plug-in JPAEntityLoader.

La conception de la prise en charge en cascade consiste à relire les opérations de eXtreme Scale EntityManager dans les fournisseurs JPA. Ces opérations incluent les opérations persist, merge et remove. Pour activer le mode en cascade, vérifiez que les paramètres de cascade pour JPA et eXtreme Scale EntityManager sont identiques.

Utilisation prudente de la mise à jour d'entités

Comme décrit plus haut, la conception de la prise en charge en cascade consiste à relire les opérations de eXtreme Scale EntityManager dans les fournisseurs JPA. Si votre application appelle la méthode ogEM.persist(consumer) dans eXtreme Scale EntityManager, même les objets ShippingAddress associés sont conservés car le paramètre cascade-persist et le plug-in JPAEntityLoader appellent la méthode jpAEM.persist(consumer) dans les fournisseurs JPA.

Cependant, si votre application met à jour une entité gérée, cette mise à jour est convertie en appel merge JPA par le plug-in JPAEntityLoader. Dans ce scénario, la prise en charge de plusieurs niveaux de relations et d'associations clé n'est pas garantie. Dans ce cas, la meilleure pratique consiste à utiliser la méthode javax.persistence.EntityManager.merge(o) et non de mettre à jour une entité gérée.

Utilisation d'un chargeur avec des mappes d'entité et des tuples

Le gestionnaire d'entité convertit tous les objets entité en objets tuple avant qu'ils soient stockés dans une mappe WebSphere eXtreme Scale. Chaque entité contient un tuple de clé et un tuple de valeur. Cette paire clé-valeur est stockée dans la mappe eXtreme Scale associée pour l'entité. Lorsque vous utilisez une mappe eXtreme Scale avec un chargeur, celui-ci doit interagir avec les objets de tuple.

Présentation

eXtreme Scale comprend des plug-in de chargeur qui simplifient l'intégration aux bases de données relationnelles. Le chargeur JPA (Java Persistence API) utilise une API de persistance Java pour interagir avec la base de données et créer les objets d'entité. Les chargeurs JPA sont compatibles avec les entités eXtreme Scale.

Tuples

Un tuple contient des infirmations sur les attributs et les associations d'une entité. Les valeurs primitives sont stockées à l'aide de leurs classes wrapper primitives. D'autres types d'objets pris en charge sont stockés dans leur format natif. Les associations aux autres entités sont stockées sous la forme d'une collection d'objets de tupple de clé représentant les clés des entités cible.

Chaque attribut ou association est stocké à l'aide d'un index de base zéro. Vous pouvez extraire l'index de chaque attribut à l'aide des méthodes getAttributePosition ou getAssociationPosition. Après extraction de la position, celle-ci demeure inchangée pendant toute la durée du cycle de vie d' eXtreme Scale. La position peut changer lorsque eXtreme Scale est redémarré. Les méthodes setAttribute, setAssociation et setAssociations permettent de mettre à jour les éléments du tuple.

Avertissement : Lorsque vous créez ou mettez à jour des objets tuple, mettez à jour chaque champ primitif avec une valeur non null. Les valeurs primitives telles que `int` ne doivent pas être égales à null. Si vous ne remplacez pas la valeur par une valeur par défaut, les performances peuvent en être affectées ainsi que les champs identifiés à l'aide de l'annotation @Version ou de l'attribut version dans le fichier XML de descripteur d'entités.

L'exemple suivant explique le traitement des tuples. Pour plus d'informations sur la définition des entités pour cet exemple, reportez-vous aux détails du schéma d'entité Order disponibles dans le didacticiel du gestionnaire d'entité de la *Présentation du produit*.WebSphere eXtreme Scale est configuré pour utiliser les chargeurs avec chacune des entités. En outre, seule l'entité Order est extraite et cette entité spécifique présente une relation plusieurs à un avec l'entité Customer. Le nom d'attribut est `customer` et il présente une relation un à plusieurs avec l'entité OrderLine.

Utilisez le projecteur pour créer des objets Tuple automatiquement à partir des entités. L'utilisation du projecteur peut simplifier celle des chargeurs lorsque vous faites appel à un utilitaire de mappage relationnel objet comme Hibernate ou JPA.

order.java

```
@Entity
public class Order {
    @Id  String orderNumber;
    java.util.Date date;
    @OneToOne(cascade=CascadeType.PERSIST) Customer customer;
    @OneToMany(cascade=CascadeType.ALL, mappedBy="order") @OrderBy("lineNumber") List<OrderLine> lines;
}
```

customer.java

```
@Entity
public class Customer {
    @Id String id;
    String firstName;
    String surname;
    String address;
    String phoneNumber;
}
```

orderLine.java

```
@Entity
public class OrderLine
{
    @Id @ManyToOne(cascade=CascadeType.PERSIST) Order order;
    @Id int lineNumber;
    @OneToOne(cascade=CascadeType.PERSIST) Item item;
    int quantity;
    double price;
}
```

Une classe OrderLoader qui implémente l'interface Loader est illustrée dans le code suivant. L'exemple suivant considère qu'un plug-in TransactionCallback associé est défini.

orderLoader.java

```
public class OrderLoader implements com.ibm.websphere.objectgrid.plugins.Loader {

 private EntityMetadata entityMetaData;
 public void batchUpdate(TxID txid, LogSequence sequence)
            throws LoaderException, OptimisticCollisionException {
        ...
        }
        public List get(TxID txid, List keyList, boolean forUpdate)
```

```
            throws LoaderException {
        ...
        }
        public void preloadMap(Session session, BackingMap backingMap)
            throws LoaderException {
  this.entityMetaData=backingMap.getEntityMetadata();
 }

}
```

La variable d'instance entityMetaData est initialisée pendant l'appel de la méthode preLoadMap à partir du eXtreme Scale. La variable *entityMetaData* n'est pas égale à null si la mappe est configurée pour utiliser des entités. Dans le cas contraire, la valeur est égale à null.

Méthode batchUpdate

La méthode batchUpdate offre la fonction permettant de connaître l'action que l'application a voulu effectuer. En fonction d'une opération insert, update ou delete, une connexion peut être établie à la base de données et la tâche effectuée. La clé et les valeurs étant de type Tuple, ils doivent être transformés pour être reconnus par l'instruction SQL.

La tableORDER a été créée avec la définition DDL (Data Definition Language) ci-dessous, tel que l'illustre le code suivant :

```
CREATE TABLE ORDER (ORDERNUMBER VARCHAR(250) NOT NULL, DATE TIMESTAMP, CUSTOMER_ID VARCHAR(250))
ALTER TABLE ORDER ADD CONSTRAINT PK_ORDER PRIMARY KEY (ORDERNUMBER)
```

Le code suivant montre comment convertir un tuple dans un objet :

```
public void batchUpdate(TxID txid, LogSequence sequence)
            throws LoaderException, OptimisticCollisionException {
        Iterator iter = sequence.getPendingChanges();
        while (iter.hasNext()) {
            LogElement logElement = (LogElement) iter.next();
            Object key = logElement.getKey();
            Object value = logElement.getCurrentValue();

            switch (logElement.getType().getCode()) {
            case LogElement.CODE_INSERT:

1)              if (entityMetaData!=null) {

// La commande comporte un seul orderNumber de clé
2)                  String ORDERNUMBER=(String) getKeyAttribute("orderNumber", (Tuple) key);
// Obtenir la valeur de la date
3)                  java.util.Date unFormattedDate = (java.util.Date) getValueAttribute("date",(Tuple)value);
// Les valeurs sont 2 associations. Traitez le client car
// la table our contient customer.id comme clé primaire
4)                  Object[] keys= getForeignKeyForValueAssociation("customer","id",(Tuple) value);
                    //Commande client est M à 1. Il ne peut exister qu'1 clé
5)                  String CUSTOMER_ID=(String)keys[0];
// analysez variable unFormattedDate et formatez-la pour la base de données comme formattedDate
6)                   String formattedDate = "2007-05-08-14.01.59.780272"; // formaté pour DB2
// Enfin, l'instruction SQL suivante pour insérer l'enregistrement
7) //INSERT INTO ORDER (ORDERNUMBER, DATE, CUSTOMER_ID) VALUES(ORDERNUMBER,formattedDate, CUSTOMER_ID)
                }
                break;
            case LogElement.CODE_UPDATE:
                break;
            case LogElement.CODE_DELETE:
                break;
            }
        }

    }
// renvoie la valeur à l'attribut tel que stocké dans la clé Tuple
 private Object getKeyAttribute(String attr, Tuple key) {
        //obtenir les métadonnées clé
        TupleMetadata keyMD = entityMetaData.getKeyMetadata();
        //obtenir la position de l'attribut
        int keyAt = keyMD.getAttributePosition(attr);
        if (keyAt > -1) {
            return key.getAttribute(keyAt);
        } else { // attribut non défini
            throw new IllegalArgumentException("Invalid position index for  "+attr);
        }

    }
// renvoie la valeur à l'attribut tel que stocké dans la valeur Tuple
```

```
    private Object getValueAttribute(String attr, Tuple value) {
        //semblable à ci-dessus à la différence que nous travaillons avec des métadonnées de valeur
        TupleMetadata valueMD = entityMetaData.getValueMetadata();

        int keyAt = valueMD.getAttributePosition(attr);
        if (keyAt > -1) {
            return value.getAttribute(keyAt);
        } else {
            throw new IllegalArgumentException("Invalid position index for  "+attr);
        }
    }
// renvoie un tableau de clés qui se réfèrent à l'association.
    private Object[] getForeignKeyForValueAssociation(String attr, String fk_attr, Tuple value) {
        TupleMetadata valueMD = entityMetaData.getValueMetadata();
        Object[] ro;

        int customerAssociation = valueMD.getAssociationPosition(attr);
        TupleAssociation tupleAssociation = valueMD.getAssociation(customerAssociation);

        EntityMetadata targetEntityMetaData = tupleAssociation.getTargetEntityMetadata();

        Tuple[] customerKeyTuple = ((Tuple) value).getAssociations(customerAssociation);

        int numberOfKeys = customerKeyTuple.length;
        ro = new Object[numberOfKeys];

        TupleMetadata keyMD = targetEntityMetaData.getKeyMetadata();
        int keyAt = keyMD.getAttributePosition(fk_attr);
        if (keyAt < 0) {
            throw new IllegalArgumentException("Invalid position index for  " + attr);
        }
        for (int i = 0; i < numberOfKeys; ++i) {
            ro[i] = customerKeyTuple[i].getAttribute(keyAt);
        }

        return ro;

    }
```

1. Vérifiez que entityMetaData n'est pas égal à null, ce qui implique que la clé et les entrées de cache de valeur sont de type Tuple. A partir de entityMetaData, Key TupleMetaData est extrait, ce qui reflète uniquement la partie clé des métadonnées Order.
2. Traitez le KeyTuple et obtenez la valeur de Key Attribute orderNumber
3. Traitez le ValueTuple et obtenez la valeur de la date d'attribut
4. Traitez le ValueTuple et obtenez la valeur de Keys from association customer
5. Extrayez CUSTOMER_ID. En fonction de la relation, une commande peut comporter un seul client, à savoir une seule clé. C'est pourquoi, la taille des clés est de 1. Pour vous simplifer la tâche, nous avons ignoré l'anayse de la date au format correct.
6. Etant donné qu'il s'agit d'une opération insert, l'instruction SQL est transmise dans la connexion de source de données pour effectuer l'opération insert.

La démarcation de transaction et l'accès à la base de données sont traités dans la rubrique «Ecriture d'un chargeur», à la page 255.

Méthode get

Si la clé n'est pas trouvée dans le cache, appelez la méthode get dans le plug-in de chargeur pour trouver la clé.

La clé est un tuple. La première étape à effectuer consiste à convertir le tuple en valeurs primitives pouvant être transmises vers l'instruction SQL SELECT. Une fois que tous les attributs sont extraits de la base de données, vous devez les convertir en tuples. Le code ci-dessous illustre la classe Order.

```
public List get(TxID txid, List keyList, boolean forUpdate) throws LoaderException {
  System.out.println("OrderLoader: Get called");
  ArrayList returnList = new ArrayList();

1)  if (entityMetaData != null) {
   int index=0;
   for (Iterator iter = keyList.iterator(); iter.hasNext();) {
2)    Tuple orderKeyTuple=(Tuple) iter.next();
```

```
    // La commande comporte un seul orderNumber de clé
3)    String ORDERNUMBERKEY = (String) getKeyAttribute("orderNumber",orderKeyTuple);
    //Nous devons exécuter une requête pour obtenir les valeurs de
4)    // SELECT CUSTOMER_ID, date FROM ORDER WHERE ORDERNUMBER='ORDERNUMBERKEY'

5)    //1) Clé externe : CUSTOMER_ID
6)    //2) date
    // En supposant que les deux sont renvoyés en tant que
7)                              String  CUSTOMER_ID = "C001";
                        // En prenant en compte les attributs extraits et initialisés
8)    java.util.Date retrievedDate = new java.util.Date();
                              // En supposant que cette date reflète celle de la base de données

    // Nous devons désormais convertir ces données en un tuple avant de les renvoyer

    //créer un tuple de valeur
9)    TupleMetadata valueMD = entityMetaData.getValueMetadata();
    Tuple valueTuple=valueMD.createTuple();


    //ajouter l'objet retrievedDate au tuple
    int datePosition = valueMD.getAttributePosition("date");
10)    valueTuple.setAttribute(datePosition, retrievedDate);

    //Nous devons ensuite ajouter l'association
11)    int customerPosition=valueMD.getAssociationPosition("customer");
    TupleAssociation customerTupleAssociation =
                                  valueMD.getAssociation(customerPosition);
    EntityMetadata customerEMD = customerTupleAssociation.getTargetEntityMetadata();
    TupleMetadata customerTupleMDForKEY=customerEMD.getKeyMetadata();
12)    int customerKeyAt=customerTupleMDForKEY.getAttributePosition("id");


    Tuple customerKeyTuple=customerTupleMDForKEY.createTuple();
    customerKeyTuple.setAttribute(customerKeyAt, CUSTOMER_ID);
13)    valueTuple.addAssociationKeys(customerPosition, new Tuple[] {customerKeyTuple});


14)    int linesPosition = valueMD.getAssociationPosition("lines");
    TupleAssociation linesTupleAssociation = valueMD.getAssociation(linesPosition);
    EntityMetadata orderLineEMD = linesTupleAssociation.getTargetEntityMetadata();
    TupleMetadata orderLineTupleMDForKEY = orderLineEMD.getKeyMetadata();
    int lineNumberAt = orderLineTupleMDForKEY.getAttributePosition("lineNumber");
    int orderAt = orderLineTupleMDForKEY.getAssociationPosition("order");

    if (lineNumberAt < 0 || orderAt < 0) {
     throw new IllegalArgumentException(
                                      "Invalid position index for lineNumber or order "+
                                      lineNumberAt + " " + orderAt);
    }
15) // SELECT LINENUMBER FROM ORDERLINE WHERE ORDERNUMBER='ORDERNUMBERKEY'
    // En supposant que deux lignes de numéro de ligne sont renvoyées avec les valeurs 1 et 2

    Tuple orderLineKeyTuple1 = orderLineTupleMDForKEY.createTuple();
    orderLineKeyTuple1.setAttribute(lineNumberAt, new Integer(1));// set Key
    orderLineKeyTuple1.addAssociationKey(orderAt, orderKeyTuple);

    Tuple orderLineKeyTuple2 = orderLineTupleMDForKEY.createTuple();
    orderLineKeyTuple2.setAttribute(lineNumberAt, new Integer(2));// Init Key
    orderLineKeyTuple2.addAssociationKey(orderAt, orderKeyTuple);

16)    valueTuple.addAssociationKeys(linesPosition, new Tuple[]
                                    {orderLineKeyTuple1, orderLineKeyTuple2 });

    returnList.add(index, valueTuple);

    index++;

   }
  }else {
   // ne prend pas en charge les tuples
  }
  return returnList;
 }
```

1. La méthode get est appelée lorsque le cache ObjectGrid n'a pas pu trouver la clé et demande au chargeur d'extraire. Testez pour la valeur entityMetaData et procédez si non égale à null.
2. La liste keyList contient des tuples.
3. Extrayez la valeur de l'attribut orderNumber.
4. Exécutez la requête pour extraire la date (valeur) et l'ID client (clé externe).
5. CUSTOMER_ID est une clé externe qui doit être définie dans le tuple d'association.

6. La date est une valeur et doit être déjà définie.
7. Etant donné que cet exemple ne met pas en oeuvre des appels JDBC, CUSTOMER_ID est utilisé par défaut.
8. Etant donné que cet exemple ne met pas en oeuvre des appels JDBC, la date est utilisée par défaut.
9. Créez la valeur Tuple.
10. Définissez la valeur de la date dans le tuple en fonction de sa position.
11. L'entité Order comporte deux associations. Commencez par l'attribut client qui se réfère à l'entité client. Vous devez définir la valeur ID dans le tuple.
12. Trouvez la position de la valeur ID dans l'entité client.
13. Définissez les valeurs des clés d'association uniquement.
14. De même, les lignes constituent des associations qui peuvent être configurées comme groupe de clés d'association, de la même manière que pour l'association client.
15. Etant donné que vous devez définir les clés pour la lineNumber associée à cette entité Order, exécutez l'instruction SQL pour extraire les valeurs lineNumber.
16. Configurez les clés d'association dans la valueTuple. La création du tuple renvoyé à la mappe de sauvegarde est ainsi achevée.

Cette rubrique présente les étapes de création des tuples et une description de l'entité Order uniquement. Effectuez les mêmes étapes pour les autres entités et le processus complet lié au plug-in TransactionCallback. Pour plus de détails, reportez-vous à la rubrique «Plug-in de gestion des événements du cycle de vie des transactions», à la page 273.

Ecriture d'un chargeur avec un contrôleur de préchargement de fragments répliques

Un chargeur avec un contrôleur de préchargement de fragments répliques est un chargeur qui implémente l'interface ReplicaPreloadController en plus de l'interface Loader.

Présentation

L'interface ReplicaPreloadController est conçue pour permettre à une réplique qui devient un fragment primaire de savoir si le fragment primaire précédent a effectué le processus de préchargement. Si le préchargement est partiellement effectué, les informations permettant de reprendre le fragment primaire précédent là où il s'est arrêté sont fournies. Avec l'implémentation de l'interfaceReplicaPreloadController, une réplique qui devient un fragment primaire continue le processus de préchargement là où s'est arrêté le fragment primaire précédent et termine le préchargement complet.

Dans un environnement WebSphere eXtreme Scale réparti, une mappe peut comporter des fragments répliques et peut précharger un volume important de données pendant l'initialisation. Le préchargement est une activité du chargeur et a lieu uniquement dans la mappe principale lors de l'initialisation. Le préchargement peut prendre un certain temps si un volume important de données sont préchargées. Si la mappe principale a terminé une grande partie des données de préchargement mais si elle s'interrompt pour une raison inconnue pendant l'initialisation, une réplique devient un fragment primaire. Dans ce cas, les données de préchargement qui ont été traitées par le fragment primaire précédent sont perdues car le nouveau fragment primaire effectue normalement un préchargement

inconditionnel. Avec un préchargement inconditionnel, le nouveau fragment primaire redémarre le processus de préchargement depuis le début et les données de préchargement précédentes sont ignorées. Si vous voulez que le nouveau fragment primaire reprenne là où le fragment primaire précédent s'est arrêté lors du préchargement, utilisez un chargeur qui implémente l'interface ReplicaPreloadController. Pour plus d'informations, consultez la documentation d'API.

Pour en savoir plus sur les chargeurs, consultez les informations sur les chargeurs dans le *Présentation du produit*. Si l'écriture d'un plug-in Loader traditionnel vous intéresse, reportez-vous à la rubrique «Ecriture d'un chargeur», à la page 255.

L'interface ReplicaPreloadController comporte la définition suivante :

```
public interface ReplicaPreloadController
{
    public static final class Status
    {
        static public final Status PRELOADED_ALREADY = new Status(K_PRELOADED_ALREADY);
        static public final Status FULL_PRELOAD_NEEDED = new Status(K_FULL_PRELOAD_NEEDED);
        static public final Status PARTIAL_PRELOAD_NEEDED = new Status(K_PARTIAL_PRELOAD_NEEDED);
    }

    Status checkPreloadStatus(Session session, BackingMap bmap);
}
```

Les sections suivantes traitent de certaines des méthodes des interfaces Loader et ReplicaPreloadController.

Méthode checkPreloadStatus

Lorsqu'un chargeur implémente l'interface ReplicaPreloadController, la méthode checkPreloadStatus est appelée avant la méthode preloadMap pendant l'initialisation de la mappe. L'état de retour de cette méthode détermine si la méthode preloadMap est appelée. Si cette méthode renvoie`Status#PRELOADED_ALREADY`, la méthode de préchargement n'est pas appelée. Dans le cas contraire, la méthode preload est exécutée. En raison de ce comportement, cette méthode doit servir de méthode d'initialisation du chargeur. Vous devez initialiser les propriétés du chargeur dans cette méthode. Celle-ci doit renvoyer l'état correct ou le préchargement risque de ne pas se dérouler comme prévu.

```
public Status checkPreloadStatus(Session session, BackingMap backingMap) {
        // Lorsqu'un chargeur implémente l'interface ReplicaPreloadController,
    // cette méthode est appelée avant la méthode preloadMap
    // pendant l'initialisation de la mappe. Que la méthode preloadMap soit ou non appelée
    // dépend du statut retourné par cette méthode. Cette méthode
    // sert donc également de méthode d'initialisation du chargeur. Elle doit
    // retourner le statut exact sous peine de compromettre le préchargement.


        // Remarque : doit initialiser cette instance de chargeur ici.
        ivOptimisticCallback = backingMap.getOptimisticCallback();
        ivBackingMapName = backingMap.getName();
        ivPartitionId = backingMap.getPartitionId();
        ivPartitionManager = backingMap.getPartitionManager();
        ivTransformer = backingMap.getObjectTransformer();
        preloadStatusKey = ivBackingMapName + "_" + ivPartitionId;

        try {
            // l'on obtient la preloadStatusMap pour enregistrer le statut du préchargement
      // qui a pu être défini par d'autres machines virtuelles Java.
            ObjectMap preloadStatusMap = session.getMap(ivPreloadStatusMapName);

            // extrayez le dernier index de bloc de données de préchargement enregistré.
            Integer lastPreloadedDataChunk = (Integer) preloadStatusMap.get(preloadStatusKey);

            if (lastPreloadedDataChunk == null) {
                preloadStatus = Status.FULL_PRELOAD_NEEDED;
```

```
            } else {
                preloadedLastDataChunkIndex = lastPreloadedDataChunk.intValue();
                if (preloadedLastDataChunkIndex == preloadCompleteMark) {
                    preloadStatus = Status.PRELOADED_ALREADY;
                } else {
                    preloadStatus = Status.PARTIAL_PRELOAD_NEEDED;
                }
            }

            System.out.println("TupleHeapCacheWithReplicaPreloadControllerLoader.
checkPreloadStatus()
     -> map = " + ivBackingMapName + ", preloadStatusKey = " + preloadStatusKey
                    + ", retrieved lastPreloadedDataChunk =" + lastPreloadedDataChunk + ",
         determined preloadStatus = "
                    + getStatusString(preloadStatus));

        } catch (Throwable t) {
            t.printStackTrace();
        }

        return preloadStatus;
    }
```

Méthode preloadMap

L'exécution de la méthode preloadMap dépend des résultats renvoyés par la méthode checkPreloadStatus. Si la méthode preloadMap est appelée, elle doit généralement extraire les informations sur l'état du préchargement de la mappe d'état de préchargement spécifiée et déterminer la procédure à suivre. Idéalement, la méthode preloadMap doit savoir si le préchargement est partiellement effectué et exactement où il doit démarrer. Pendant le préchargement des données, la méthode preloadMap doit mettre à jour l'état de préchargement dans la mappe d'état de préchargement spécifiée. L'état de préchargement stocké dans la mappe d'état de préchargement est extrait par la méthode checkPreloadStatus lorsqu'elle doit vérifier l'état de préchargement.

```
public void preloadMap(Session session, BackingMap backingMap)
     throws LoaderException {
        EntityMetadata emd = backingMap.getEntityMetadata();
        if (emd != null && tupleHeapPreloadData != null) {
            // La méthode getPreLoadData est similaire à l'extraction des données
      // depuis la base de données. Ces données sont envoyées dans le cache dans le cadre
      // du préchargement.
            ivPreloadData = tupleHeapPreloadData.getPreLoadData(emd);

            ivOptimisticCallback = backingMap.getOptimisticCallback();
            ivBackingMapName = backingMap.getName();
            ivPartitionId = backingMap.getPartitionId();
            ivPartitionManager = backingMap.getPartitionManager();
            ivTransformer = backingMap.getObjectTransformer();
            Map preloadMap;

            if (ivPreloadData != null) {
                try {
                    ObjectMap map = session.getMap(ivBackingMapName);

                    // obtenez la  preloadStatusMap pour enregistrer l'état de préchargement.
                    ObjectMap preloadStatusMap = session.getMap(ivPreloadStatusMapName);

                    // Remarque : lorsque cette méthode preloadMap est appelée,
         // checkPreloadStatus a été appelé, Both preloadStatus
         // et preloadedLastDataChunkIndex ont tous les deux été définis. Et
         // preloadStatus doit être soit PARTIAL_PRELOAD_NEEDED,
         // soit FULL_PRELOAD_NEEDED qui requiert à nouveau un préchargement.

                    // En cas de volume important de données à précharger, les données sont
    // généralement divisés en blocs et le préchargement
         // traite chacun de ces blocs au sein de son propre service de transaction répartie. Ici,
         // nous nous contentons de précharger quelques entrées censées représenter un bloc.
 // de sorte que le préchargement traite une entrée dans un service de transaction
         // répartie pour simuler le préchargement par blocs.

                    Set entrySet = ivPreloadData.entrySet();
                    preloadMap = new HashMap();
                    ivMap = preloadMap;
```

```
          // dataChunkIndex représente le bloc de données
// en cours de traitement
          int dataChunkIndex = -1;
          boolean shouldRecordPreloadStatus = false;
          int numberOfDataChunk = entrySet.size();
          System.out.println("    numberOfDataChunk to be preloaded = "
  + numberOfDataChunk);

          Iterator it = entrySet.iterator();
          int whileCounter = 0;
          while (it.hasNext()) {
              whileCounter++;
              System.out.println("preloadStatusKey = " + preloadStatusKey + " ,
   whileCounter = " + whileCounter);

              dataChunkIndex++;

              // si dataChunkIndex <= preloadedLastDataChunkIndex
              // pas besoin de traiter car il a été préchargé auparavant
  // par une autre machine virtuelle Java. Uniquement besoin de traiter dataChunkIndex
  // > preloadedLastDataChunkIndex
              if (dataChunkIndex <= preloadedLastDataChunkIndex) {
                  System.out.println("ignore current dataChunkIndex =
      " + dataChunkIndex + " that has been previously
      preloaded.");
                  continue;
              }

              // Remarque : cet exemple simule un bloc de données comme entrée.
              // chaque clé représente un bloc de données à des fins de clarté.
              // Si le serveur primaire ou le fragment primaire s'arrête pour une
              // raison inconnue, l'état du préchargement qui
   // le statut du préchargement qui indique l'avancement
   // du préchargement doit être disponible dans preloadStatusMap. Un
  // fragment réplique qui devient un fragment primaire peut obtenir le statut
               // du préchargement et déterminer comment relancer le
  // et déterminer comment faire à nouveau pour précharger.
              // Remarque : l'enregistrement du statut
              // du préchargement doit être dans le même
   // service de transaction répartie que le positionnement des données dans le cache,
              // ce qui fait que si ce service
  // est annulé ou échoue, le statut enregistré est
  // le statut réel.

              Map.Entry entry = (Entry) it.next();
              Object key = entry.getKey();
              Object value = entry.getValue();
              boolean tranActive = false;

              System.out.println("processing data chunk. map = " +
    this.ivBackingMapName + ", current dataChunkIndex = " +
    dataChunkIndex + ", key = " + key);

              try {
                  shouldRecordPreloadStatus = false; // redéfinir sur false
                  session.beginNoWriteThrough();
                  tranActive = true;

                  if (ivPartitionManager.getNumOfPartitions() == 1) {
                      // si uniquement 1 partition, pas besoin de s'occuper
      // de la partition.
                      // simplement envoyer les données dans le cache
                      map.put(key, value);
                      preloadMap.put(key, value);
                      shouldRecordPreloadStatus = true;
                  } else if (ivPartitionManager.getPartition(key) == ivPartitionId) {
                      // si la mappe est partitionnée, il faut prendre en compte
      // la clé de partition ne télécharger que les données appartenant
      // à cette partition.
                      map.put(key, value);
                      preloadMap.put(key, value);
                      shouldRecordPreloadStatus = true;
                  } else {
                      // ignorer cette entrée car elle n'appartient pas
      // à cette partition.
                  }

                  if (shouldRecordPreloadStatus) {
                      System.out.println("record preload status. map = " +
```

```
                    this.ivBackingMapName + ", preloadStatusKey = " +
                    preloadStatusKey + ", current dataChunkIndex ="
                    + dataChunkIndex);
                                if (dataChunkIndex == numberOfDataChunk) {
                                    System.out.println("record preload status. map = " +
                       this.ivBackingMapName + ", preloadStatusKey = " +
                      preloadStatusKey + ", mark complete =" +
                      preloadCompleteMark);
                       // signifie qu'il s'agit du dernier bloc de données, si validation
                   // réussie, le préchargement de l'enregistrement est effectué.
                   // à ce stade, le préchargement est considéré comme effectué
                        // utilisez -99 comme marque spéciale pour l'état de préchargement terminé.

                                    preloadStatusMap.get(preloadStatusKey);

                                    // a put follow a get devient un update si le get
                    // renvoie un objet, sinon, il devient insert.
                                    preloadStatusMap.put(preloadStatusKey,
new Integer(preloadCompleteMark));

                                } else {
                                    // enregistre le dataChunkIndex préchargé actuel
                    // dans preloadStatusMap a put follow a get devient un update si le get
                    // renvoie un objet, sinon, devient
                    // insert.
                                    preloadStatusMap.get(preloadStatusKey);
                                    preloadStatusMap.put(preloadStatusKey, new Integer(dataChunkIndex));
                                }
                            }

                            session.commit();
                            tranActive = false;

                            // pour simuler le préchargement d'une grande quantité de données
                            // mettez cette unité d'exécution en veille pour 30 sec.
                            // L'app réelle ne doit PAS mettre cette unité d'exécution en veille
                            Thread.sleep(10000);

                        } catch (Throwable e) {
                            e.printStackTrace();
                            throw new LoaderException("preload failed with exception: " + e, e);
                        } finally {
                            if (tranActive && session != null) {
                                try {
                                    session.rollback();
                                } catch (Throwable e1) {
                                    // préchargement ignorant l'exception de l'annulation
                                }
                            }
                        }
                    }

                    // à ce stade, le préchargement est considéré comme devant être impérativement effectué
                    // utilisez -99 comme marque spéciale pour l'état de préchargement terminé.
                    // cela garantit que la marque complète est définie.
                    // en outre, lors du partitionnement, chaque partition ne sait pas
            // quand il s'agit de son dernier bloc de données. Le bloc suivant sert donc
            // de rapport de statut terminé pour l'ensemble du chargement.
                        System.out.println("Overall preload status complete -> record preload status.
map = " + this.ivBackingMapName + ",
                     preloadStatusKey = " + preloadStatusKey + ", mark complete =" +
                    preloadCompleteMark);
                        session.begin();
                        preloadStatusMap.get(preloadStatusKey);
                        // a put follow a get devient une mise à jour si le get renvoie un objet,
                        // autrement, il devient insert.
                        preloadStatusMap.put(preloadStatusKey, new Integer(preloadCompleteMark));
                        session.commit();

                        ivMap = preloadMap;
                    } catch (Throwable e) {
                        e.printStackTrace();
                        throw new LoaderException("preload failed with exception: " + e, e);
                    }
                }
            }
        }
```

Mappe d'état de préchargement

Vous devez utiliser une mappe d'état de préchargement pour prendre en charge l'implémentation de l'interface ReplicaPreloadController. La méthode preloadMap doit toujours commencer par vérifier l'état du préchargement stocké dans la mappe d'état de préchargement et le mettre à jour dès qu'elle envoie des données dans le cache. La méthode checkPreloadStatus peut extraire l'état de préchargement de la mappe d'état de préchargement, déterminer l'état de préchargement et renvoyer l'état à son demandeur. La mappe d'état de préchargement doit se situer dans le même groupe de mappaes que les autres mappes dotées de chargeurs de contrôleur de préchargement de fragments répliques.

Plug-in de gestion des événements du cycle de vie des transactions

Le plug-in TransactionCallback permet de personnaliser les opérations de vérification et de comparaison des versions des objets du cache lorsqu'on utilise la stratégie de verrouillage optimiste.

Il est possible de fournir un objet connectable de rappel optimiste qui implémente l'interface com.ibm.websphere.objectgrid.plugins.OptimisticCallback. Pour les mappes d'entités, un plug-in OptimisticCallback hautes performances est automatiquement configuré.

Utilité

L'interface OptimisticCallback permet de fournir des opérations de comparaison optimiste entre les valeurs d'une mappe. Une implémentation d'OptimisticCallback est nécessaire lorsqu'on utilise la stratégie de verrouillage optimiste. WebSphere eXtreme Scalefournit une implémentation par défaut d'OptimisticCallback. Mais, en principe, c'est à l'application de connecter sa propre implémentation de cette interface. Pour plus d'informations, voir les informations concernant les stratégies de verrouillage dans le manuel *Présentation du produit*.

Implémentation par défaut

La structure eXtreme Scale fournit une implémentation par défaut de l'interface OptimisticCallback qui est utilisée si l'application ne connecte pas d'objet OptimisticCallback fourni par ses soins, comme on l'a vu à la précédente section. L'implémentation par défaut retourne toujours la valeur spéciale NULL_OPTIMISTIC_VERSION comme objet version de la valeur et elle n'actualise jamais cet objet version. Cette action fait de la comparaison optimiste une fonction "no operation". Dans la plupart des cas, l'on ne souhaite pas que se produise une fonction "no operation" lorsqu'on utilise la stratégie de verrouillage optimiste. Vos applications doivent implémenter l'interface OptimisticCallback et connecter leurs propres implémentations OptimisticCallback afin de ne pas utiliser l'implémentation par défaut. Mais il existe au moins un scénario où l'implémentation OptimisticCallback fournie par défaut a toute son utilité. Prenons le cas de figure suivant :

- Un loader est connecté pour la mappe de sauvegarde.
- Le loader sait comment effectuer la comparaison optimiste sans l'assistance d'un plug-in OptimisticCallback.

Comment le loader peut-il effectuer une vérification optimiste des versions sans l'assistance d'un objet OptimisticCallback ? Le loader connaît l'objet de classe value

et il sait quel champ de l'objet value est utilisé comme valeur optimiste de version. Supposons, par exemple, que l'interface suivante soit utilisée pour l'objet value de la mappe Employee :

```
public interface Employee
{
    // Sequential sequence number used for optimistic versioning.
    public long getSequenceNumber();
    public void setSequenceNumber(long newSequenceNumber);
    // Other get/set methods for other fields of Employee object.
}
```

Dans cet exemple, le loader sait qu'il peut utiliser la méthode getSequenceNumber pour obtenir la version actuelle d'un objet value Employee. Le loader incrémente la valeur retournée pour générer un nouveau numéro de version avant d'actualiser le stockage de persistance avec la nouvelle valeur d'Employee. Dans le cas d'un loader JDBC (Java Database Connectivity), c'est le numéro actuel de séquence dans la clause where d'une instruction SQL update surqualifiée qui est utilisé et le loader utilise le nouveau numéro de séquence qui est généré pour définir la colonne des numéros de séquence.

Autre possibilité : le loader recourt à une certaine fonction fournie en dorsal, qui actualise automatiquement une colonne masquée utilisables pour la vérification optimiste des versions. Dans certains cas, une procédure stockée ou déclencheur peut être utilisée pour aider à maintenir une colonne contenant les informations de version. Si le loader utilise l'une de ces techniques pour maintenir des informations optimistes de version, l'application n'aura pas besoin de fournir d'implémentation d'OptimisticCallback. L'implémentation par défaut d'OptimisticCallback est utilisable dans ce scénario car le loader peut vérifier les versions de manière optimiste sans l'assistance d'un objet OptimisticCallback.

Implémentation par défaut des entités

Les entités sont stockées dans l'ObjectGrid à l'aide d'objets tuple. Le comportement de l'implémentation par défaut d'OptimisticCallback est semblable à celui des mappes de non-entités. Mais le champ version dans l'entité est identifié à l'aide de l'annotation @Version ou de l'attribut version dans le fichier XML de descripteur d'entités.

L'attribut version peut être de l'un des types suivants : int, Integer, short, Short, long, Long ou java.sql.Timestamp. Une entité ne doit avoir qu'un seul attribut version défini. L'attribut version ne doit être défini que pendant la construction. Une fois l'entité persistante, la valeur de l'attribut version ne doit pas être modifiée.

Si un attribut version n'est pas configuré et que l'on utilise la stratégie de verrouillage optimiste, le tuple tout entier est versionné en utilisant son état tout entier.

Dans l'exemple qui suit, l'entité Employee a un attribut long de version nommé SequenceNumber :

```
@Entity
public class Employee {
private long sequence;
    // Sequential sequence number used for optimistic versioning.
    @Version
    public long getSequenceNumber() {
        return sequence;
    }
```

```
    public void setSequenceNumber(long newSequenceNumber) {
        this.sequence = newSequenceNumber;
    }
    // Other get/set methods for other fields of Employee object.
}
```

Ecrire une implémentation d'OptimisticCallback

Une implémentation d'OptimisticCallback doit implémenter l'interface OptimisticCallback et se conformer aux conventions habituelles des plug-in ObjectGrid.

La liste suivante décrit ou explique chacune des méthodes de l'interface OptimisticCallback :

NULL_OPTIMISTIC_VERSION

Cette valeur spéciale est retournée par la méthode getVersionedObjectForValue si c'est l'implémentation par défaut d'OptimisticCallback qui est utilisée au lieu d'une implémentation fournie par une application.

Méthode getVersionedObjectForValue

La méthode getVersionedObjectForValue peut retourner une copie de la valeur ou d'un attribut de la valeur, qui peut être utilisée à des fins de vérification des versions. Cette méthode est appelée chaque fois qu'un objet est associé à une transaction. Lorsqu'aucune API Loader n'est connectée à une mappe de sauvegarde, cette dernière utilise cette valeur au moment de la validation afin d'effectuer une comparaison optimiste des versions. La comparaison optimiste des versions est utilisée par la mappe de sauvegarde pour s'assurer que la version n'a pas changé depuis le premier accès de cette transaction à l'entrée de mappe qui a été modifiée par cette transaction. Si entre-temps une autre transaction a modifié la version de cette entrée de mappe, la comparaison échoue et la mappe de sauvegarde affiche une exception OptimisticCollisionException pour forcer la transaction à s'annuler. Si une API Loader est connectée, la mappe de sauvegarde n'utilise pas les informations de vérification optimiste des versions. Car c'est à l'API Loader d'effectuer cette comparaison optimiste et d'actualiser les informations de version quand c'est nécessaire. Normalement, l'API Loader obtient l'objet de version initial du LogElement transmis à sa méthode batchUpdate, laquelle méthode est appelée lorsqu'une opération de vidage se produit ou lorsqu'une transaction est validée.

Le code suivant montre comment l'objet EmployeeOptimisticCallbackImpl utilise l'implémentation :

```
public Object getVersionedObjectForValue(Object value)
{
    if (value == null)
    {
        return null;
    }
    else
    {
        Employee emp = (Employee) value;
        return new Long( emp.getSequenceNumber() );
    }
}
```

Comme le montre l'exemple ci-dessus, l'attribut sequenceNumber est retourné dans un objet java.lang.Long object, attendu par l'API Loader, ce qui implique que la personne qui a écrit l'API soit a écrit l'implémentation EmployeeOptimisticCallbackImpl, soit a collaboré étroitement avec celle qui a implémenté EmployeeOptimisticCallbackImpl. Dans le dernier cas, ces deux personnes se sont mises d'accord sur la valeur qui est retournée par la méthode getVersionedObjectForValue. Comme on l'a vu plus haut, l'implémentation par défaut d'OptimisticCallback retourne la valeur spéciale NULL_OPTIMISTIC_VERSION en tant qu'objet version.

Méthode updateVersionedObjectForValue

Cette méthode est appelée chaque fois qu'une transaction a actualisé une valeur et que l'on a besoin d'un nouvel objet versionné. Si la méthode getVersionedObjectForValue retourne un attribut de la valeur, cette méthode actualise normalement la valeur de l'attribut avec un nouvel objet version. Si la méthode getVersionedObjectForValue retourne une copie de la valeur, en principe, cette méthode n'effectue aucune mise à jour. Avec cette méthode, l'implémentation par défaut d'OptimisticCallback n'effectue aucune mise à jour car l'implémentation par défaut de getVersionedObjectForValue retourne la valeur spéciale NULL_OPTIMISTIC_VERSION en tant qu'objet version. L'exemple qui suit montre l'implémentation utilisée par l'objet EmployeeOptimisticCallbackImpl qui est utilisé dans la section OptimisticCallback :

```
public void updateVersionedObjectForValue(Object value)
{
    if ( value != null )
    {
        Employee emp = (Employee) value;
        long next = emp.getSequenceNumber() + 1;
        emp.updateSequenceNumber( next );
    }
}
```

Comme le montre l'exemple ci-dessus, l'attribut sequenceNumber s'incrémente de un afin que, lors du prochain appel de la méthode getVersionedObjectForValue, la valeur java.lang.Long qui est retournée soit une valeur de type long, égale à celle du numéro de séquence d'origine. Plus un, par exemple, sera la version suivante de cette instance d'Employee. Là encore, cette exemple implique que la personne qui a écrit l'API soit a écrit l'implémentation EmployeeOptimisticCallbackImpl, soit a collaboré étroitement avec celle qui a implémenté cette interface.

Méthode serializeVersionedValue

Cette méthode écrit dans le flux spécifié la valeur versionnée. Selon l'implémentation, la valeur versionnée peut être utilisée pour identifier les collisions de mises à jour optimistes. Dans certaines implémentations, la valeur versionnée est une copie de la valeur d'origine. D'autres implémentations peuvent avoir un numéro de séquence ou un autre objet pur indiquer la version de la valeur. Comme l'implémentation effective est inconnue, cette méthode est fournie pour effectuer la sérialisation appropriée. L'implémentation par défaut appelle la méthode writeObject.

Méthode inflateVersionedValue

Cette méthode prend la version sérialisée de la valeur versionnée et elle retourne l'objet effectif de la valeur versionnée. Selon l'implémentation, la valeur versionnée peut être utilisée pour identifier les collisions de mises à jour optimistes. Dans

certaines implémentations, la valeur versionnée est une copie de la valeur d'origine. D'autres implémentations peuvent avoir un numéro de séquence ou un autre objet pur indiquer la version de la valeur. Comme l'implémentation effective est inconnue, cette méthode est fournie pour effectuer la désérialisation appropriée. L'implémentation par défaut appelle la méthode readObject.

Utiliser une implémentation d'OptimisticCallback fournie par une application

Deux approches permettent d'ajouter un OptimisticCallback dans la configuration BackingMap : la configuration XML et la configuration par programmation.

Configuration par programmation d'une implémentation d'OptimisticCallback

L'exemple qui suit montre comment une application peut par programmation connecter un objet OptimisticCallback pour la mappe de sauvegarde Employee dans l'instance ObjectGrid locale grid1 :

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid1" );
BackingMap bm = dg.defineMap("employees");
EmployeeOptimisticCallbackImpl cb = new EmployeeOptimisticCallbackImpl();
bm.setOptimisticCallback( cb );
```

Configuration par XML d'une implémentation d'OptimisticCallback

L'objet EmployeeOptimisticCallbackImpl de l'exemple qui précède doit implémenter l'interface OptimisticCallback. L'application peut également utiliser un fichier XML pour connecter son objet OptimisticCallback.

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
    <objectGrid name="grid1">
        <backingMap name="employees" pluginCollectionRef="employees" lockStrategy="OPTIMISTIC" />
    </objectGrid>
</objectGrids>

<backingMapPluginCollections>
    <backingMapPluginCollection id="employees">
        <bean id="OptimisticCallback" className="com.xyz.EmployeeOptimisticCallbackImpl" />
    </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Traitement des transactions

Introduction aux emplacements de plug-in

Un emplacement de plug-in est un espace de stockage de transactions qui est réservé aux plug-in partageant un contexte transactionnel. Ces emplacements permettent aux plug-in eXtreme Scale de communiquer entre eux, de partager un contexte transactionnel et de s'assurer que les ressources transactionnelles sont correctement utilisées de manière cohérente au sein de la transaction.

Dans un emplacement, un plug-in peut stocker un contexte transactionnel (connexion à la base de données, connexion Java Message Service (JMS), etc.). Le

contexte transactionnel qui est stocké peut être extrait par tout plug-in qui connaît le numéro de l'emplacement, lequel sert de clé pour l'extraction du contexte transactionnel.

Utiliser des emplacements de plug-in

Les emplacements de plug-in font partie de l'interface TxID. Voir la documentation de l'API pour plus d'informations sur cette interface.Les emplacements sont des entrées d'un tableau ArrayList. Les plug-in peuvent réserver une entrée dans ce tableau en appelant la méthode ObjectGrid.reserveSlot et en indiquant qu'ils souhaitent un emplacement sur tous les objets TxID. Une fois la réservation effectuée, les plug-in peuvent placer du contexte transactionnel dans les emplacements de chacun des objets TxID pour pouvoir extraire ultérieurement ce contexte. Les opérations put et get sont coordonnées par les numéros d'emplacements qui sont retournés par la méthode ObjectGrid.reserveSlot.

Un plug-in a normalement un cycle de vie. L'utilisation d'emplacements de plug-in doit donc se faire en tenant compte de ce cycle de vie. En principe, le plug-in doit réserver des emplacements pendant la phase d'initialisation et obtenir un numéro pour chacun des emplacements. Pendant l'exécution normale, le plug-in place au moment approprié du contexte transactionnel dans l'emplacement réservé sur l'objet TxID. Ce moment approprié se situe en général au début de la transaction. Au sein de la transaction, le plug-in (ou d'autres plug-in) peut alors récupérer dans le TxID, à partir du numéro de l'emplacement, le contexte transactionnel qui a été stocké de la sorte.

Normalement, le plug-in procède à un nettoyage en supprimant le contexte transactionnel et les emplacements. Le fragment suivant de code montre comment utiliser des emplacements dans un plug-in TransactionCallback :

```
public class DatabaseTransactionCallback implements TransactionCallback {
    int connectionSlot;
    int autoCommitConnectionSlot;
    int psCacheSlot;
    Properties ivProperties = new Properties();

    public void initialize(ObjectGrid objectGrid) throws TransactionCallbackException {
        // In initialization stage, reserve desired plug-in slots by calling the
        //reserveSlot method of ObjectGrid and
        // passing in the designated slot name, TxID.SLOT_NAME.
        // Note: you have to pass in this TxID.SLOT_NAME that is designated
        // for application.
        try {
            // cache the returned slot numbers
            connectionSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
            psCacheSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
            autoCommitConnectionSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
        } catch (Exception e) {
        }
    }

    public void begin(TxID tx) throws TransactionCallbackException {
        // put transactional contexts into the reserved slots at the
        // beginning of the transaction.
        try {
            Connection conn = null;
            conn = DriverManager.getConnection(ivDriverUrl, ivProperties);
            tx.putSlot(connectionSlot, conn);
            conn = DriverManager.getConnection(ivDriverUrl, ivProperties);
            conn.setAutoCommit(true);
            tx.putSlot(autoCommitConnectionSlot, conn);
            tx.putSlot(psCacheSlot, new HashMap());
        } catch (SQLException e) {
            SQLException ex = getLastSQLException(e);
            throw new TransactionCallbackException("unable to get connection", ex);
        }
    }

    public void commit(TxID id) throws TransactionCallbackException {
        // get the stored transactional contexts and use them
        // then, clean up all transactional resources.
        try {
            Connection conn = (Connection) id.getSlot(connectionSlot);
```

```
            conn.commit();
            cleanUpSlots(id);
        } catch (SQLException e) {
            SQLException ex = getLastSQLException(e);
            throw new TransactionCallbackException("commit failure", ex);
        }
    }

    void cleanUpSlots(TxID tx) throws TransactionCallbackException {
        closePreparedStatements((Map) tx.getSlot(psCacheSlot));
        closeConnection((Connection) tx.getSlot(connectionSlot));
        closeConnection((Connection) tx.getSlot(autoCommitConnectionSlot));
    }

    /**
     * @param map
     */
    private void closePreparedStatements(Map psCache) {
        try {
            Collection statements = psCache.values();
            Iterator iter = statements.iterator();
            while (iter.hasNext()) {
                PreparedStatement stmt = (PreparedStatement) iter.next();
                stmt.close();
            }
        } catch (Throwable e) {
        }

    }

    /**
     * Close connection and swallow any Throwable that occurs.
     *
     * @param connection
     */
    private void closeConnection(Connection connection) {
        try {
            connection.close();
        } catch (Throwable e1) {
        }
    }

    public void rollback(TxID id) throws TransactionCallbackException
        // get the stored transactional contexts and use them
        // then, clean up all transactional resources.
        try {
            Connection conn = (Connection) id.getSlot(connectionSlot);
            conn.rollback();
            cleanUpSlots(id);
        } catch (SQLException e) {
        }
    }

    public boolean isExternalTransactionActive(Session session) {
        return false;
    }

    // Getter methods for the slot numbers, other plug-in can obtain the slot numbers
    // from these getter methods.

    public int getConnectionSlot() {
        return connectionSlot;
    }
    public int getAutoCommitConnectionSlot() {
        return autoCommitConnectionSlot;
    }
    public int getPreparedStatementSlot() {
        return psCacheSlot;
    }
```

Le fragment suivant de code montre comment le Loader peut récupérer le contexte transactionnel qui a été stocké par l'exemple précédent de plug-in TransactionCallback :

```
public class DatabaseLoader implements Loader
{
    DatabaseTransactionCallback tcb;
    public void preloadMap(Session session, BackingMap backingMap) throws LoaderException
    {
        // The preload method is the initialization method of the Loader.
        // Obtain interested plug-in from Session or ObjectGrid instance.
        tcb =
   (DatabaseTransactionCallback)session.getObjectGrid().getTransactionCallback();
    }
    public List get(TxID txid, List keyList, boolean forUpdate) throws LoaderException
    {
        // get the stored transactional contexts that is put by tcb's begin method.
        Connection conn = (Connection)txid.getSlot(tcb.getConnectionSlot());
        // implement get here
```

```
        return null;
    }
    public void batchUpdate(TxID txid, LogSequence sequence) throws LoaderException,
   OptimisticCollisionException
    {
        // get the stored transactional contexts that is put by tcb's begin method.
        Connection conn = (Connection)txid.getSlot(tcb.getConnectionSlot());
        // implement batch update here ...
    }
}
```

Gestionnaire de transactions externes

En général, les transactions eXtreme Scale débutent par la méthode Session.begin et finissent par la méthode Session.commit. Toutefois, lorsqu'une ObjectGrid est imbriquée, un coordinateur de transactions externes peut débuter et terminer les transactions. Dans ce cas, il n'est pas nécessaire d'appeler les méthodes begin ou commit.

Coordination de transactions externes

Le plug-in TransactionCallback est étendu avec la méthode isExternalTransactionActive(Session session) qui associe le session eXtreme Scale à une transaction externe. L'en-tête de méthode est la suivante :

```
public synchronized boolean isExternalTransactionActive(Session session)
```

Par exemple, eXtreme Scale peut être configuré pour une intégration avec WebSphere Application Server et WebSphere Extended Deployment.

En outre, eXtreme Scale offre un plug-in intégré, désigné sous le nom de WebSphere «Plug-in de gestion des événements du cycle de vie des transactions», à la page 273, qui décrit comment créer le plug-in pour les environnements WebSphere Application Server bien qu'il soit possible d'adapter le plug-in pour d'autres infrastructures.

Cette intégration transparente repose essentiellement sur l'exploitation de l'API ExtendedJTATransaction dans WebSphere Application Server version 5.x et version 6.x. Toutefois, si vous utilisez WebSphere Application Server version 6.0.2, vous devez appliquer le correctif APAR PK07848 pour prendre en charge cette méthode. Utilisez l'exemple de code ci-dessous pour associer une session ObjectGrid avec un ID de transaction WebSphere Application Server :

```
/**
* Cette méthode est requise pour associer une session objectGrid avec un
* ID de transaction WebSphere Application Server.
*/
Map/**/ localIdToSession;
public synchronized boolean isExternalTransactionActive(Session session)
{
    // gardez à l'esprit que cet ID local implique l'enregistrement de cette session
    // pour une utilisation ultérieure.
    localIdToSession.put(new Integer(jta.getLocalId()), session);
    return true;
}
```

Extraction d'une transaction externe

Il est parfois conseillé d'extraire un objet de service de transaction externe pour le plug-in TransactionCallback à utiliser. Sur le serveur WebSphere Application Server,consultez l'objet ExtendedJTATransaction à partir de son espace de noms, tel qu'illustré dans l'exemple suivant :

```
public J2EETransactionCallback() {
    super();
    localIdToSession = new HashMap();
    String lookupName="java:comp/websphere/ExtendedJTATransaction";
    try
    {
        InitialContext ic =  new InitialContext();
        jta = (ExtendedJTATransaction)ic.lookup(lookupName);
        jta.registerSynchronizationCallback(this);
    }
    catch(NotSupportedException e)
    {
        throw new RuntimeException("Cannot register jta callback", e);
    }
    catch (NamingException e) {
        throw new RuntimeException("Cannot get transaction object");
    }
}
```

Pour d'autres produits, vous pouvez utiliser une approche similaire pour extraire l'objet de service de transaction.

Validation de contrôle par rappel externe

Le plug-in TransactionCallback doit recevoir un signal externe pour valider ou annuler la session eXtreme Scale. Pour ce faire, utilisez le rappel du service de transaction externe. Implémentez l'interface de rappel externe et enregistrez-la auprès du service de transaction externe. Par exemple, à l'aide de WebSphere Application Server, implémentez l'interface SynchronizationCallback, tel qu'illustré dans l'exemple suivant :

```
public class J2EETransactionCallback implements
 com.ibm.websphere.objectgrid.plugins.TransactionCallback, SynchronizationCallback {
   public J2EETransactionCallback() {
      super();
      String lookupName="java:comp/websphere/ExtendedJTATransaction";
      localIdToSession = new HashMap();
      try {
         InitialContext ic =  new InitialContext();
         jta = (ExtendedJTATransaction)ic.lookup(lookupName);
         jta.registerSynchronizationCallback(this);
      } catch(NotSupportedException e) {
         throw new RuntimeException("Cannot register jta callback", e);
      }
      catch (NamingException e) {
         throw new RuntimeException("Cannot get transaction object");
      }
   }

   public synchronized void afterCompletion(int localId, byte[] arg1,boolean didCommit) {
      Integer lid = new Integer(localId);
      // trouver la session pour l'ID local
      Session session = (Session)localIdToSession.get(lid);
      if(session != null) {
          try {
            // si WebSphere Application Server est validé lors du
            // renforcement de la transaction dans la mappe de sauvegarde.
            // Nous avons déjà effectué un vidage dans beforeCompletion
            if(didCommit) {
               session.commit();
            } else {
               // otherwise rollback
               session.rollback();
            }
         } catch(NoActiveTransactionException e) {
            // impossible en théorie
         } catch(TransactionException e) {
            // ne devrait pas échouer étant donné le vidage
         } finally {
            // efface toujours la session de la mappe.
            localIdToSession.remove(lid);
         }
      }
   }

   public synchronized void beforeCompletion(int localId, byte[] arg1) {
      Session session = (Session)localIdToSession.get(new Integer(localId));
      if(session != null) {
```

```
            try {
               session.flush();
            } catch(TransactionException e) {
               // WebSphere Application Server ne définit pas formellement
               // une méthode pour signaler
               // que la transaction a échoué, donc procédez comme suit
               throw new RuntimeException("Cache flush failed", e);
            }
         }
      }
   }
```

Utilisation des API eXtreme Scale avec le plug-in TransactionCallback

Le plug-in TransactionCallback désactive la validation automatique dans eXtreme Scale. Le modèle d'utilisation normal pour une API eXtreme Scale est le suivant :

```
Session ogSession = ...;
ObjectMap myMap = ogSession.getMap("MyMap");
ogSession.begin();
MyObject v = myMap.get("key");
v.setAttribute("newValue");
myMap.update("key", v);
ogSession.commit();
```

Lorsque ce plug-in TransactionCallback est utilisé, eXtreme Scale considère que l'application utilise l'API eXtreme Scale lorsqu'une transaction gérée par conteneur est identifiée. L'extrait de code précédent modifie le code ci-dessous dans cet environnement :

```
public void myMethod() {
   UserTransaction tx = ...;
   tx.begin();
   Session ogSession = ...;
   ObjectMap myMap = ogSession.getMap("MyMap");
   yObject v = myMap.get("key");
   v.setAttribute("newValue");
   myMap.update("key", v);
   tx.commit();
}
```

La méthode myMethod s'apparente à un scénario d'application Web. L'application utilise l'interface UserTransaction classique pour commencer, valider et annuler les transactions. L'API eXtreme Scale commence et valide automatiquement la transaction gérée par conteneur. Si la méthode est une méthode EJB (Enterprise JavaBeans) utilisant l'attribut TX_REQUIRES, supprimez la référence UserTransaction et les appels pour commencer et valider les transactions et vous constaterez que la méthode fonctionnera de la même manière. Dans ce cas, le conteneur se charge de démarrer et d'arrêter la transaction.

Plug-in WebSphereTransactionCallback

L'utilisation du plug-in WebSphereTransactionCallback permet aux applications d'entreprise qui s'exécutent en environnement WebSphere Application Server de gérer les transactions ObjectGrid.

Lorsqu'on utilise une session ObjectGrid au sein d'une méthode configurée pour utiliser des transactions gérées par conteneur, le conteneur d'entreprise automatiquement commence, valide ou annule la transaction ObjectGrid. Lorsqu'on utilise des objets UserTransaction de l'API Java Transaction (JTA), la transaction ObjectGrid est gérée automatiquement par l'objet UserTransaction.

Pour une discussion détaillée de l'implémentation de ce plug-in, voir «Gestionnaire de transactions externes», à la page 280.

Remarque : L'ObjectGrid ne prend pas en charge les transactions à 2 phases ou transactions XA. Ce plug-in n'enregistre pas la transaction ObjectGrid auprès du gestionnaire de transactions. Il en résulte que, si l'ObjectGrid ne parvient pas à valider, les autres ressources gérées par la transaction XA ne sont pas annulées.

Activation du plug-in WebSphereTransactionCallback

L'activation de WebSphereTransactionCallback dans la configuration d'ObjectGrid peut se faire par programmation ou par configuration XML.

Configuration par XML du plug-in WebSphereTransactionCallback

La configuration XML suivante crée l'objet WebSphereTransactionCallback et l'ajoute à un ObjectGrid. Le texte suivant doit se trouver dans le fichier `myGrid.xml` :

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
    xmlns="http://ibm.com/ws/objectgrid/config">
    <objectGrids>
        <objectGrid name="myGrid">
            <bean id="TransactionCallback" className=
 "com.ibm.websphere.objectgrid.plugins.builtins.WebSphereTransactionCallback" />

        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

Configuration par programmation du plug-in WebSphereTransactionCallback

Le fragment de code suivant crée l'objet WebSphereTransactionCallback et l'ajoute à un ObjectGrid :

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
WebSphereTransactionCallback wsTxCallback= new WebSphereTransactionCallback ();
myGrid.setTransactionCallback(wsTxCallback);
```

Chapitre 6. Ecriture du code de tâches administratives

Outre les interfaces de programmation d'applications du système (API), WebSphere eXtreme Scale contient également des API d'administration permettant aux applications de surveiller et d'administrer les serveurs et les clients.

API de serveurs intégrés

WebSphere eXtreme Scale comprend des interfaces de programmes d'application (API) et des interfaces de programmation de système permettant d'intégrer des serveurs et des clients eXtreme Scale dans vos applications Java existantes. Nous allons décrire ces API de serveurs intégrés.

Instanciation du serveur eXtreme Scale

Plusieurs propriétés permettent de configurer l'instance du serveur eXtreme Scale, qu'il est possible d'extraire de la méthode ServerFactory.getServerProperties. L'objet ServerProperties étant un singleton, chaque appel à la méthode getServerProperties extrait la même instance.

Le code suivant permet de créer un serveur :

```
Server server = ServerFactory.getInstance();
```

Toutes les propriétés définies avant la première invocation de getInstance sont utilisées pour initialiser le serveur.

Définir les propriétés du serveur

Vous pouvez définir les propriétés du serveur jusqu'au premier appel à la méthode ServerFactory.getInstance. Ce premier appel instancie le serveur eXtreme Scale et lit toutes les propriétés configurées. Les propriétés définies après la création n'ont aucun effet. L'exemple qui suit montre comment définir les propriétés avant d'instancier une instance Server.

```
// L'on obtient les propriétés du serveur associées à ce processus.
ServerProperties serverProperties = ServerFactory.getServerProperties();

// L'on définit le nom du serveur pour ce processus.
serverProperties.setServerName("EmbeddedServerA");

// L'on définit le nom de la zone dans laquelle est contenu ce processus.
serverProperties.setZoneName("EmbeddedZone1");

// L'on définit les informations de point de contact requises
// pour l'amorçage du service de catalogue.
serverProperties.setCatalogServiceBootstrap("localhost:2809");

// L'on définit le nom de l'hôte d'écoute de l'ORB à utiliser pour la liaison.
serverProperties.setListenerHost("host.local.domain");

// L'on définit le port d'écoute de l'ORB à utiliser pour la liaison.
serverProperties.setListenerPort(9010);

// L'on désactive tous les beans gérés pour ce processus.
serverProperties.setMBeansEnabled(false);

Server server = ServerFactory.getInstance();
```

Incorporer le service de catalogue

Tout paramètre JVM marqué par la méthode CatalogServerProperties.setCatalogServer peut héberger le service de catalogue d'eXtreme Scale. Cette méthode demande à l'environnement d'exécution du serveur eXtreme Scale d'instancier le service de catalogue lors du démarrage du serveur. Le code qui suit montre comment instancier le serveur de catalogue eXtreme Scale :

```
CatalogServerProperties catalogServerProperties =
 ServerFactory.getCatalogProperties();
catalogServerProperties.setCatalogServer(true);

Server server = ServerFactory.getInstance();
```

Incorporer le conteneur eXtreme Scale

La méthode Server.createContainer permet à n'importe quelle machine virtuelle Java d'héberger plusieurs conteneurs eXtreme Scale. Le code qui suit montre comment instancier un conteneur eXtreme Scale :

```
Server server = ServerFactory.getInstance();
DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy(
    new File("META-INF/embeddedDeploymentPolicy.xml").toURI().toURL(),
    new File("META-INF/embeddedObjectGrid.xml").toURI().toURL());
Container container = server.createContainer(policy);
```

Processus serveur autonome

Vous pouvez démarrer ensemble tous les services, ce qui est utilise aussi bien en phase de développement qu'en production. En démarrant les services ensemble, le même processus se charge de toutes les tâches suivantes : démarrage du service de catalogue, démarrage d'un ensemble de conteneurs, exécution de la logique de connexion client. Démarrer les services de cette manière résout les problèmes de programmation antérieurs au déploiement dans un environnement réparti. Le code qui suit montre comment instancier un serveur eXtreme Scale autonome :

```
CatalogServerProperties catalogServerProperties =
 ServerFactory.getCatalogProperties();
catalogServerProperties.setCatalogServer(true);

Server server = ServerFactory.getInstance();
DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy(
    new File("META-INF/embeddedDeploymentPolicy.xml").toURI().toURL(),
    new File("META-INF/embeddedObjectGrid.xml").toURI().toURL());
Container container = server.createContainer(policy);
```

Intégrer eXtreme Scale dans WebSphere Application Server

La configuration d'eXtreme Scale s'effectue automatiquement lorsqu'on installe WebSphere Extended Deployment DataGrid dans un environnement WebSphere Application Server. Vous n'êtes pas obligé de définir des propriétés avant d'accéder au serveur pour créer un conteneur. Le code qui suit montre comment instancier un serveur eXtreme Scale dans WebSphere Application Server :

```
Server server = ServerFactory.getInstance();
DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy(
    new File("META-INF/embeddedDeploymentPolicy.xml").toURI().toURL(),
    new File("META-INF/embeddedObjectGrid.xml").toURI().toURL);
Container container = server.createContainer(policy);
```

«Utilisation de l'API des serveurs imbriqués», à la page 287 contient un exemple expliquant pas à pas comment démarrer par programmation un service de catalogue et un conteneur intégrés.

Utilisation de l'API des serveurs imbriqués

Avec WebSphere eXtreme Scale, vous pouvez utiliser une interface de programmation d'application pour gérer le cycle de vie des conteneurs et serveurs imbriqués. Vous pouvez configurer le serveur à l'aide d'un programme avec les options que vous pouvez également configurer avec la ligne de commande ou les propriétés de serveur incluses dans un fichier. Vous pouvez configurer le serveur imbriqué pour en faire un serveur conteneur et/ou un service de catalogue.

Avant de commencer

Vous devez disposer d'une méthode permettant d'exécuter du code depuis une machine virtuelle Java existante. les classes eXtreme Scale doivent être disponibles dans l'arborescence du chargeur de classe.

Pourquoi et quand exécuter cette tâche

Vous pouvez effectuer de nombreuses tâches d'administration à l'aide de l'API d'administration. L'API est couramment utilisée comme serveur interne pour stocker l'état d'une application Web. Le serveur Web peut démarrer un serveur WebSphere eXtreme Scale imbriqué et signaler le serveur conteneur au service de catalogue. Le serveur est ensuite ajouté comme membre d'une grille répartie plus importante. Cette utilisation peut offrir des possibilité d'évolution et une haute disponibilité à un fichier de données qui reste sinon volatile.

Vous pouvez contrôler à l'aide d'un programme le cycle de vie complet d'un serveur eXtreme Scale imbriqué. Les exemples sont aussi génériques que possible et n'illustrent que des exemples de code spécifiques aux étapes présentées.

Procédure

1. Procurez-vous l'objet ServerProperties de la classe ServerFactory et configurez les options nécessaires.

   Chaque serveur eXtreme Scale possède un ensemble de propriétés configurables. Lorsqu'un serveur est démarré à partir de la ligne de commande, ces propriétés reçoivent les valeurs par défaut, mais vous pouvez remplacer plusieurs propriétés en fournissant un fichier ou une source externe. Dans la portée imbriquée, vous pouvez directement définir les propriétés avec un objet ServerProperties. Vous devez définir ces propriétés avant d'obtenir une instance de serveur de la classe ServerFactory. L'exemple de fragment de code ci-après obtient un objet ServerProperties, définit le champ CatalogServiceBootStrap et initialise plusieurs paramètres de serveur facultatifs. Pour une liste des paramètres configurables, reportez-vous à la documentation de l'API.

   ```
   ServerProperties props = ServerFactory.getServerProperties();
   props.setCatalogServiceBootstrap("host:port"); // requis pour se connecter à un service
   de catalogue spécifique
   props.setServerName("ServerOne"); // nommez le serveur
   props.setTraceSpecification("com.ibm.ws.objectgrid=all=enabled"); // Définit la
   spécification de trace
   ```

2. Si vous souhaitez que le serveur soit un service de catalogue, procurez-vous l'objet CatalogServerProperties.

   Chaque serveur imbriqué peut être un service de catalogue, un serveur conteneur ou un serveur conteneur et un service de catalogue. L'exemple ci-après obtient l'objet CatalogServerProperties, active l'option de service de catalogue et configure divers paramètres de service de catalogue.

```
CatalogServerProperties catalogProps = ServerFactory.getCatalogProperties();
catalogProps.setCatalogServer(true); // false par défaut ; doit être défini comme
service de catalogue
catalogProps.setQuorum(true); // active/désactive le quorum
```

3. Procurez-vous une instance Server à partir de la classe ServerFactory. L'instance Server est un singleton de portée processus chargé de gérer l'appartenance dans la grille. Une fois que cette instance a été instanciée, ce processus est connecté et devient hautement disponible pour les autres serveurs de la grille. L'exemple suivant montre comment créer l'instance Server :

   ```
   Server server = ServerFactory.getInstance();
   ```

   Si nous considérons l'exemple précédent, la classe ServerFactory fournit une méthode statique qui renvoie une instance Server. La classe ServerFactory est prévue pour être la seule interface permettant d'obtenir une instance Server. Par conséquent la classe garantit que l'instance est un singleton ou une instance pour chaque machine virtuelle Java ou chargeur de classe isolé. La méthode getInstance initialise l'instance Server. Vous devez configurer toutes les propriétés du serveur avant d'initialiser l'instance. La classe Server est chargée de créer des instances Container. Vous pouvez utiliser à la fois la classe ServerFactory et la classe Server pour gérer le cycle de vie de l'instance Server imbriquée.

4. Démarrez une instance Container à l'aide de l'instance Server.

   Pour que des fragments puissent être positionnées sur un serveur imbriqué, vous devez créer un conteneur sur le serveur. L'interface Server contient une méthode createContainer et accepte un argument DeploymentPolicy. L'exemple ci-après utilise l'instance de serveur que vous avez obtenue pour créer un conteneur à l'aide du fichier de règles de déploiement. Notez que les conteneurs requièrent un chargeur de classe pour lequel les fichiers binaires de l'application sont disponibles, à des fins de sérialisation. Vous pouvez rendre ces fichiers binaires disponibles en appelant la méthode createContainer avec comme chargeur de classe du contexte de l'unité d'exécution, celui que vous souhaitez utiliser.

   ```
   DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy(new
      URL("file://urltodeployment.xml"),
    new URL("file://urltoobjectgrid.xml"));
   Container container = server.createContainer(policy);
   ```

5. Supprimez et nettoyez un conteneur.

   Vous pouvez supprimer et nettoyer un serveur conteneur en exécutant la méthode teardown sur l'instance Container obtenue. L'exécution de la méthode teardown sur un conteneur nettoie ce dernier de manière appropriée et supprime le conteneur du serveur imbriqué.

   La procédure de nettoyage du conteneur inclut le déplacement et le démontage de tous les fragments positionnés dans ce conteneur. Chaque serveur peut contenir plusieurs conteneurs et fragments. Le nettoyage d'un conteneur n'affecte pas le cycle de vie de l'instance Server parent. L'exemple ci-après montre comment exécuter la méthode teardown sur un serveur. La méthode teardown est rendue accessible via l'interface ContainerMBean. En utilisant l'interface ContainerMBean, si vous n'avez plus accès à ce conteneur à l'aide d'un programme, vous pouvez toujours le supprimer et le nettoyer avec son bean géré. Une méthode terminate existe également dans l'interface Container ; ne l'utilisez pas, sauf si cela est indispensable. Cette méthode est plus puissante et ne coordonne pas un déplacement et un nettoyage des fragments appropriés.

   ```
   container.teardown();
   ```

6. Arrêtez le serveur imbriqué.

   Lorsque vous arrêtez un serveur imbriqué, vous arrêtez également les conteneurs et les fragments en cours d'exécution sur ce serveur. Lorsque vous

arrêtez un serveur imbriqué, vous devez nettoyer toutes les connexions ouvertes et déplacer ou démonter tous les fragments. L'exemple ci-après illustre comment arrêter un serveur et utiliser la méthode waitFor sur l'instance Server pour s'assurer que cette dernière s'arrête complètement. Comme pour l'exemple de conteneur, la méthode stopServer est rendue accessible via l'interface ServerMBean. A l'aide de cette interface, vous pouvez arrêter un serveur avec le bean géré (MBean) correspondant.

```
ServerFactory.stopServer();  // Utilise la fabrique pour arrêter le singleton du serveur
 // ou
server.stopServer(); // Utilise directement l'instance Server
server.waitFor(); // Est renvoyé une fois que le serveur a correctement terminé ses
procédures d'arrêt
```

Exemple de code complet :

```
 import java.net.MalformedURLException;
import java.net.URL;

import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.deployment.DeploymentPolicy;
import com.ibm.websphere.objectgrid.deployment.DeploymentPolicyFactory;
import com.ibm.websphere.objectgrid.server.Container;
import com.ibm.websphere.objectgrid.server.Server;
import com.ibm.websphere.objectgrid.server.ServerFactory;
import com.ibm.websphere.objectgrid.server.ServerProperties;

public class ServerFactoryTest {

    public static void main(String[] args) {

        try {

            ServerProperties props = ServerFactory.getServerProperties();
            props.setCatalogServiceBootstrap("catalogservice-hostname:catalogservice-port");
            props.setServerName("ServerOne"); // name server
            props.setTraceSpecification("com.ibm.ws.objectgrid=all=enabled"); // TraceSpec

            /*
             * Dans la plupart des cas, le serveur ne sert que de serveur conteneur
             * et se connecte à un service de catalogue externe. Cette utilisation
             * favorise davantage la haute disponibilité. L'extrait de code commenté
             * ci-après permet à ce serveur de devenir un service de catalogue.
             *
             *
             * CatalogServerProperties catalogProps =
             * ServerFactory.getCatalogProperties();
             * catalogProps.setCatalogServer(true); // activez le service de catalogue
             * catalogProps.setQuorum(true); // activez le quorum
             */

            Server server = ServerFactory.getInstance();

            DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy
     (new URL("url to deployment xml"),  new URL("url to objectgrid xml file"));
            Container container = server.createContainer(policy);

            /*
             * Le fragment est maintenant positionné sur ce conteneur si les exigences
             * de déploiement sont satisfaites.
             * Cela englobe la création du serveur et du conteneur imbriqués.
             *
             * Les lignes ci-après illustrent simplement l'appel des méthodes de nettoyage
             */

            container.teardown();
            server.stopServer();
            int success = server.waitFor();

        } catch (ObjectGridException e) {
            // Container failed to initialize
        } catch (MalformedURLException e2) {
            // invalid url to xml file(s)
        }

    }

}
```

Surveillance à l'aide de l'API Statistics

L'API Statistics est l'interface directe avec l'arborescence interne des statistiques. Les statistiques sont désactivées par défaut, mais peuvent être activées en définissant une interface StatsSpec. Une interface StatsSpec définit la manière dont WebSphere eXtreme Scale doit surveiller les statistiques.

Pourquoi et quand exécuter cette tâche

Vous pouvez utiliser l'API locale StatsAccessor pour interroger les données et accéder aux statistiques d'une instance ObjectGrid qui se trouve sur la même machine virtuelle Java (JVM) que le code en cours d'exécution. Pour plus d'informations sur les interfaces spécifiques, voir la documentation de l'API. Utilisez les étapes ci-après pour activer la surveillance de l'arborescence des statistiques interne.

Procédure

1. Extrayez l'objet StatsAccessor. L'interface StatsAccessor suit le modèle des singletons. Par conséquent, en dehors des problèmes liés au chargeur de classe, il doit exister une instance StatsAccessor pour chaque JVM. Cette classe sert d'interface principale pour toutes les opérations sur les statistiques locales. Le code ci-après illustre l'extraction de la classe de l'accesseur. Appelez cette opération avant tout autre appel ObjectGrid.

```
public class LocalClient
{

   public static void main(String[] args) {

      // extrayez un descripteur de StatsAccessor
      StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

   }

}
```

2. Définissez l'interface StatsSpec de la grille. Définissez cette JVM de sorte qu'elle ne collecte toutes les statistiques qu'au niveau d'ObjectGrid. Vous devez vérifier qu'une application active toutes les statistiques qui peuvent être requises avant de commencer des transactions. L'exemple ci-après définit l'interface StatsSpec à l'aide d'un champ de constante statique et d'une chaîne de spécification. L'utilisation d'un champ de constante statique est plus simple car le champ a déjà défini la spécification. Toutefois, en utilisant une chaîne de spécification, vous pouvez autoriser toutes les combinaisons de statistiques requises.

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

      // extrayez un descripteur de StatsAccessor
      StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

      // Définissez la spécification via le champ statique
      StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
      accessor.setStatsSpec(spec);

      // Définissez la spécification via la chaîne de spécification
      StatsSpec spec = new StatsSpec("og.all=enabled");
      accessor.setStatsSpec(spec);

}
```

3. Envoyez des transactions à la grille pour force la collecte des données en vue de la surveillance. Pour collecter des données utiles pour les statistiques, vous devez envoyer des transactions à la grille. L'extrait de code suivant insère un

enregistrement dans MapA, qui se trouve dans ObjectGridA. Les statistiques se trouvant au niveau d'ObjectGrid, toute mappe dans l'ObjectGrid renvoie les mêmes résultats.

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

      // extrayez un descripteur de StatsAccessor
      StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

      // Définissez la spécification via le champ
      StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
      accessor.setStatsSpec(spec);

      ObjectGridManager manager =
    ObjectGridmanagerFactory.getObjectGridManager();
      ObjectGrid grid = manager.getObjectGrid("ObjectGridA");
      Session session = grid.getSession();
      Map map = session.getMap("MapA");

      // Effectuez une insertion
      session.begin();
      map.insert("SomeKey", "SomeValue");
      session.commit();
}
```

4. Interrogez un objet StatsFact à l'aide de l'API StatsAccessor. Tous les chemins d'accès aux statistiques sont associés à une interface StatsFact. L'interface StatsFact est une marque de réservation générique permettant d'organiser et d'inclure un objet StatsModule. Pour que vous puissiez accéder au véritable module de statistiques, l'objet StatsFact doit être extrait.

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

      // extrayez un descripteur de StatsAccessor
      StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

      // Définissez la spécification via le champ statique
      StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
      accessor.setStatsSpec(spec);

      ObjectGridManager manager =
    ObjectGridManagerFactory.getObjectGridManager();
      ObjectGrid grid = manager.getObjectGrid("ObjectGridA");
      Session session = grid.getSession();
      Map map = session.getMap("MapA");

      // Effectuez une insertion
      session.begin();
      map.insert("SomeKey", "SomeValue");
      session.commit();

      // Extrayez StatsFact

      StatsFact fact = accessor.getStatsFact(new String[] {"EmployeeGrid"},
    StatsModule.MODULE_TYPE_OBJECT_GRID);

}
```

5. Interagissez avec l'objet StatsModule. L'objet StatsModule est contenu dans l'interface StatsFact. Vous pouvez obtenir une référence au module à l'aide de l'interface StatsFact. L'interface StatsFact étant une interface générique, vous devez transtyper le module renvoyé dans le type StatsModule attendu. Cette tâche collectant des statistiques eXtreme Scale, l'objet StatsModule renvoyé est transtypé dans un type OGStatsModule. Une fois que le module est transtypé, vous avez accès à toutes les statistiques disponibles.

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

      // extrayez un descripteur de StatsAccessor
      StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

      // Définissez la spécification via le champ statique
      StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
      accessor.setStatsSpec(spec);

      ObjectGridManager manager =
    ObjectGridmanagerFactory.getObjectGridManager();
      ObjectGrid grid = manager.getObjectGrid("ObjectGridA");
      Session session = grid.getSession();
      Map map = session.getMap("MapA");

      // Effectuez une insertion
      session.begin();
      map.insert("SomeKey", "SomeValue");
      session.commit();

      // Extrayez StatsFact
      StatsFact fact = accessor.getStatsFact(new String[] {"EmployeeGrid"},
    StatsModule.MODULE_TYPE_OBJECT_GRID);

      // Extrayez le module et l'heure
      OGStatsModule module = (OGStatsModule)fact.getStatsModule();
      ActiveTimeStatistic timeStat =
    module.getTransactionTime("Default", true);
      double time = timeStat.getMeanTime();

}
```

Surveillance à l'aide de la fonction PMI de WebSphere Application Server

WebSphere eXtreme Scale prend en charge PMI (Performance Monitoring Infrastructure) lorsqu'il est exécuté dans un serveur d'applications WebSphere Application Server ou WebSphere Extended Deployment. PMI collecte des données de performances relatives aux applications exécutables et offre des interfaces permettant aux applications externes de surveiller les données de performances. Vous pouvez utiliser la console d'administration ou l'outil wsadmin pour accéder aux données de surveillance.

Avant de commencer

Vous pouvez utiliser PMI pour surveiller votre environnement lorsque vous utilisez WebSphere eXtreme Scale avec WebSphere Application Server.

Pourquoi et quand exécuter cette tâche

WebSphere eXtreme Scale utilise la fonction PMI personnalisée de WebSphere Application Server pour ajouter sa propre instrumentation de PMI. Avec cette approche, vous pouvez activer et désactiver la fonction PMI de WebSphere eXtreme Scale à l'aide de la console d'administration ou des interfaces JMX (Java Management Extensions) de l'outil wsadmin. En outre, vous pouvez accéder aux statistiques de WebSphere eXtreme Scale à l'aide des interfaces PMI et JMX standard utilisées par les outils de surveillance et notamment Tivoli Performance Viewer.

Procédure

1. Activez la fonction PMI de eXtreme Scale. Vous devez activer PMI pour afficher les statistiques PMI. Pour plus d'informations, voir «Activation de PMI».
2. Extrayez les statistiques PMI de eXtreme Scale. Affichez les performances de vos applications eXtreme Scale à l'aide de Tivoli Performance Viewer. Pour plus d'informations, voir «Récupération des statistiques PMI», à la page 295.

Que faire ensuite

Pour plus d'informations sur l'outil wsadmin, voir «Accès aux beans gérés à l'aide de l'outil wsadmin», à la page 303.

Activation de PMI

Vous pouvez utiliser l'infrastructure PMI (Performance Monitoring Infrastructure) de WebSphere Application Server pour activer ou désactiver les statistiques à tout niveau. Par exemple, vous pouvez choisir d'activer les statistiques du nombre d'occurrences d'une mappe donnée, mais non le nombre de statistiques en entrée ou les statistiques de durée de mise à jour par lots du chargeur. Vous pouvez activer PMI dans la console d'administration ou à l'aide de scripts.

Avant de commencer

Votre serveur d'applications doit être démarré et une application compatible eXtreme Scale doit y être installée. Pour activer PMI à l'aide de scripts, vous devez pouvoir vous connecter et utiliser l'outil wsadmin. Pour plus d'informations sur l'outil wsadmin, reportez-vous à la rubrique Outil wsadmin, dans le Centre de documentation de WebSphere Application Server.

Pourquoi et quand exécuter cette tâche

Utilisez l'infrastructure PMI de WebSphere Application Server pour fournir un mécanisme granulaire à l'aide duquel vous pouvez activer ou désactiver les statistiques à tout niveau. Par exemple, vous pouvez choisir d'activer les statistiques du nombre d'occurrences d'une mappe donnée, mais non le nombre d'entrées ou les statistiques de durée de mise à jour par lots du chargeur. Cette section montre comment utiliser la console d'administration et les scripts wsadmin pour activer l'infrastructure PMI d'ObjectGrid.

Procédure

- **Activez PMI dans la console d'administration.**
  1. Dans la console d'administration, cliquez sur **Contrôle et réglage** → **Performance Monitoring Infrastructure** → *nom_serveur*.
  2. Vérifiez que la case Activer l'infrastructure PMI (Performance Monitoring Infrastructure) est cochée. Ce paramètre est activé par défaut. S'il ne l'est pas, cochez la case et redémarrez le serveur.
  3. Cliquez sur **Personnalisé**. Dans l'arborescence de configuration, sélectionnez l'ObjectGrid et le module Mappes d'ObjectGrid. Activez les statistiques de chaque module.

  La catégorie des types de transaction des statistiques ObjectGrid est créée lors de la phase d'exécution. Vous ne pouvez voir que les sous-catégories des statistiques ObjectGrid et des statistiques de mappe dans la page **Exécution**.
- **Activez PMI à l'aide de scripts.**

1. Ouvrez une invite de ligne de commande. Accédez au répertoire racine_install/bin. Entrez wsadmin pour démarrer l'outil de ligne de commande wsadmin.
2. Modifiez la configuration de l'environnement d'exécution de l'infrastructure PMI d'eXtreme Scale. Vérifiez que PMI est activé pour le serveur à l'aide des commandes suivantes :

   ```
   wsadmin>set s1 [$AdminConfig getid /Cell:CELL_NAME/Node:NODE_NAME/
       Server:APPLICATION_SERVER_NAME/]
   wsadmin>set pmi [$AdminConfig list PMIService $s1]
   wsadmin>$AdminConfig show $pmi.
   ```

   Si PMI n'est pas activé, exécutez les commandes suivantes pour activer PMI :

   ```
   wsadmin>$AdminConfig modify $pmi {{enable true}}
   wsadmin>$AdminConfig save
   ```

   Si vous avez besoin d'activer PMI, redémarrez le serveur.
3. Définissez des variables pour modifier l'ensemble de statistiques en ensemble personnalisé à l'aide des commandes suivantes :

   ```
   wsadmin>set perfName [$AdminControl completeObjectName type=Perf,
   process=APPLICATION_SERVER_NAME,*]
   wsadmin>set perfOName [$AdminControl makeObjectName $perfName]
   wsadmin>set params [java::new {java.lang.Object[]} 1]
   wsadmin>$params set 0 [java::new java.lang.String custom]
   wsadmin>set sigs [java::new {java.lang.String[]} 1]
   wsadmin>$sigs set 0 java.lang.String
   ```
4. Spécifiez un ensemble de statistiques personnalisé à l'aide de la commande suivante :

   ```
   wsadmin>$AdminControl invoke_jmx $perfOName setStatisticSet $params $sigs
   ```
5. Définissez des variables pour activer les statistiques de l'infrastructure PMI d'objectGridModule à l'aide des commandes suivantes :

   ```
   wsadmin>set params [java::new {java.lang.Object[]} 2]
   wsadmin>$params set 0 [java::new java.lang.String objectGridModule=1]
   wsadmin>$params set 1 [java::new java.lang.Boolean false]
   wsadmin>set sigs [java::new {java.lang.String[]} 2]
   wsadmin>$sigs set 0 java.lang.String
   wsadmin>$sigs set 1 java.lang.Boolean
   ```
6. Définissez la chaîne des statistiques à l'aide de la commande suivante :

   ```
   wsadmin>set params2 [java::new {java.lang.Object[]} 2]
   wsadmin>$params2 set 0 [java::new java.lang.String mapModule=*]
   wsadmin>$params2 set 1 [java::new java.lang.Boolean false]
   wsadmin>set sigs2 [java::new {java.lang.String[]} 2]
   wsadmin>$sigs2 set 0 java.lang.String
   wsadmin>$sigs2 set 1 java.lang.Boolean
   ```
7. Définissez la chaîne des statistiques à l'aide de la commande suivante :

   ```
   wsadmin>$AdminControl invoke_jmx $perfOName setCustomSetString $params2 $sigs2
   ```

Ces étapes activent l'infrastructure PMI de l'environnement d'exécution d'eXtreme Scale, mais ne modifient pas la configuration de l'infrastructure PMI. Si vous redémarrez l'application, les paramètres PMI sont perdus, exceptée l'activation principale de PMI.

Exemple

Vous pouvez effectuer les étapes suivantes pour activer les statistiques PMI de l'exemple d'application :

1. Lancez l'application à l'aide de l'adresse Web `http://hôte:port/ObjectGridSample`, hôte et port correspondant au nom d'hôte et au numéro de port HTTP du serveur où l'exemple est installé.
2. Dans l'exemple d'application, cliquez sur ObjectGridCreationServlet, puis sur les boutons d'action 1, 2, 3, 4 et 5 pour générer des actions sur l'ObjectGrid et les mappes. Ne fermez pas tout de suite cette page de servlet.
3. Dans la console d'administration, cliquez sur **Contrôle et réglage** → **Performance Monitoring Infrastructure** → *nom_serveur*. Cliquez sur l'onglet **Exécution**.
4. Cliquez sur le bouton d'option **Personnalisé**.
5. Développez le module Mappes d'ObjectGrid dans l'arborescence d'exécution, puis cliquez sur le lien clusterObjectGrid. Le groupe Mappes d'ObjectGrid contient une instance ObjectGrid appelée clusterObjectGrid et le groupe clusterObjectGrid contient quatre mappes : counters, employees, offices, et sites. Dans l'instance ObjectGrids se trouve une instance clusterObjectGrid et sous cette instance, le type de transaction DEFAULT.
6. Vous pouvez activer les statistiques de votre choix. Par exemple, vous pouvez activer le nombre d'entrées de mappe pour la mappe des employés et le temps de réponse des transactions pour le type de transaction DEFAULT.

Que faire ensuite

Une fois que PMI est activé, vous pouvez afficher les statistiques PMI à l'aide de la console d'administration ou de scripts.

Récupération des statistiques PMI

En récupérant les statistiques PMI, vous pouvez voir les performances de vos applications eXtreme Scale.

Avant de commencer

- Activez la fonction de suivi des statistiques PMI pour votre environnement. Pour plus d'informations, voir «Activation de PMI», à la page 293.
- Les chemins dans cette tâche partent du principe que vous récupérez les statistiques pour l'exemple d'application, mais vous pouvez utiliser ces statistiques pour toute autre application avec des étapes similaires.
- Si vous utilisez la console d'administration, vous devez être capable de vous y connecter. Si vous utilisez un script, vous devez être capable de vous connecter à wsadmin.

Pourquoi et quand exécuter cette tâche

Vous pouvez récupérer les statistiques PMI pour les afficher dans Tivoli Performance Viewer en suivant les étapes dans la console d'administration ou par script.

- Etapes de la console d'administration
- Etapes du script

Pour plus d'informations concernant les statistiques qui peuvent être récupérées, voir «Modules PMI», à la page 296.

Procédure

- Récupérez les statistiques PMI dans la console d'administration.

1. Dans la console d'administration, cliquez sur **Contrôle et réglage → Performance viewer → Activité actuelle**
2. Sélectionnez le serveur que vous voulez contrôler à l'aide de Tivoli Performance Viewer, puis activez le contrôle.
3. Cliquez sur le serveur pour afficher la page Performance viewer.
4. Développez l'arborescence de configuration. Cliquez sur **ObjectGrid Maps → clusterObjectGrid**, sélectionnez **employés**. Développez **ObjectGrids → clusterObjectGrid** et sélectionnez **DEFAULT**.
5. Dans le modèle d'application ObjectGrid, accédez au servlet ObjectGridCreationServlet, cliquez sur le bouton 1 et remplissez les mappes. Vous pouvez afficher les statistiques dans l'afficheur.

- Récupérez les statistiques PMI avec un script.
  1. Dans une ligne de commande, accédez au répertoire `install_root/bin`. Entrez `wsadmin` pour lancer l'outil wsadmin.
  2. Définissez les variables pour l'environnement à l'aide des commandes suivantes :

     ```
     wsadmin>set perfName [$AdminControl completeObjectName type=Perf,*]
     wsadmin>set perfOName [$AdminControl makeObjectName $perfName]
     wsadmin>set mySrvName [$AdminControl completeObjectName type=Server,
      name=APPLICATION_SERVER_NAME,*]
     ```

  3. Définissez les variables pour obtenir les statistiques de mapModule à l'aide des commandes suivantes :

     ```
     wsadmin>set params [java::new {java.lang.Object[]} 3]
     wsadmin>$params set 0 [$AdminControl makeObjectName $mySrvName]
     wsadmin>$params set 1 [java::new java.lang.String mapModule]
     wsadmin>$params set 2 [java::new java.lang.Boolean true]
     wsadmin>set sigs [java::new {java.lang.String[]} 3]
     wsadmin>$sigs set 0 javax.management.ObjectName
     wsadmin>$sigs set 1 java.lang.String
     wsadmin>$sigs set 2 java.lang.Boolean
     ```

  4. Obtenez les statistiques de mapModule à l'aide de la commande suivante :

     ```
     wsadmin>$AdminControl invoke_jmx $perfOName getStatsString $params $sigs
     ```

  5. Définissez les variables pour obtenir les statistiques d'objectGridModule à l'aide des commandes suivantes :

     ```
     wsadmin>set params2 [java::new {java.lang.Object[]} 3]
     wsadmin>$params2 set 0 [$AdminControl makeObjectName $mySrvName]
     wsadmin>$params2 set 1 [java::new java.lang.String objectGridModule]
     wsadmin>$params2 set 2 [java::new java.lang.Boolean true]
     wsadmin>set sigs2 [java::new {java.lang.String[]} 3]
     wsadmin>$sigs2 set 0 javax.management.ObjectName
     wsadmin>$sigs2 set 1 java.lang.String
     wsadmin>$sigs2 set 2 java.lang.Boolean
     ```

  6. Obtenez les statistiques d'objectGridModule à l'aide de la commande suivante :

     ```
     wsadmin>$AdminControl invoke_jmx $perfOName getStatsString $params2 $sigs2
     ```

Résultats

Vous pouvez afficher les statistiques dans Tivoli Performance Viewer.

Modules PMI

Vous pouvez surveiller les performances de vos applications avec les modules PMI (Performance Monitoring Infrastructure).

objectGridModule

Le module objectGridModule contient une statistique de durée : le temps de réponse des transactions. Une transaction est définie comme la durée entre l'appel de méthode Session.begin et l'appel de méthode Session.commit. Cette durée est suivie comme temps de réponse des transactions. L'élément racine de la structure objectGridModule, "root", sert de point d'entrée aux statistiques de WebSphere eXtreme Scale. Cet élément racine contient des ObjectGrids comme éléments enfant et ces derniers possèdent des types de transaction comme éléments enfant. Les statistiques de temps de réponse sont associées à chaque type de transaction.

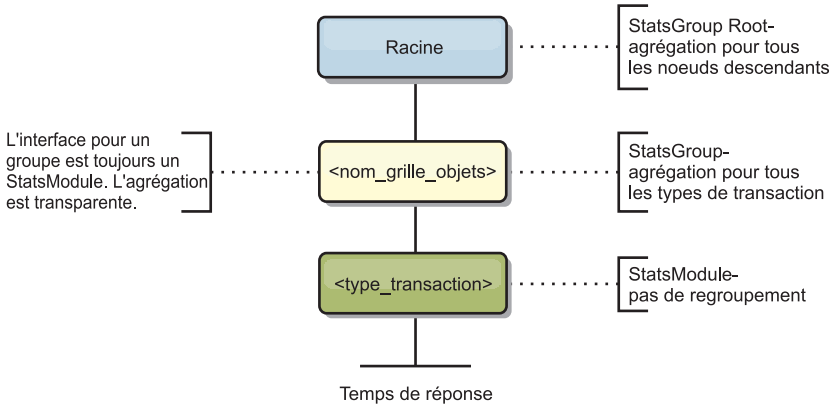


Figure 4. Structure de module ObjectGridModule

Le diagramme ci-après illustre un exemple de structure ObjectGridModule. Dans cet exemple, il existe deux instances ObjectGrid sur le système : ObjectGrid A et ObjectGrid B. L'instance ObjectGrid A possède deux types de transaction : A et default. L'instance ObjectGrid B ne possède que le type de transaction default.

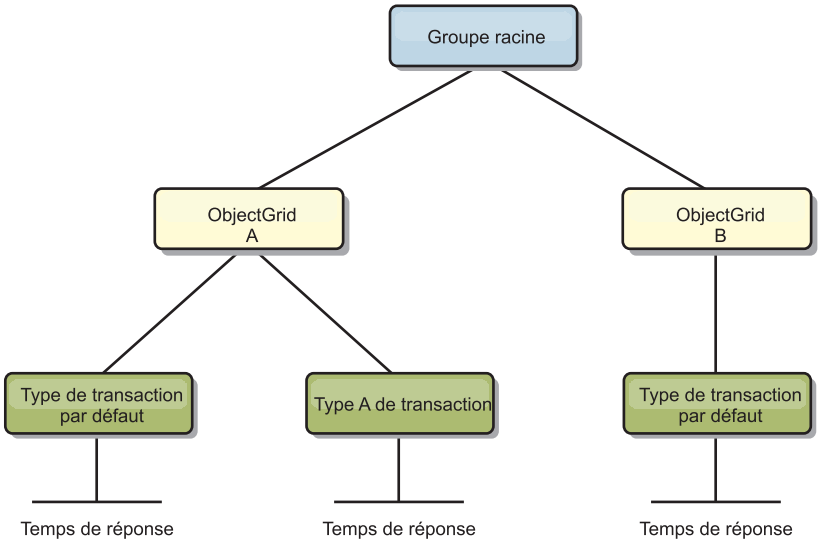


Figure 5. Exemple de structure de module ObjectGridModule

Les types de transaction sont définis par les développeurs d'applications car ils savent quels types de transaction sont utilisés par leurs applications. Le type de transaction est défini à l'aide de la méthode Session.setTransactionType(String) suivante :

```
/**
* Définit le type de transaction des transactions futures.
*
* Une fois que cette méthode a été appelée, toutes les transactions futures sont de
* même type jusqu'à ce qu'un autre type de transaction ait été défini. Si aucun type
* de transaction n'est défini, le type de transaction TRANSACTION_TYPE_DEFAULT
* par défaut est utilisé.
*
* Les types de transaction sont principalement utilisés à des fins de suivi
* des données statistiques.
* Les utilisateurs peuvent prédéfinir les types des transactions qui sont
* exécutées dans une application. L'idée consiste à regrouper les
* transactions de mêmes caractéristiques dans une même catégorie (type),
* afin qu'une statistique de temps de réponse des transactions puisse
* être utilisée pour rechercher chaque type de transaction.
*
* Ce suivi est utile si votre application possède différents types de
* transaction.
* Parmi eux, certains types de transaction, comme les transactions de mise à
* jour, possèdent un délai de traitement supérieur à celui d'autres
* transactions, telles que les transactions en lecture seule. Si le type de
* transaction est utilisé, les différentes transactions sont recherchées par des
* statistiques différentes, afin que ces dernières puissent être plus utiles.
*
* @param tranType Type de transaction des transactions futures.
*/
void setTransactionType(String tranType);
```

L'exemple suivant spécifie `updatePrice` comme type de transaction :

```
// Spécifiez le type de transaction updatePrice
// La durée entre session.begin() et session.commit() fait l'objet d'un suivi
// dans les statistiques de durée de "updatePrice".
session.setTransactionType("updatePrice");
session.begin();
map.update(stockId, new Integer(100));
session.commit();
```

La première ligne indique que le type de transaction suivant est updatePrice. Il existe une statistiques updatePrice sous l'instance ObjectGrid qui correspond à la session de l'exemple. A l'aide d'interfaces JMX (Java Management Extensions), vous pouvez obtenir le temps de réponse des transactions updatePrice. Vous pouvez également extraire les statistiques agrégées de tous les types de transaction sur l'instance ObjectGrid spécifiée.

mapModule

La structure mapModule contient trois statistiques sur les mappes eXtreme Scale :

- **Nombre d'occurrences de mappe** - *BoundedRangeStatistic* : Recherche le nombre d'occurrences d'une mappe. Le nombre d'occurrences est une valeur flottante comprise entre 0 et 100 compris, qui représente le pourcentage d'occurrences de mappe en relation avec les opérations d'extraction de mappe.
- **Nombre d'entrées** - *CountStatistic* : Recherche le nombre d'entrées dans la mappe.
- **Temps de réponse de la mise à jour par lots du chargeur** - *TimeStatistic* : Recherche le temps de réponse utilisé pour l'opération de mise à jour par lots du chargeur.

L'élément racine de la structure mapModule, "root", sert de point d'entrée aux statistiques des mappes ObjectGrid. Cet élément racine contient des ObjectGrids comme éléments enfant et ces derniers possèdent des mappes comme éléments

enfant. Trois statistiques sont répertoriées pour chaque instance de mappe. La structure mapModule est illustrée dans le diagramme suivant :

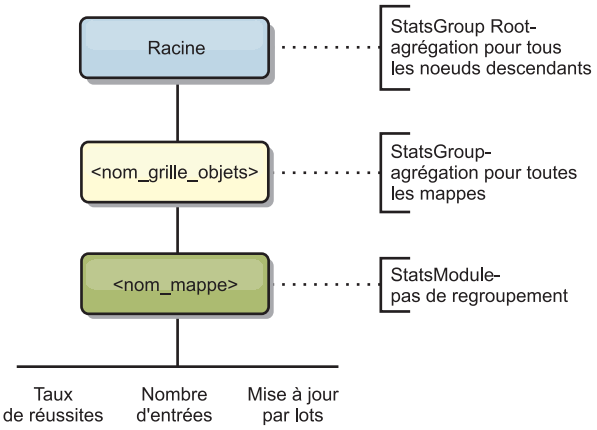
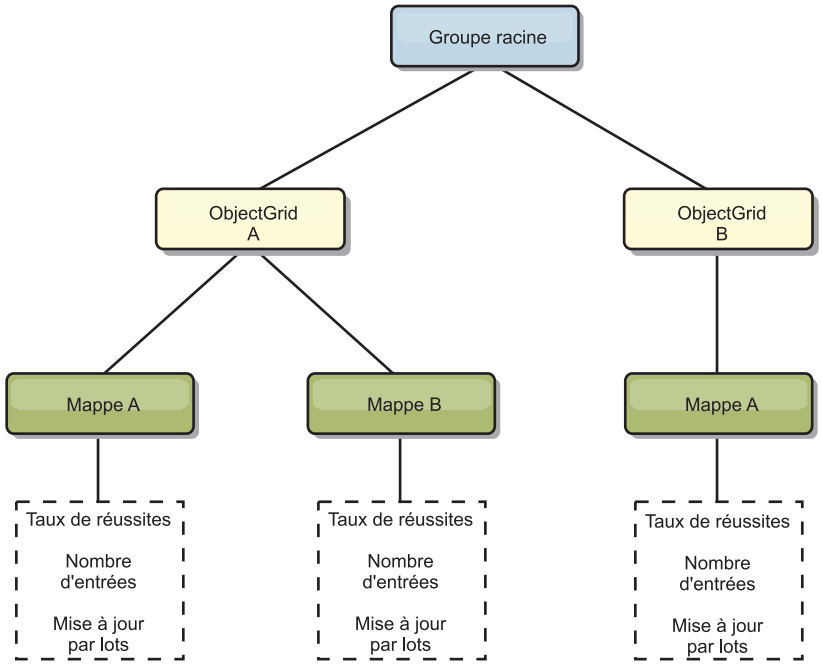


Figure 6. structure mapModule

Le diagramme suivant illustre un exemple de structure mapModule :

Figure 7. Exemple de structure de module mapModule



hashIndexModule

La structure hashIndexModule contient les statistiques suivantes sur les index de niveau mappe :

- **Nombre de recherches** - *CountStatistic* : Nombre d'appels de l'opération de recherche d'index.
- **Nombre de collisions** - *CountStatistic* : Nombre de collisions de l'opération de recherche.

- **Nombre d'échecs** - *CountStatistic* : Nombre d'échecs pour l'opération de recherche.
- **Nombre de résultats** - *CountStatistic* : Nombre de clés renvoyées par l'opération de recherche.
- **Nombre de mises à jour par lots** - *CountStatistic* : Nombre de mises à jour par lots sur cet index. Si la mappe correspondante es modifié qu'une quelconque manière, la méthode doBatchUpdate() de l'index est appelée. Cette statistiques indique la fréquence à laquelle votre index est modifié ou mis à jour.
- **Durée de recherche** - *TimeStatistic* : Temps que prend l'opération de recherche pour s'exécuter

L'élément racine de la structure hashIndexModule, "root", sert de point d'entrée aux statistiques de HashIndex. Cet élément racine contient des ObjectGrids comme éléments enfant, les ObjectGrids contiennent des mappes comme éléments enfant et enfin, ces mappes contiennent des instances HashIndex comme éléments enfant et les noeuds terminaux de l'arborescence. Trois statistiques sont répertoriées pour chaque instance HashIndex. La structure hashIndexModule est illustrée dans le diagramme suivant :

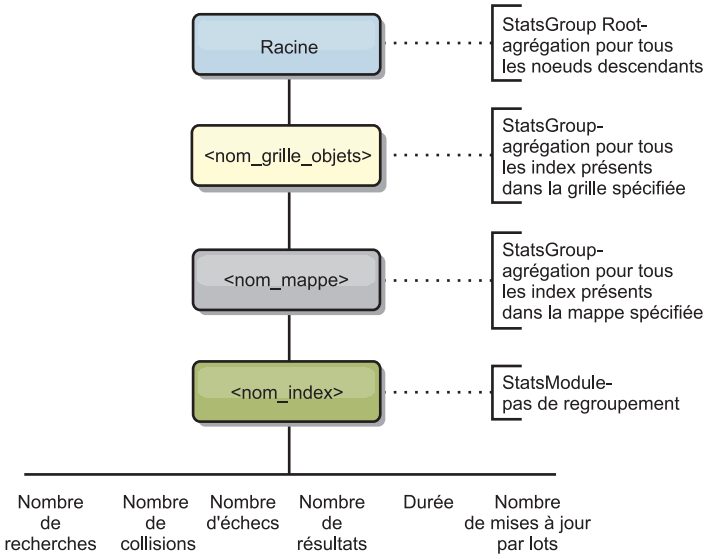


Figure 8. structure de module hashIndexModule

Le diagramme suivant illustre un exemple de structure hashIndexModule :

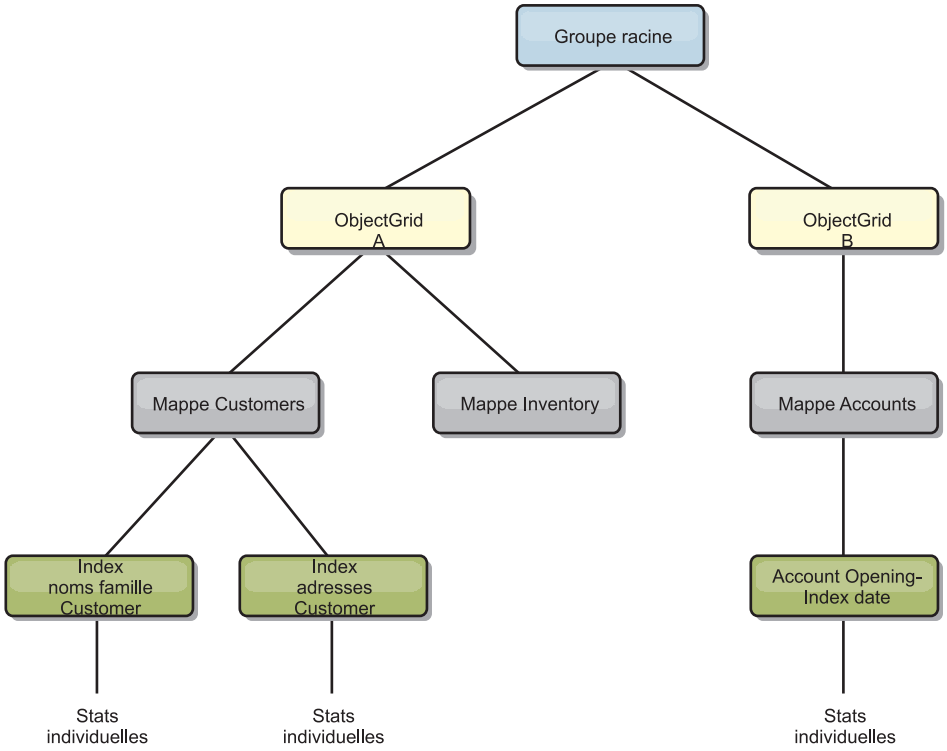


Figure 9. Exemple de structure de module hashIndexModule

agentManagerModule

La structure agentManagerModule contient les statistiques sur les agents de niveau mappe :

- **Temps de réduction** : *TimeStatistic* - Durée nécessaire pour que l'agent termine l'opération de réduction.
- **Durée totale** : *TimeStatistic* - Durée totale nécessaire à l'agent pour effectuer toutes les opérations.
- **Temps de sérialisation de l'agent** : *TimeStatistic* - Durée nécessaire pour sérialiser l'agent.
- **Temps d'inflation de l'agent** : *TimeStatistic* - Durée nécessaire pour l'inflation de l'agent sur le serveur.
- **Temps de sérialisation des résultats** : *TimeStatistic* - Durée nécessaire pour sérialiser les résultats de l'agent.
- **Temps d'inflation des résultats** : *TimeStatistic* - Durée nécessaire pour l'inflation des résultats de l'agent.
- **Nombre d'échecs** : *CountStatistic* - Nombre de fois que l'agent a échoué.
- **Nombre d'appels** : *CountStatistic* - Nombre d'appels d'AgentManager.
- **Nombre de partitions** : *CountStatistic* - Nombre de partitions vers lesquelles l'agent est envoyé.

L'élément racine de la structure agentManagerModule, "root", sert de point d'entrée aux statistiques d'AgentManager. Cet élément racine contient des ObjectGrids comme éléments enfant, les ObjectGrids contiennent des mappes comme éléments enfant et enfin, ces mappes contiennent des instances AgentManager comme éléments enfant et les noeuds terminaux de l'arborescence. Trois statistiques sont

répertoriées pour chaque instance AgentManager.

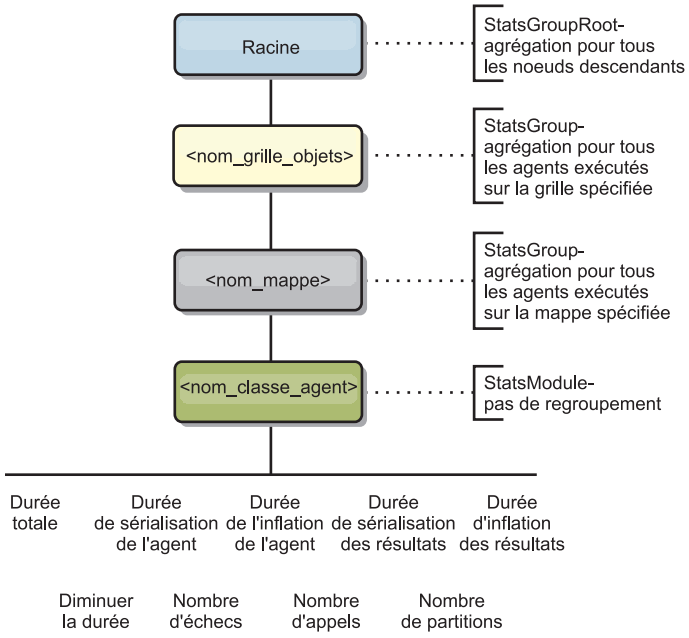


Figure 10. Structure agentManagerModule

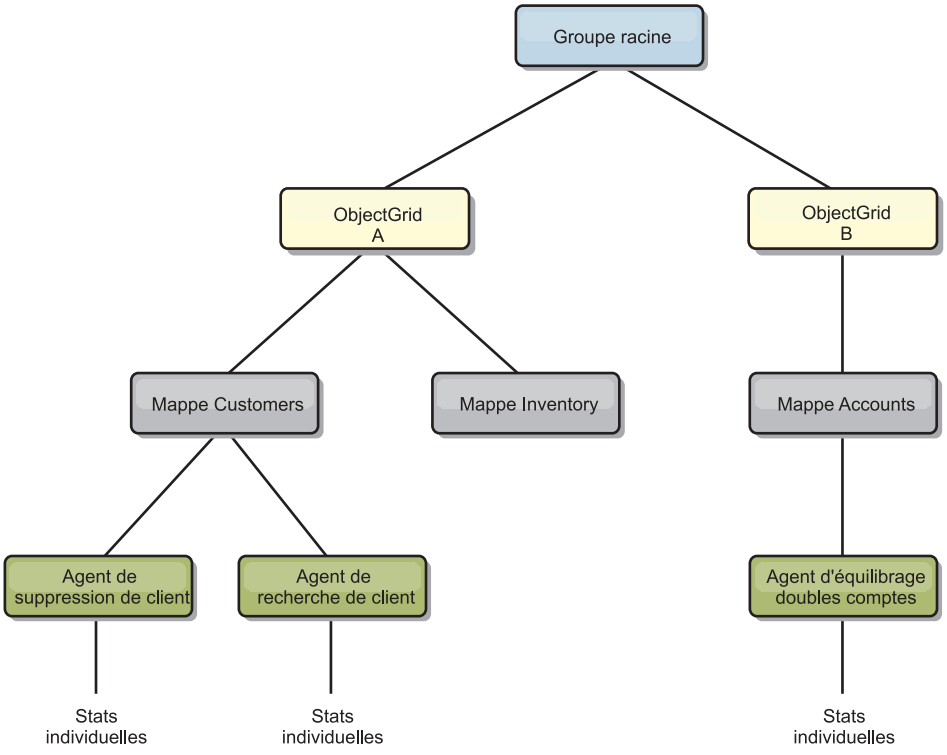


Figure 11. Exemple de structure agentManagerModule

queryModule

La structure queryModule contient les statistiques sur les requêtes eXtreme Scale :

- **Temps de création du plan** : *TimeStatistic* - Durée nécessaire pour créer le plan de requête.

- **Temps d'exécution** : *TimeStatistic* - Durée nécessaire pour exécuter la requête.
- **Nombre d'exécutions** : *CountStatistic* - Nombre de fois que la requête a été exécutée.
- **Nombre de résultats** : *CountStatistic* - Nombre de résultats pour chaque ensemble de résultats de chaque exécution de requête.
- **Nombre d'échecs** : *CountStatistic* - Nombre de fois que la requête a échoué.

L'élément racine de la structure queryModule, "root", sert de point d'entrée aux statistiques des requêtes. Cet élément racine contient des ObjectGrids comme éléments enfant et ces derniers possèdent des objets de requête comme éléments enfant et les noeuds terminaux de l'arborescence. Trois statistiques sont répertoriées pour chaque instance de requête.

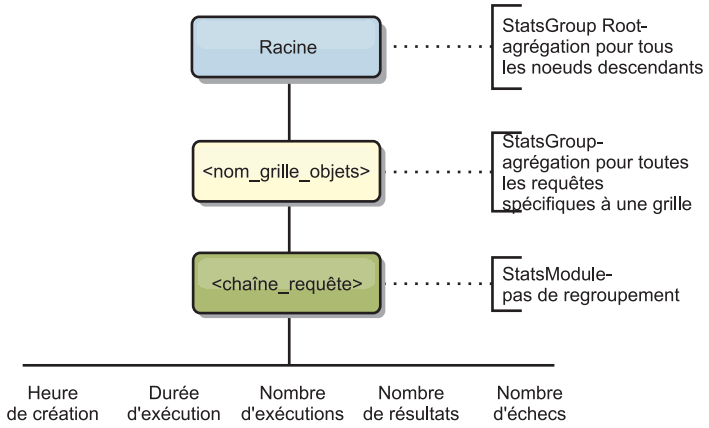


Figure 12. structure queryModule

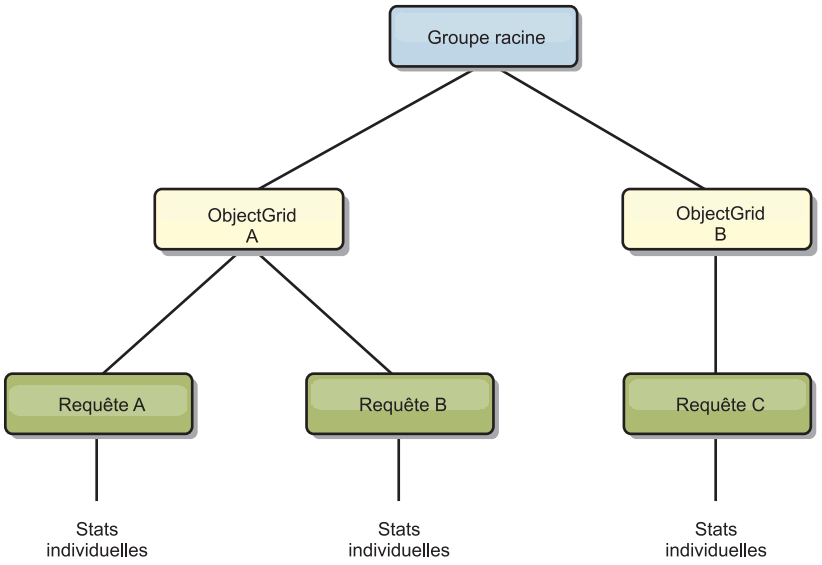


Figure 13. Exemple de structure queryModule QueryStats.jpg

Accès aux beans gérés à l'aide de l'outil wsadmin

Vous pouvez utiliser l'utilitaire wsadmin fourni dans WebSphere Application Server pour accéder aux informations sur les beans gérés.

Exécutez l'outil wsadmin depuis le répertoire bin de votre installation WebSphere Application Server. L'exemple suivant restaure une vue de la position actuelle du fragment dans un logiciel eXtreme Scale dynamique. Vous pouvez exécuter wsadmin depuis n'importe quelle installation où eXtreme Scale est en cours d'exécution. Vous n'avez pas besoin de l'exécuter sur le service de catalogue.

```
$ wsadmin.sh -lang jython
wsadmin>placementService = AdminControl.queryNames
 ("com.ibm.websphere.objectgrid:*,type=PlacementService")
wsadmin>print AdminControl.invoke(placementService,
 "listObjectGridPlacement","library ms1")

<objectGrid name="library" mapSetName="ms1">
  <container name="container-0" zoneName="DefaultDomain"
  hostName="host1.company.org" serverName="server1">
     <shard type="Primary" partitionName="0"/>
     <shard type="SynchronousReplica" partitionName="1"/>
  </container>
  <container name="container-1" zoneName="DefaultDomain"
  hostName="host2.company.org" serverName="server2">
     <shard type="SynchronousReplica" partitionName="0"/>
     <shard type="Primary" partitionName="1"/>
  </container>
  <container name="UNASSIGNED" zoneName="_ibm_SYSTEM"
  hostName="UNASSIGNED" serverName="UNNAMED">
    <shard type="SynchronousReplica" partitionName="0"/>
    <shard type="AsynchronousReplica" partitionName="0"/>
  </container>
</objectGrid>
```

Chapitre 7. Programmation de l'intégration de JPA

La Java Persistence API (JPA) est une spécification de mappage des objets Java à des bases de données relationnelles. JPA contient une spécification ORM (Object-Relational Mapping) complète utilisant des annotations de métadonnées de langage Java, des descripteurs XML ou les deux pour définir le mappage entre les objets Java et une base de données relationnelle. Un certain nombre d'implémentations commerciales et de code source ouvert sont disponibles.

Pour utiliser JPA, vous devez disposer d'un fournisseur JPA pris en charge, tel qu'OpenJPA ou Hibernate, des fichiers JAR et un fichier `META-INF/persistence.xml` dans votre chemin d'accès aux classes.

Loaders JPA

La Java Persistence API (JPA) est une spécification de mappage des objets Java à des bases de données relationnelles. JPA contient une spécification ORM (Object-Relational Mapping) complète utilisant des annotations de métadonnées de langage Java, des descripteurs XML ou les deux pour définir le mappage entre les objets Java et une base de données relationnelle. Un certain nombre d'implémentations commerciales et de code source ouvert sont disponibles.

Vous pouvez utiliser une implémentation de plug-in de chargeur Java Persistence API (JPA) avec eXtreme Scale pour interagir avec les bases de données prises en charge par le chargeur choisi. Pour utiliser JPA, vous devez disposer d'un fournisseur JPA pris en charge, tel qu'OpenJPA ou Hibernate, des fichiers JAR et un fichier `META-INF/persistence.xml` dans votre chemin d'accès aux classes.

Les plug-in JPALoader com.ibm.websphere.objectgrid.jpa.JPALoader et JPAEntityLoader com.ibm.websphere.objectgrid.jpa.JPAEntityLoader sont deux plug-in pré-intégrés de chargeur JPA qui permettent de synchroniser les mappes ObjectGrid avec une base de données. Pour utiliser cette fonction, vous devez disposer d'une implémentation JPA, comme Hibernate ou OpenJPA. La base de données peut correspondre à tout programme d'arrière plan prise en charge par le fournisseur JPA choisi.

Vous pouvez utiliser le plug-in JPALoader lorsque vous stockez des données à l'aide de l'API ObjectMap. Utilisez le plug-in JPAEntityLoader lorsque vous stockez des données à l'aide de l'API EntityManager.

Architecture du chargeur JPA

Le chargeur JPA est utilisé pour les mappes eXtreme Scale qui stockent des objets Java simples (Plain Old Java Objects - POJO).

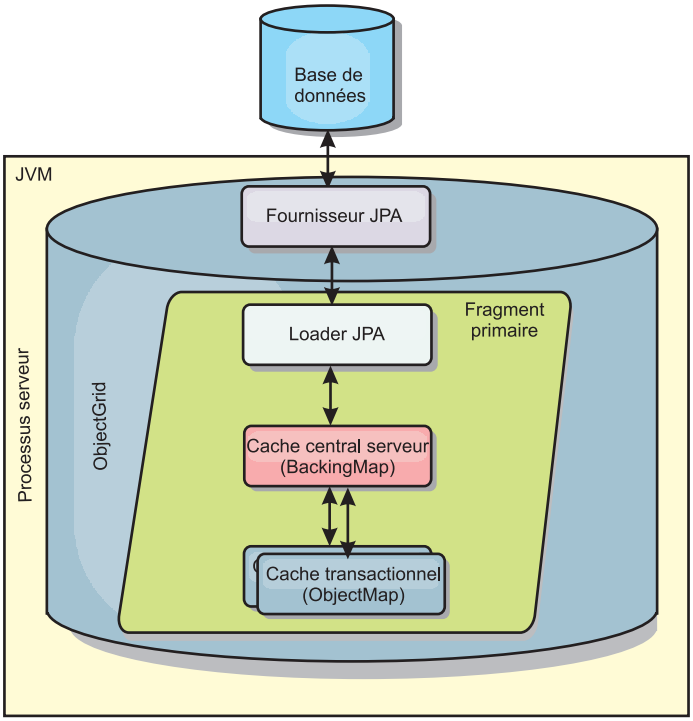


Figure 14. Architecture du chargeur JPA

Lorsqu'une méthode ObjectMap.get(Object key) est appelée, l'exécution eXtreme Scale vérifie tout d'abord si l'entrée est contenue dans la couche de l'ObjectMap. Si ce n'est pas le cas, l'exécution délègue la demande au chargeur JPA. Sur demande de chargement de la clé, JPALoader appelle la méthode JPA EntityManager.find(Object key) pour trouver les données à partir du chargeur JPA. Si les données sont contenues dans le gestionnaire d'entités JPA, elles sont renvoyées ; dans le cas contraire, le fournisseur JPA interagit avec la base de données pour obtenir la valeur.

Lors d'une mise à jour de l'ObjectMap, par exemple à l'aide de la méthode ObjectMap.update(Object key, Object value), l'exécution eXtreme Scale crée un élément LogElement pour cette mise à jour et l'envoie au JPALoader. Le JPALoader appelle la méthode JPA EntityManager.merge(Object value) pour mettre à jour la valeur dans la base de données.

Le plug-in JPAEntityLoader implique les quatre mêmes couches. Toutefois, étant donné que le plug-in JPAEntityLoader est utilisé pour les mappes qui stockent des entités eXtreme Scale, les relations entre les entités peuvent compliquer le scénario d'usage. Une entité eXtreme Scale se distingue d'une entité JPA. Pour plus d'informations, consultez «Plug-in JPAEntityLoader», à la page 261.

Méthodes

Les chargeurs présentent trois méthodes principales :

1. get : renvoie une liste de valeurs qui correspond à l'ensemble des clés qui sont transmises par l'extraction des données à l'aide de JPA. La méthode utilise JPA pour trouver les entités dans la base de données. Pour le plug-in JPALoader, la liste renvoyée contient une liste des entités JPA extraite directement de

l'opération find. Pour le plug-in JPAEntityLoader, la liste renvoyée contient des tuples de valeur d'entité eXtreme Scale qui sont convertis à partir des entités JPA.

2. batchUpdate : écrit les données des mappes ObjectGrid dans la base de données. En fonction des différents types d'opérations (insert, update ou delete), le chargeur utilise les opérations JPA persist, merge et remove pour mettre à jour les données dans la base de données. Pour le JPALoader, les objets de la mappe sont directement utilisés en tant qu'entités JPA. Pour le JPAEntityLoader, les tuples d'entité de la mappe sont convertis en objets utilisés en tant qu'entités JPA.
3. preloadMap : précharge la mappe à l'aide de la méthode de chargeur client ClientLoader.load. Pour les mappes partitionnées, la méthode preloadMap est uniquement appelée dans une seule partition. La partition est spécifiée par la propriété preloadPartition de la classe JPALoader ou JPAEntityLoader. Si la valeur preloadPartition est inférieure à zéro ou supérieure à (*nombre_total_de_partitions* - 1), le préchargement est désactivé.

Les plug-in JPALoader et JPAEntityLoader s'associent à la classe JPATxCallback pour coordonner les transactions eXtreme Scale et les transactions JPA. La classe JPATxCallback doit être configurée dans l'instance ObjectGrid pour utiliser ces deux chargeurs.

Présentation de l'utilitaire de préchargement client JPA

L'utilitaire de préchargement client JPA (Java Persistence API) charge des données dans les mappes de sauvegarde eXtreme Scale en utilisant une connexion client à ObjectGrid.

Cette fonction peut simplifier le chargement des mappes eXtreme Scale lorsque les requêtes à la base de données ne peuvent pas être partitionnées. Un programme de chargement, tel qu'un chargeur JPA peut également être utilisé et constitue la solution idéale lorsque les données peuvent être chargées en parallèle.

L'utilitaire de préchargement client JPA peut utiliser les implémentations OpenJPA ou Hibernate JPA pour charger ObjectGrid depuis une base de données. Etant donné que WebSphere eXtreme Scale n'interagit pas directement avec la base de données ou l'API JDBC (Java Database Connectivity), toutes les bases de données prises en charge par OpenJPA ou Hibernate peuvent être utilisées pour charger ObjectGrid.

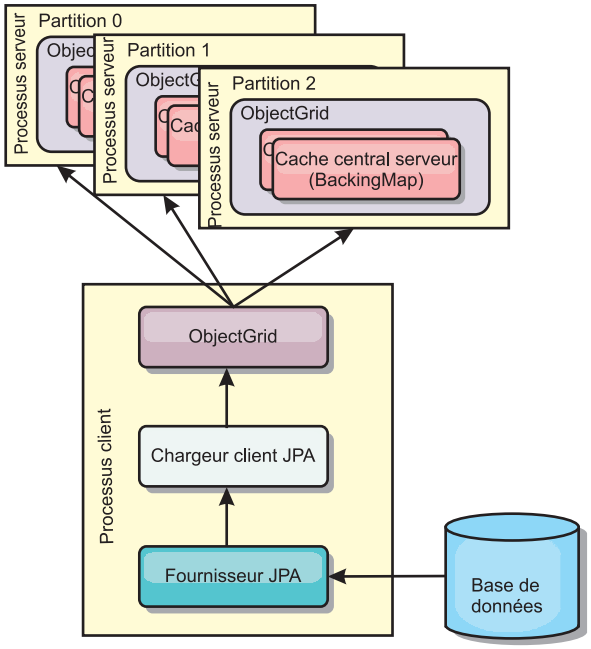


Figure 15. Chargeur client qui utilise l'implémentation JPA pour charger ObjectGrid

Une application utilisateur fournit généralement au chargeur client un nom d'unité de persistance, un nom de classe entité et une requête JPA. Le chargeur client extrait le gestionnaire d'entités JPA en fonction du nom d'unité de persistance, utilise le gestionnaire d'entités pour interroger les données à partir de la base de données avec la classe entité et la requête JPA fournies et enfin, charge les données dans les mappes ObjectGrid réparties. Lorsque la requête implique des relations à plusieurs niveaux, vous pouvez optimiser les performances à l'aide d'une chaîne de requête personnalisée. Une mappe de propriétés de persistance peut éventuellement être spécifiée pour remplacer les propriétés de persistance configurées.

Un chargeur client peut charger des données selon deux modes différents, tel qu'illustré dans le tableau suivant :

Tableau 12. Modes du chargeur client

Mode	Description
Préchargement	Efface et charge toutes les entrées dans la mappe de sauvegarde. Si la mappe est une mappe d'entités, toutes les mappes d'entités connexes seront également effacées si l'option CascadeType.REMOVE d'ObjectGrid est activée.
Rechargement	La requête JPA est exécutée sur ObjectGrid pour invalider toutes les entités de la mappe qui correspondent à la requête. Si la mappe est une mappe d'entités, toutes les mappes d'entités connexes seront également effacées si l'option CascadeType.INVALIDATE d'ObjectGrid est activée.

Dans tous les cas, une requête JPA est utilisée pour sélectionner et charger les entités voulues de la base de données et pour les stocker dans les mappes

ObjectGrid. Si la mappe ObjectGrid n'est pas une mappe d'entités, les entités JPA seront détachées et stockées directement. Si la mappe ObjectGrid est une mappe d'entités, les entités JPA sont stockées en tant que tuples d'entités ObjectGrid. Vous pouvez spécifier une requête JPA ou utiliser la requête par défaut `select o from EntityName o`.

Pour en savoir plus sur la configuration de l'utilitaire de préchargement client JPA, consultez les informations correspondantes dans le *Guide de programmation*

Programmation de l'utilitaire de préchargement client JPA

L'utilitaire de préchargement client JPA (Java Persistence API) charge des données dans les mappes de sauvegarde eXtreme Scale en utilisant une connexion client à ObjectGrid. Vous pouvez implémenter le préchargement et le rechargement des données dans votre application.

Utilisation de l'interface StateManager

Utilisez la méthode setObjectGridState de l'interface StateManager pour affecter à l'état d'ObjectGrid l'une des valeurs suivantes : OFFLINE, ONLINE, QUIESCE ou PRELOAD. L'interface StateManager empêche les autres clients d'accéder à l'ObjectGrid, tant qu'il n'est pas en ligne.

Par exemple, affectez à l'état d'ObjectGrid la valeur PRELOAD avant de charger les mappes avec des données. Une fois que le chargement des données est terminé, réaffectez à l'état d'ObjectGrid la valeur ONLINE. Pour plus d'informations, voir les informations sur la définition de la disponibilité d'un ObjectGrid, dans le *Guide d'administration*.

Si vous préchargez des mappes différentes dans un ObjectGrid, affectez à cet ObjectGrid l'état `PRELOAD`, puis réaffectez-lui l'état `ONLINE` une fois que toutes les mappes ont terminé le chargement des données. Cette coordination peut être effectuée par l'interface ClientLoadCallback. Affectez à l'ObjectGrid l'état `PRELOAD` après la première notification preStart de l'instance ObjectGrid, puis réaffectez-lui l'état `ONLINE` après la dernière notification postFinish.

Si vous avez besoin de précharger des mappes à partir de différentes machines virtuelles Java, vous devez effectuer une coordination entre les multiples JVM. Si vous préchargez des mappes différentes dans un Affectez à cet ObjectGrid l'état `PRELOAD` une fois avant le préchargement de la première mappe dans l'une des machines virtuelles Java, puis réaffectez-lui l'état `ONLINE` une fois que toutes les mappes ont terminé le chargement des données sur toutes les JVM.

Exemple de préchargement basé sur le client

Le flux du préchargement des données est le suivant :

1. Effacez le contenu de la mappe à précharger. Dans le cas d'une mappe d'entités, si une relation est configurée comme une suppression en cascade, le contenu des mappes associées est également supprimé.
2. Exécutez la requête sur JPA pour les entités d'un lot. La taille du lot est de 1000.
3. Pour chaque lot, générez une liste de clés et une liste de valeurs pour chaque partition.
4. Pour chaque partition, appelez l'agent de grille de données pour insérer ou mettre à jour directement les données côté serveur si elles se trouvent sur un

client eXtreme Scale. Si la grille est une instance locale, insérez ou mettez à jour directement les données dans les mappes ObjectGrid.

L'exemple de fragment de code ci-après illustre un chargement de client simple.

```
// Extrayez le gestionnaire d'états
StateManager stateMgr = StateManagerFactory.getStateManager();

// Affectez à l'état ObjectGrid la valeur PRELOAD avant d'appeler
// ClientLoader.loader
stateMgr.setObjectGridState(AvailabilityState.PRELOAD, objectGrid);

ClientLoader c = ClientLoaderFactory.getClientLoader();

// Charge les données
c.load(objectGrid, "CUSTOMER", "custPU", null,
    null, null, null, true, null);

// Réaffectez à l'état ObjectGrid la valeur ONLINE
stateMgr.setObjectGridState(AvailabilityState.ONLINE, objectGrid);
```

Dans cet exemple, la mappe CUSTOMER est configurée comme une mappe d'entités. La classe d'entité Customer, qui est configurée dans le fichier XML du descripteur des métadonnées d'entité ObjectGrid possède une relation one-to-many avec les entités Order. L'option CascadeType.ALL de l'entité Customer est activée sur la relation avec l'entité Order.

Avant l'appel de la méthode ClientLoader.load, l'état ObjectGrid a pour valeur PRELOAD.

Voici les paramètres utilisés dans la méthode ClientLoader.load :

1. **objectGrid** : Instance ObjectGrid. Il s'agit d'une instance ObjectGrid côté client.
2. **"CUSTOMER"** : Mappe à charger. Le client possédant une relation cascade-all avec les entités Order, ces dernières sont également chargées.
3. **"custPU"** : Nom de l'unité de persistance JPA pour les entités Customer et Order.
4. **null** : La mappe persistenceProps est null, ce qui signifie que les propriétés de persistance par défaut configurées dans le fichier persistence.xml seront utilisées.
5. **null** : La classe d'entité est configurée avec la valeur null. Elle prend la valeur de la classe d'entité configurée dans le fichier XML du descripteur des métadonnées d'entité ObjectGrid pour la mappe "CUSTOMER", dans le cas présent, Customer.class.
6. **null** : Le loadSql est null, ce qui signifie que la valeur par défaut "select o from CUSTOMER o" est utilisée pour interroger les entités JPA.
7. **null** : La mappe des paramètres de requête est null.
8. **true** : Indique que le mode de chargement des données est le préchargement. Par conséquent, les opérations clear seront appelées sur les mappes CUSTOMER et ORDER pour supprimer toutes les données avant le chargement en raison de leur relation de suppression en cascade.
9. **null** : ClientLoaderCallback a la valeur null.

Pour plus d'informations sur les paramètres requis, voir l'API ClientLoader dans la documentation de l'API.

Exemple de rechargement

Recharger une mappe revient à précharger une mappe, mais en affectant à l'argument isPreload la valeur false dans la méthode ClientLoader.load.

En mode rechargement, le flux du chargement de données est le suivant :

1. Exécutez la requête fournie sur la mappe ObjectGrid et invalidez tous les résultats. Dans le cas d'une mappe d'entités, si une relation est configurée avec l'option CascadeType.INVALIDATE, les entités associées sont également invalidées dans leurs mappes.
2. Exécutez la requête fournie sur l'API JPA pour interroger les entités JPA par lots. La taille du lot est de 1000.
3. Pour chaque lot, générez une liste de clés et une liste de valeurs pour chaque partition.
4. Pour chaque partition, appelez l'agent de grille de données pour insérer ou mettre à jour directement les données côté serveur si elles se trouvent sur un client eXtreme Scale. Si la grille est une configuration eXtreme Scale locale, insérez ou mettez à jour directement les données dans les mappes ObjectGrid.

Voici un exemple de rechargement :

```
// Extrayez le gestionnaire d'états
StateManager stateMgr = StateManagerFactory.getStateManager();

// Affectez à l'état ObjectGrid la valeur PRELOAD avant d'appeler
// ClientLoader.loader
stateMgr.setObjectGridState(AvailabilityState.PRELOAD, objectGrid);

ClientLoader c = ClientLoaderFactory.getClientLoader();

// Charge les données
String loadSql = "select c from CUSTOMER c
    where c.custId >= :startCustId and c.custId < :endCustId ";
Map<String, Long> params = new HashMap<String, Long>();
params.put("startCustId", 1000L);
params.put("endCustId", 2000L);

c.load(objectGrid, "CUSTOMER", "customerPU", null, null,
    loadSql, params, false, null);

// Réaffectez à l'état ObjectGrid la valeur ONLINE
stateMgr.setObjectGridState(AvailabilityState.ONLINE, objectGrid);
```

Cet exemple diffère de l'exemple de préchargement dans la mesure ou un loadSql et ses paramètres sont fournis. Cet exemple ne recharge que les données client dont l'id est compris entre 1000 et 2000.

Notez que cette chaîne de requête respecte à la fois la syntaxe de la requête JPA et la syntaxe de la requête d'entité eXtreme Scale. Cette chaîne de requête est importante car elle est exécutée deux fois ; sur ObjectGrid pour invalider les entités ObjectGrid renvoyées, puis sur le JPA pour charger les entités JPA renvoyées.

Appel d'un chargeur client dans une implémentation de chargeur

L'interface Loader contient une méthode preload :

```
void preloadMap(Session session, BackingMap backingMap) throws
LoaderException;
```

Cette méthode indique au chargeur de précharger les données dans la mappe. Une implémentation de chargeur peut utiliser un chargeur client pour précharger les données dans toutes ses partitions. Par exemple, le chargeur JPA utilise le chargeur client pour précharger les données dans la mappe.

Pour plus d'informations, voir la rubrique relative à la présentation des chargeurs JPA dans la *Présentation du produit*.

Vous trouverez ci-après un exemple de préchargement de la mappe à l'aide du chargeur du client dans la méthode preloadMap. L'exemple vérifie d'abord si le numéro de partition actuel est identique à celui de la partition de préchargement. S'il ne l'est pas, aucune action n'est effectuée. Si les numéros des partitions correspondent, le chargeur du client est appelé pour charger les données dans les mappes. Il est important d'appeler le chargeur du client dans une et une seule partition.

```
ObjectGrid og = session.getObjectGrid();
int partitionId = backingMap.getPartitionId();
int numPartitions = backingMap.getPartitionManager().getNumOfPartitions();

// N'appelle les données du chargeur client que dans une partition
if (partitionId == preloadPartition) {

    ClientLoader loader = ClientLoaderFactory.getClientLoader();

    // Apelle le chargeur client pour charger les données
    try {

        loader.load(og, backingMap.getName(), txCallback.getPersistenceUnitName(),
     null, entityClass, null, null, true, null);
    } catch (ObjectGridException e) {
        LoaderException le = new LoaderException("Exception caught in ObjectMap " +
     ogName + "." + mapName);
        le.initCause(e);
        throw le;
    }
}
```

Remarque : Configurez l'attribut "preloadMode" de la BackingMap en lui donnant la valeur true, de manière à ce que la méthode preload soit exécutée de de manière asynchrone. Faute de quoi, la méthode preload bloquera l'activation de l'instance ObjectGrid.

Chargement manuel du client

La méthode ClientLoader.load offre une fonction de chargement client qui convient dans la plupart des cas. Toutefois, si vous souhaitez charger des données sans la méthode ClientLoader.load, vous pouvez implémenter votre propre préchargement.

Voici un modèle de chargement manuel de client :

```
// Extrayez le gestionnaire d'états
StateManager stateMgr = StateManagerFactory.getStateManager();

// Affectez à l'état ObjectGrid la valeur PRELOAD avant d'appeler ClientLoader.loader
stateMgr.setObjectGridState(AvailabilityState.PRELOAD, objectGrid);


// Charge les données
...

// Réaffectez à l'état ObjectGrid la valeur ONLINE
stateMgr.setObjectGridState(AvailabilityState.ONLINE, objectGrid);
```

Si vous chargez les données à partir du côté client, le chargement des données à l'aide d'un agent DataGrid peut améliorer les performances. Si un agent DataGrid est utilisé, toutes les lectures et écritures de données sont effectuées dans le processus serveur. Vous pouvez également concevoir votre application de sorte que les agents DataGrid sur plusieurs partitions soient exécutés en parallèle, afin d'améliorer encore les performances.

Pour implémenter le préchargement des données avec un agent DataGrid, voir l'exemple ci-après.

Une fois que vous avez créé l'implémentation de préchargement des données, vous pouvez créer un chargeur générique pour effectuer les tâches suivantes :

1. Interroger les données de la base de données par lots.
2. Générer une liste de clés et une liste de valeurs pour chaque partition.
3. Pour chaque partition, appeler la méthode agentMgr.callReduceAgent(agent, aKey) pour exécuter l'agent du serveur dans une unité d'exécution. En exécutant l'agent dans une unité d'exécution, vous pouvez exécuter simultanément les agents sur plusieurs partitions.

Exemple : Préchargement des données avec un agent DataGrid

Si vous chargez les données à partir du côté client, le chargement des données à l'aide d'un agent DataGrid peut améliorer les performances. Si un agent DataGrid est utilisé, toutes les lectures et écritures de données sont effectuées dans le processus serveur. Vous pouvez également concevoir votre application de sorte que les agents DataGrid sur plusieurs partitions soient exécutés en parallèle, afin d'améliorer encore les performances.

Voici un exemple illustrant comment charger les données avec un agent DataGrid :

```
import java.io.Externalizable;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectOutput;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;

import com.ibm.websphere.objectgrid.NoActiveTransactionException;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectGridRuntimeException;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.TransactionException;
import com.ibm.websphere.objectgrid.datagrid.ReduceGridAgent;
import com.ibm.websphere.objectgrid.em.EntityManager;

public class InsertAgent implements ReduceGridAgent, Externalizable {

    private static final long serialVersionUID = 6568906743945108310L;

    private List keys = null;

    private List vals = null;

    protected boolean isEntityMap;

    public InsertAgent() {
    }
```

```
public InsertAgent(boolean entityMap) {
    isEntityMap = entityMap;
}

public Object reduce(Session sess, ObjectMap map) {
    throw new UnsupportedOperationException(
        "ReduceGridAgent.reduce(Session, ObjectMap)");
}

public Object reduce(Session sess, ObjectMap map, Collection arg2) {
    Session s = null;
    try {
        s = sess.getObjectGrid().getSession();
        ObjectMap m = s.getMap(map.getName());
        s.beginNoWriteThrough();
        Object ret = process(s, m);
        s.commit();
        return ret;
    } catch (ObjectGridRuntimeException e) {
        if (s.isTransactionActive()) {
            try {
                s.rollback();
            } catch (TransactionException e1) {
            } catch (NoActiveTransactionException e1) {
            }
        }
        throw e;
    } catch (Throwable t) {
        if (s.isTransactionActive()) {
            try {
                s.rollback();
            } catch (TransactionException e1) {
            } catch (NoActiveTransactionException e1) {
            }
        }
        throw new ObjectGridRuntimeException(t);
    }

}

public Object process(Session s, ObjectMap m) {
    try {

        if (!isEntityMap) {
            // Dans le cas des objets Java simples, c'est très simple :
            // il suffit de tout placer dans la
            // mappe à l'aide d'insert
            insert(m);
        } else {
            // 2. Cas des entités.
            // Dans le cas des entités, nous pouvons stocker les entités
            EntityManager em = s.getEntityManager();
            persistEntities(em);

        }
        return Boolean.TRUE;
    } catch (ObjectGridRuntimeException e) {
        throw e;
    } catch (ObjectGridException e) {
        throw new ObjectGridRuntimeException(e);
    } catch (Throwable t) {
        throw new ObjectGridRuntimeException(t);
    }

}

/**
```

```
 * Il s'agit en fait d'un nouveau chargement
 * @param s
 * @param m
 * @throws ObjectGridException
 */
protected void insert(ObjectMap m) throws ObjectGridException {

    int size = keys.size();

    for (int i = 0; i < size; i++) {
        m.insert(keys.get(i), vals.get(i));
    }

}


protected void persistEntities(EntityManager em) {
    Iterator<Object> iter = vals.iterator();

    while (iter.hasNext()) {
        Object value = iter.next();
        em.persist(value);
    }
}

public Object reduceResults(Collection arg0) {
    return arg0;
}

public void readExternal(ObjectInput in)
  throws IOException, ClassNotFoundException {
    int v = in.readByte();
    isEntityMap = in.readBoolean();
    vals = readList(in);
    if (!isEntityMap) {
        keys = readList(in);
    }
}

public void writeExternal(ObjectOutput out) throws IOException {
    out.write(1);
    out.writeBoolean(isEntityMap);

    writeList(out, vals);
    if (!isEntityMap) {
        writeList(out, keys);
    }

}

public void setData(List ks, List vs) {
    vals = vs;
    if (!isEntityMap) {
        keys = ks;
    }
}

/**
 * @return Returns the isEntityMap.
 */
public boolean isEntityMap() {
    return isEntityMap;
}

static public void writeList(ObjectOutput oo, Collection l)
  throws IOException {
    int size = l == null ? -1 : l.size();
```

```
        oo.writeInt(size);
        if (size > 0) {
            Iterator iter = l.iterator();
            while (iter.hasNext()) {
                Object o = iter.next();
                oo.writeObject(o);
            }
        }
    }

    public static List readList(ObjectInput oi)
      throws IOException, ClassNotFoundException {
        int size = oi.readInt();
        if (size == -1) {
            return null;
        }


        ArrayList list = new ArrayList(size);
        for (int i = 0; i < size; ++i) {
            Object o = oi.readObject();
            list.add(o);
        }
        return list;
    }
}
```

Programme de mise à jour de données JPA en fonction de la date/heure

Un programme de mise à jour de données JPA (Java Persistence API) en fonction de la date/heure met à jour les mappes ObjectGrid avec les dernières modifications apportées à la base de données.

Lorsque des modifications ont été apportées directement dans une base de données mise au premier plan par WebSphere eXtreme Scale, ces modifications ne sont pas reflétées simultanément dans la grille eXtreme Scale. Pour implémenter correctement eXtreme Scale en tant qu'espace de traitement de base de données en mémoire, prenez en compte le fait que votre grille peut être désynchronisée de la base de données. Le programme de mise à jour de base de données en fonction de la date/heure utilise la fonction SCN (System Change Number) disponible dans Oracle 10g et l'horodatage de modification de ligne de DB2 9.5 pour surveiller les modifications apportées à la base de données pour l'invalidation et la mise à jour. Le programme de mise à jour permet également aux applications de disposer d'un champ défini par l'utilisateur à cette même fin.

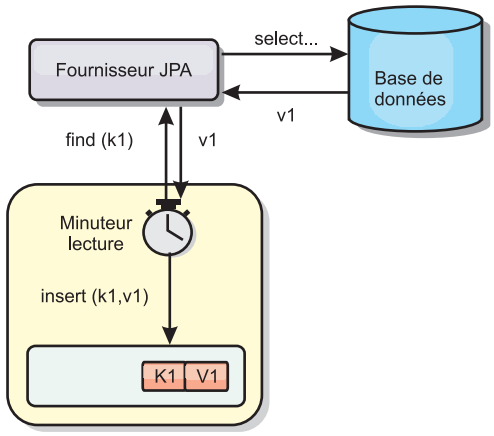


Figure 16. Actualisation régulière

>Le programme de mise à jour de base de données en fonction de la date/heure interroge régulièrement la base de données à l'aide des interfaces JPA pour obtenir les entités JPA représentant les enregistrements nouvellement insérés et mis à jour dans la base de données. Pour mettre à jour régulièrement les enregistrements, chaque enregistrement de la base de données doit comporter un horodatage pour identifier l'heure ou la séquence de dernière insertion ou mise à jour de l'enregistrement. L'horodatage ne doit pas être nécessairement au format d'horodatage. La valeur d'horodatage peut se présenter au format long ou sous la forme d'un entier si elle génère une valeur unique croissante.

Plusieurs bases de données commerciales fournissent cette fonction.

Par exemple, dans DB2 9.5, vous pouvez définir une colonne avec le format ROW CHANGE TIMESTAMP comme suit :

```
ROWCHGTS TIMESTAMP NOT NULL
      GENERATED ALWAYS
      FOR EACH ROW ON UPDATE AS
      ROW CHANGE TIMESTAMP
```

Dans Oracle, vous pouvez utiliser la pseudo-colonne **ora_rowscn**, qui représente le numéro de modification système de l'enregistrement.

Le programme de mise à jour de base de données en fonction de la date/heure met à jour les mappes ObjectGrid de trois façons différentes :

1. INVALIDATE_ONLY. Invalide les entrées dans la mappe ObjectGrid si les enregistrements correspondants de la base de données ont été modifiés.
2. UPDATE_ONLY. Met à jour les entrées dans la mappe ObjectGrid si les enregistrements correspondants de la base de données ont été modifiés. Toutefois, tous les enregistrements nouvellement insérés dans la base de données sont ignorés.
3. INSERT_UPDATE. Remplace les entrées existantes dans la mappe ObjectGrid par les dernières valeurs de la base de données. En outre, tous les enregistrements nouvellement insérés dans la base de données sont insérées dans la mappe ObjectGrid.

Pour plus d'informations sur la configuration du programme de mise à jour de données JPA en fonction de la date/heure, consultez les informations correspondantes dans le *Guide d'administration*.

Démarrage du programme de mise à jour en fonction de la date/heure

Lorsque vous démarrez le programme de mise à jour JPA (Java Persistence API) en fonction de la date/heure, les mappes ObjectGrid sont mises à jour avec les dernières modifications de la base de données.

Avant de commencer

Configurez le programme de mise à jour en fonction de la date/heure. Voir les informations sur la configuration d'un programme de mise à jour de données JPA en fonction de la date/heure, dans le *Guide d'administration*.

Pourquoi et quand exécuter cette tâche

Pour plus d'information sur le fonctionnement du programme de mise à jour en fonction de la date/heure Java Persistence API (JPA), voir «Programme de mise à jour de données JPA en fonction de la date/heure», à la page 316.

Procédure

- Démarrez un programme de mise à jour de base de données en fonction de la date/heure.
  - **Automatiquement pour un système eXtreme Scale réparti :**

    Si vous créez la configuration timeBasedDBUpdate pour la mappe de sauvegarde, le programme de mise à jour de base de données en fonction de la date/heure est automatiquement démarré lorsqu'un fragment primaire ObjectGrid réparti est activé. Pour un ObjectGrid possédant plusieurs partitions, le programme de mise à jour de base de données en fonction de la date/heure ne démarre qu'à la partition 0.
  - **Automatiquement pour un système eXtreme Scale local :**

    Si vous créez la configuration timeBasedDBUpdate pour la mappe de sauvegarde, le programme de mise à jour de base de données en fonction de la date/heure est automatiquement démarré lorsque la mappe locale est activée.
  - **Manuellement :**

    Vous pouvez également démarrer ou arrêter un programme de mise à jour de base de données en fonction de la date/heure à l'aide de l'API TimeBasedDBUpdater.

    ```
    public synchronized void startDBUpdate(ObjectGrid objectGrid, String mapName,
     String punitName, Class entityClass, String timestampField, DBUpdateMode mode) {
    ```

    1. **ObjectGrid** : Instance ObjectGrid (locale ou client).
    2. **mapName** : Nom de la mappe de sauvegarde pour laquelle le programme de mise à jour de base de données en fonction de la date/heure est démarré.
    3. **punitName** : Nom de l'unité de persistance JPA pour la création d'une fabrique de gestionnaire d'entités JPA ; la valeur par défaut est le nom de la première unité de persistance définie dans le fichier `persistence.xml`.
    4. **entityClass** : Nom de classe d'entité utilisé pour interagir avec le fournisseur JPA (Java Persistence API) ; ce nom est utilisé pour extraire les entités JPA à l'aide de requêtes d'entité.
    5. **timestampField** : Zone d'horodatage de la classe d'entité permettant d'identifier l'heure ou la séquence de la dernière mise à jour ou insertion d'un enregistrement dorsal de base de données.

6. **mode** : Mode de mise à jour de base de données en fonction de la date/heure ; un type INVALIDATE_ONLY entraîne l'invalidation des entrées de la mappe ObjectGrid si les enregistrements correspondants dans la base de données ont été modifiés ; un type UPDATE_ONLY indique de mettre à jour les entrées existantes de la mappe ObjectGrid avec les dernières valeurs de la base de données ; toutefois, tous les enregistrements nouvellement insérés dans la base de données sont ignorés ; un type INSERT_UPDATE indique de mettre à jour les entrées existantes de la mappe ObjectGrid avec les dernières valeurs de la base de données et tous les enregistrements nouvellement insérés dans la base de données sont insérés dans la mappe ObjectGrid.

Si vous souhaitez arrêter le programme de mise à jour de base de données en fonction de la date/heure, vous pouvez appeler la méthode suivante :

```
public synchronized void stopDBUpdate(ObjectGrid objectGrid, String mapName)
```

Les paramètres ObjectGrid et mapName doivent correspondre à ceux transmis dans la méthode startDBUpdate.

- Créez la zone d'horodatage dans votre base de données.
  - **DB2**

    Comme composant de la fonction de verrouillage optimiste, DB2 9.5 offre une fonction d'horodatage des modifications de ligne. Vous pouvez définir une colonne ROWCHGTS à l'aide du format ROW CHANGE TIMESTAMP, comme suit :

    ```
    ROWCHGTS TIMESTAMP NOT NULL
         GENERATED ALWAYS
         FOR EACH ROW ON UPDATE AS
         ROW CHANGE TIMESTAMP
    ```

    Par annotation ou par configuration, vous pouvez ensuite indiquer comme champ d'horodatage le champ d'entité qui correspond à cette colonne. Exemple :

    ```
    @Entity(name = "USER_DB2")
    @Table(name = "USER1")
    public class User_DB2 implements Serializable {

        private static final long serialVersionUID = 1L;

        public User_DB2() {
        }

        public User_DB2(int id, String firstName, String lastName) {
            this.id = id;
            this.firstName = firstName;
            this.lastName = lastName;
        }

        @Id
        @Column(name = "ID")
        public int id;

        @Column(name = "FIRSTNAME")
        public String firstName;

        @Column(name = "LASTNAME")
        public String lastName;

        @com.ibm.websphere.objectgrid.jpa.dbupdate.annotation.Timestamp
        @Column(name = "ROWCHGTS", updatable = false, insertable = false)
        public Timestamp rowChgTs;
    }
    ```

  - **Oracle**

Dans Oracle, il existe une pseudo-colonne ora_rowscn pour le numéro de modification système de l'enregistrement. Vous pouvez utiliser cette colonne aux mêmes fins. Exemple de l'entité qui utilise le champ ora_rowscn comme champ d'horodatage de la mise à jour de la base de données en fonction de la date/heure :

```
@Entity(name = "USER_ORA")
@Table(name = "USER1")
public class User_ORA implements Serializable  {

    private static final long serialVersionUID = 1L;

    public User_ORA() {
    }

    public User_ORA(int id, String firstName, String lastName) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    @Id
    @Column(name = "ID")
    public int id;

    @Column(name = "FIRSTNAME")
    public String firstName;

    @Column(name = "LASTNAME")
    public String lastName;

    @com.ibm.websphere.objectgrid.jpa.dbupdate.annotation.Timestamp
    @Column(name = "ora_rowscn", updatable = false, insertable = false)
    public long rowChgTs;
}
```

– **Autres bases de données**

Pour les autres types de base de données, vous pouvez créer une colonne afin de rechercher les modifications. Les valeurs de cette colonne doivent être gérées manuellement par l'application qui met à jour la table.

Prenez par exemple, une base de données Apache Derby : Vous pouvez créer une colonne "ROWCHGTS" pour rechercher les numéros de modification. En outre, le dernier numéro de modification est recherché pour cette table. Chaque fois qu'un enregistrement est inséré ou mis à jour, le dernier numéro de modification de la table est incrémenté et la valeur de la colonne ROWCHGTS de l'enregistrement est mise à jour avec ce numéro incrémenté.

```
@Entity(name = "USER_DER")
@Table(name = "USER1")
public class User_DER implements Serializable {

    private static final long serialVersionUID = 1L;

    public User_DER() {
    }

    public User_DER(int id, String firstName, String lastName) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    @Id
    @Column(name = "ID")
    public int id;
```

```
    @Column(name = "FIRSTNAME")
    public String firstName;

    @Column(name = "LASTNAME")
    public String lastName;

    @com.ibm.websphere.objectgrid.jpa.dbupdate.annotation.Timestamp
    @Column(name = "ROWCHGTS", updatable = true, insertable = true)
    public long rowChgTs;
}
```

Chapitre 8. Programmation de l'intégration à Spring

Apprenez comment intégrer vos applications eXtreme Scale à Spring.

Présentation générale de l'intégration à Spring

Spring est une infrastructure très utilisée pour le développement d'applications Java. WebSphere eXtreme Scale assure une prise en charge permettant à Spring de gérer les transactions eXtreme Scale et de configurer les clients et serveurs qui contiennent votre grille de données internes à la mémoire.

Transactions natives gérées par Spring

Spring fournit des transactions gérées par les conteneurs qui sont similaires à un serveur d'application Java Platform, Enterprise Edition. Cependant, le mécanisme Spring peut se connecter à différentes implémentations. WebSphere eXtreme Scale fournit une intégration du gestionnaire de transactions permettant à Spring de gérer les cycles de vie des transactions ObjectGrid. Voir dans le *Guide de programmation* les explications sur les transactions natives.

Prise en charge des beans d'extension et des espaces de noms gérés par Spring

En outre, eXtreme Scale s'intègre à Spring pour autoriser les beans de style Spring définis pour les points d'extension ou les plug-in. Cette fonction permet d'obtenir des configurations plus complexes et davantage de flexibilité pour la configuration des points d'extension.

En plus des beans d'extension gérés par Spring, eXtreme Scale fournit un espace de noms Spring intitulé "objectgrid". Les beans et les implémentations pré-intégrées sont prédéfinis dans cet espace de noms, ce qui facilite la configuration d'eXtreme Scale pour l'utilisateur. Voir Prise en charge des beans d'extension Spring et des espaces de nom pour des explications plus détaillées, avec un exemple de comment démarrer un serveur conteneur eXtreme Scale à l'aide de configurations Spring.

Prise en charge de la portée du fragment

Avec la configuration classique de style Spring, un bean ObjectGrid peut être de type singleton ou prototype. ObjectGrid prend aussi en charge une nouvelle portée dite "de segment". Si un bean est défini en tant que portée de fragment, seul un bean est créé par fragment. Toutes les demandes de beans avec un ou plusieurs ID correspondant à cette définition dans le même fragment entraînera le renvoi de cette instance de bean par le conteneur Spring.

L'exemple suivant montre un com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl défini avec une portée de fragment. Par conséquent, seule une instance de la classe JPAPropFactoryImpl est créée par fragment.

```
<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl" scope="shard" />
```

Spring Web Flow

Spring Web Flow stocke son état de session dans la session HTTP par défaut. Si une application est configurée pour utiliser eXtreme Scale pour la gestion de session, alors Spring l'utilise automatiquement pour stocker cet état et devient tolérante aux pannes de la même manière que la session.

Encapsulation

Les extensions eXtreme Scale Spring se trouvent dans le fichier `ogspring.jar`. Ce fichier d'archive Java (JAR) doit se trouver sur le chemin de classe pour que la prise en charge de Spring fonctionne. Si une application JEE qui s'exécute dans un WebSphere Extended Deployment augmente le WebSphere Application Server Network Deployment, alors l'application doit placer le fichier `spring.jar` et ses fichiers associés dans les modules de fichiers d'archives d'entreprise. Vous devez également placer le fichier `ogspring.jar` au même emplacement.

Transactions gérées par Spring

Spring est une infrastructure populaire permettant de développer des applications Java. WebSphere eXtreme Scale permet à Spring de gérer les transactions eXtreme Scale et de configurer les clients et les serveurs eXtreme Scale.

Transactions natives avec WebSphere eXtreme Scale

Spring fournit des transactions gérées par conteneur du style d'un serveur d'applications Java Platform, Enterprise Edition, mais le mécanisme Springs accepte des implémentations différentes. Cette rubrique décrit un gestionnaire eXtreme Scale Platform Transaction utilisable avec Spring. Les programmeurs peuvent ainsi annoter leurs objets Java simples (POJO), puis demander à Spring d'acquérir automatiquement les sessions d'eXtreme Scale et les transactions begin, commit, rollback, suspend et resume d'eXtreme Scale. Les transactions Spring sont décrites de manière plus détaillée dans le chapitre 10 de la documentation officiel de référence de Spring. Nous allons expliquer ici comment créer un gestionnaire de transactions eXtreme Scale et comment l'utiliser avec des POJO annotés. Nous expliquerons également comment utiliser cette approche avec un système eXtreme Scale client ou local et une application de style grilles de données cohabitantes.

Création d'un gestionnaire de transactions

eXtreme Scale fournit une implémentation d'un gestionnaire de transactions de plateforme Spring. Ce gestionnaire peut fournir des sessions eXtreme Scale gérées aux objets Java simples gérés par Spring. A l'aide d'annotations, Spring gère ces sessions pour les objets Java simples en termes de cycle de vie des transactions. Le fragment XML suivant montre comment créer un gestionnaire de transactions :

```
<aop:aspectj-autoproxy/>
 <tx:annotation-driven transaction-manager="transactionManager"/>

 <bean id="ObjectGridManager"
    class="com.ibm.websphere.objectgrid.ObjectGridManagerFactory"
    factory-method="getObjectGridManager"/>

 <bean id="ObjectGrid"
    factory-bean="ObjectGridManager"
    factory-method="createObjectGrid"/>

 <bean id="transactionManager"
    class="com.ibm.websphere.objectgrid.spring.ObjectGridSpringFactory"
    factory-method="getLocalPlatformTransactionManager"/>
 </bean>
```

```
<bean id="Service" class="com.ibm.websphere.objectgrid.spring.test.TestService">
 <property name="txManager" ref+"transactionManager"/>
</bean>
```

Ce code illustre le gestionnaire de transactions déclaré et connecté au bean Service qui utilisera les transactions Spring. Nous montrerons cela à l'aide d'annotations, ce qui explique la présence de la clause tx:annotation au début du code.

Obtention d'une session ObjectGrid pour la transaction Spring en cours

Un objet Java simple qui contient des méthodes gérées par Spring peut maintenant obtenir la session ObjectGrid de la transaction en cours, à l'aide du code suivant :

```
Session s = txManager.getSession();
```

Ce code renvoie la session à utiliser par l'objet Java simple. Les beans participant à la même transaction reçoivent la même session lorsqu'ils appellent cette méthode. Spring gère automatiquement la méthode begin pour la session et appelle automatiquement la méthode commit ou rollback si nécessaire. Vous pouvez également obtenir un gestionnaire d'entités ObjectGrid en appelant simplement getEntityManager à partir de l'objet Session.

Exemple d'objet Java simple utilisant des annotations

Voici un objet Java simple qui utilise des annotations pour déclarer ses intentions transactionnelles à Spring. Vous pouvez constater que la classe possède une annotation de niveau de classe indiquant que toutes les méthodes doivent utiliser par défaut la sémantique de transaction REQUIRED. La classe implémente une interface avec une méthode pour toutes les méthodes de la classe. Cela est nécessaire pour que la programmation orientée aspect (AOP) Spring fonctionne lorsqu'elle ne peut pas modifier le code intermédiaire. La classe contient une variable d'instance txManager que nous associons au gestionnaire de transactions ObjectGrid à l'aide du fichier xml Spring. Chaque méthode appelle simplement la méthode txManager.getSession pour obtenir la session à utiliser pour la méthode. La méthode queryNewTx est annotée pour indiquer une sémantique REQUIRES_NEW. Cela signifie que toute transaction existante est suspendue et qu'un transaction indépendante est créée pour cette méthode.

```
@Transactional(propagation=Propagation.REQUIRED)
public class TestService implements ITestService
{
    SpringLocalTxManager txManager;

    public TestService()
    {

    }

    public void initialize()
        throws ObjectGridException
    {
        Session s = txManager.getSession();
        ObjectMap m = s.getMap("TEST");
        m.insert("Hello", "Billy");
    }

    public void update(String updatedValue)
        throws ObjectGridException
    {
        Session s = txManager.getSession();
        System.out.println("Update using " + s);
        ObjectMap m = s.getMap("TEST");
        String v = (String)m.get("Hello");
        m.update("Hello", updatedValue);
    }

    public String query()
        throws ObjectGridException
    {
```

```
        Session s = txManager.getSession();
        System.out.println("Query using " + s);
        ObjectMap m = s.getMap("TEST");
        return (String)m.get("Hello");
    }

    @Transactional(propagation=Propagation.REQUIRES_NEW)
    public String queryNewTx()
        throws ObjectGridException
    {
        Session s = txManager.getSession();
        System.out.println("QueryTX using " + s);
        ObjectMap m = s.getMap("TEST");
        return (String)m.get("Hello");
    }

    public void testRequiresNew(ITestService bean)
        throws ObjectGridException
    {
        update("1");
        String txValue = bean.query();
        if(!txValue.equals("1"))
        {
            System.out.println("Requires didnt work");
            throw new IllegalStateException("requires didn't work");
        }
        String committedValue = bean.queryNewTx();
        if(committedValue.equals("1"))
        {
            System.out.println("Requires new didnt work");
            throw new IllegalStateException("requires new didn't work");
        }
    }

    public SpringLocalTxManager getTxManager() {
        return txManager;
    }

    public void setTxManager(SpringLocalTxManager txManager) {
        this.txManager = txManager;
    }
}
```

Définition de l'instance ObjectGrid pour une unité d'exécution

Une même machine virtuelle Java peut héberger de nombreuses instances ObjectGrid. Chaque fragment primaire placé dans une machine virtuelle Java possède sa propre instance ObjectGrid. Une machine virtuelle Java servant de client pour un ObjectGrid éloigné utilise une instance ObjectGrid renvoyée par le contexte du cluster client de la méthode de connexion pour interagir avec cette grille. Pour pouvoir appeler une méthode sur un objet Java simple à l'aide de transactions Spring pour ObjectGrid, l'unité d'exécution doit être précédée de l'instance ObjectGrid à utiliser. L'instance TransactionManager possède une méthode permettant de spécifier une instance ObjectGrid spécifique. Une fois cette instance spécifiée, les appels txManager.getSession ultérieurs renverront les sessions de cette instance ObjectGrid.

Amorçage simple pour tests

L'exemple suivant montre un exemple d'interface main permettant d'exercer cette fonctionnalité :

```
ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext(new String[]
            {"applicationContext.xml"});
    SpringLocalTxManager txManager = (SpringLocalTxManager)ctx.getBean("transactionManager");
    txManager.setObjectGridForThread(og);

    ITestService s = (ITestService)ctx.getBean("Service");
    s.initialize();
    assertEquals(s.query(), "Billy");
    s.update("Bobby");
    assertEquals(s.query(), "Bobby");
    System.out.println("Requires new test");
    s.testRequiresNew(s);
    assertEquals(s.query(), "1");
```

Nous utilisons ici un contexte d'application Spring. Ce contexte d'application permet d'obtenir une référence au gestionnaire de transactions et de spécifier un

ObjectGrid à utiliser sur cette unité d'exécution. Le code obtient ensuite une référence au service et appelle les méthodes dessus. A chaque appel de méthode à ce niveau, Spring crée une session et effectue des appels begin/commit autour de l'appel de méthode. Les éventuelles exceptions génèrent une annulation.

Interface SpringLocalTxManager

L'interface SpringLocalTxManager est implémentée par le gestionnaire de transactions de plateformes ObjectGrid et elle comporte toutes les interfaces publiques. Les méthodes de cette interface permettent de sélectionner l'instance ObjectGrid à utiliser sur une unité d'exécution et d'obtenir une session pour l'unité d'exécution. Les objets Java simples qui utilisent des transactions locales ObjectGrid doivent recevoir une référence à cette instance de gestionnaire et une seule instance doit être créée (sa portée doit être singleton). Cette instance est créée à l'aide d'une méthode statique sur ObjectGridSpringFactory. getLocalPlatformTransactionManager().

eXtreme Scale pour JTA et transactions globales

eXtreme Scale ne prend pas en charge JTA ou la validation en deux phases pour diverses raisons et notamment pour une question d'évolutivité. Par conséquent, sauf sur un dernier participant à phase unique, ObjectGrid n'interagit pas dans les transactions globales de type XA ou JTA. Ce gestionnaire de plateforme est destiné à simplifier autant que possible l'utilisation de transactions ObjectGrid locales pour les développeurs Spring.

Beans d'extension gérés par Spring

Dans le fichier `objectgrid.xml`, vous pouvez déclarer des objets POJO utilisables comme points d'extension. Si vous nommez les beans et spécifiez ensuite le nom de la classe, eXtreme Scale créera normalement des instances de la classe spécifiée et utilisera ces instances comme plug-in. eXtreme Scale peut désormais déléguer à Spring le rôle de fabrique de beans pour obtenir des instances de ces objets plug-in.

Si une application utilise Spring, de tels objets POJO doivent impérativement être connectés au reste de l'application.

Une application peut enregistrer une instance de fabrique de beans Spring à utiliser pour un ObjectGrid nommé spécifique. L'application doit créer une instance de fabrique de beans ou un contexte d'application Spring, puis l'enregistrer auprès de l'ObjectGrid à l'aide de la méthode statique suivante :

```
void registerSpringBeanFactoryAdapter(String objectGridName, Object springBeanFactory)
```

Cette méthode indique que si ObjectGrid trouve un bean d'extension (par exemple, un ObjectTransformer, Loader, TransactionCallback, etc) dont le nom de classe commence par le préfixe {spring}, le reste du nom doit être utilisé comme nom de bean Spring et l'instance de bean doit être obtenue à l'aide de la fabrique de beans Spring. ObjectGrid peut également créer une fabrique de beans Spring à partir d'un fichier XML de configuration Spring par défaut. Si aucune fabrique de beans n'a été enregistrée pour un ObjectGrid donné, cet ObjectGrid tente de rechercher un fichier XML nommé 'ObjectGridName'_spring.xml. Par exemple, si votre grille s'appelle GRID, le fichier XML sera appelé '/GRID_spring.xml' et devra se trouver dans le chemin d'accès aux classes du package racine. Si ce fichier est détecté, ObjectGrid construit un contexte d'application à l'aide de ce fichier et des beans à partir de cette fabrique de beans. Voici un exemple de nom de classe :

```
"{spring}MyLoaderBean"
```

Ce code indique à ObjectGrid de demander à Spring un bean intitulé "MyLoaderBean". Cette approche peut être utilisée pour spécifier des objets Java simples gérés par Spring pour tout point d'extension dans ObjectGrid, à partir du moment où la fabrique de beans a été enregistrée d'avance. Les extensions Spring ObjectGrid se trouvent dans le fichier ogspring.jar. Ce fichier JAR doit se trouver dans le chemin d'accès aux classes pour que le support Spring fonctionne correctement. Si une application JavaEE est exécutée dans un déploiement réseau étendu par XD, elle doit placer le fichier spring.jar et ses fichiers associés dans les modules EAR. Le fichier ogspring.jar doit également être placé au même endroit.

Prise en charge des beans d'extension Spring et des espaces de noms

WebSphere eXtreme Scale fournit une fonction permettant de déclarer des objets Java simples (POJO) afin de les utiliser en tant que points d'extension dans le fichier `objectgrid.xml` et le moyen de nommer les beans, puis de spécifier le nom de classe. En règle générale, les instances de la classe spécifiée sont créées et ces objets sont utilisés en tant que plug-in. Maintenant, eXtreme Scale peut déléguer à Spring l'obtention d'instances de ces objets de plug-in. Si une application utilise Spring, alors de tels objets Java simples doivent être connectés au reste de l'application.

Dans certains cas, vous devez utiliser Spring pour configurer certains objets de plug-in. Prenez l'exemple de la configuration suivante :

```
<objectGrid name="Grid">
    <bean id="TransactionCallback" className="com.ibm.websphere.objectgrid.jpa.JPATxCallback">
        <property name="persistenceUnitName" type="java.lang.String"  value="employeePU" />
    </bean>
    ...
</objectGrid>
```

L'implémentation pré-intégrée de TransactionCallback, classe com.ibm.websphere.objectgrid.jpa.JPATxCallback, est configurée comme classe TransactionCallback. Cette classe est configurée avec la propriété persistenceUnitName (voir l'exemple précédent). La classe JPATxCallback comporte également l'attribut JPAPropertyFactory attribute, qui est de type java.lang.Object. La configuration ObjectGrid XML ne prend pas ce type de configuration en charge.

L'intégration de eXtreme Scale Spring résout ce problème en déléguant à Spring la création des beans. La configuration révisée est comme suit :

```
<objectGrid name="Grid">
    <bean id="TransactionCallback" className="{spring}jpaTxCallback"/>
    ...
</objectGrid>
```

Le fichier Spring pour l'objet "Grid" comporte les informations suivantes :

```
<bean id="jpaTxCallback" class="com.ibm.websphere.objectgrid.jpa.JPATxCallback" scope="shard">
    <property name="persistenceUnitName" value="employeeEMPU"/>
    <property name="JPAPropertyFactory" ref ="jpaPropFactory"/>
</bean>

<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.
JPAPropFactoryImpl" scope="shard">
</bean>
```

Ici, TransactionCallback est spécifié comme `{spring}jpaTxCallback` et les beans jpaTxCallback et jpaPropFactory sont configurés dans le fichier Spring comme indiqué dans l'exemple précédent. La configuration Spring permet de configurer un bean JPAPropertyFactory en tant que paramètre de l'objet JPATxCallback.

Classe de bean Spring par défaut

Lorsque eXtreme Scale détecte un plug-in ou un bean d'extension (par exemple, ObjectTransformer, Loader ou TransactionCallback etc.) avec une valeur de nom de classe dotée du préfixe `{spring}`, alors eXtreme Scale utilise le reste du nom en tant que nom de bean Spring et obtient l'instance de bean à l'aide de la classe Spring Bean.

Par défaut, si aucune classe de bean n'a été enregistrée pour un objet ObjectGrid donné, il tente alors de trouver un fichier `ObjectGridName_spring.xml`. Par exemple, si votre grille est appelée "Grid" alors le fichier XML est appelé `/Grid_spring.xml`. Ce fichier devrait se trouver dans le chemin de classe ou dans un répertoire `META-INF` qui correspond au chemin de classe. Si ce fichier est trouvé, alors eXtreme Scale construit un ApplicationContext à l'aide de ce fichier et des beans de construction provenant de cette classe de beans.

Classe de bean Spring personnalisée

WebSphere eXtreme Scale fournit également une interface de programme d'application ObjectGridSpringFactory pour enregistrer une instance de fabrique de beans Spring pour une utilisation pour un ObjectGrid spécifiquement nommé. Cette interface de programme d'application enregistre une instance de BeanFactory dans eXtreme Scale à l'aide de la méthode statique suivante :

```
void registerSpringBeanFactoryAdapter(String objectGridName, Object
springBeanFactory)
```

Prise en charge de l'espace de noms

Depuis la version 2.0, Spring dispose d'un mécanisme pour les extensions basées sur le schéma au format XML Spring de base pour la définition et la configuration des beans. ObjectGrid utilise cette nouvelle fonction pour définir et configurer les beans ObjectGrid. Avec l'extension de schéma Spring XML, une partie des implémentations des plug-in eXtreme Scale et de certains beans ObjectGrid sont prédéfinies dans l'espace de noms "objectgrid". Lors de l'écriture des fichiers de configuration Spring, il n'est pas nécessaire de spécifier le nom de classe complet des implémentations pré-intégrées. Vous pouvez plutôt référencer les beans prédéfinis.

De plus, si vous définissez les attributs des beans dans le schéma XML, cela diminue le risque de fournir un nom d'attribut erroné. La validation XML basée sur le schéma XML peut détecter ce type d'erreurs plus tôt lors du cycle de développement.

Ces beans définis dans les extensions de schéma XML sont :

- transactionManager
- registre
- serveur
- catalogue
- catalogServerProperties
- conteneur
- JPALoader
- JPATxCallback
- JPAEntityLoader

- LRUEvictor
- LFUEvictor
- HashIndex

Ces beans sont définis dans le schéma XML objectgrid.xsd XML. Ce fichier XSD est envoyé en tant que fichier `com/ibm/ws/objectgrid/spring/namespace/objectgrid.xsd` dans le fichier `ogspring.jar`. Pour obtenir une description détaillée du fichier XSD et des beans qui y sont définis, voir les informations sur le fichier descripteur Spring dans le *Guide d'administration*.

Reprenons l'exemple utilisant JPATxCallback de la section précédente. Dans la section précédente, le bean JPATxCallback est configuré comme suit :

```
<bean id="jpaTxCallback" class="com.ibm.websphere.objectgrid.jpa.JPATxCallback" scope="shard">
    <property name="persistenceUnitName" value="employeeEMPU"/>
    <property name="JPAPropertyFactory" ref ="jpaPropFactory"/>
</bean>

<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl" scope="shard">
</bean>
```

Si elle fait appel à cette fonction d'espace de noms, la configuration Spring XML peut s'écrire comme suit :

```
<objectgrid:JPATxCallback id="jpaTxCallback" persistenceUnitName="employeeEMPU"
 jpaPropertyFactory="jpaPropFactory" />

<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl"
 scope="shard">
</bean>
```

Notez ici que plutôt que de spécifier la classe "com.ibm.websphere.objectgrid.jpa.JPATxCallback" comme dans l'exemple précédent, nous utilisons directement le bean prédéfini "objectgrid:JPATxCallback". Comme vous pouvez le voir, cette configuration est moins détaillée et facilite la détection d'erreurs.

Démarrage d'un serveur conteneur avec des beans d'extension Spring

Cet exemple illustre comment démarrer un serveur ObjectGrid à l'aide de beans d'extension gérés par ObjectGrid Spring et de la prise en charge des espaces de noms.

Fichier XML ObjectGrid

Tout d'abord, définissez un fichier ObjectGrid XML très simple qui contient un ObjectGrid "Grid" et une mappe "Test". Le fichier ObjectGrid est doté d'un plug-in ObjectGridEventListener appelé "partitionListener" et la mappe "Test" est dotée d'un expulseur appelé "testLRUEvictor". Notez que les plug-in ObjectGridEventListener et de l'expulseur sont configurés à l'aide de Spring, car leurs noms contiennent "{spring}".

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
    <objectGrids>
        <objectGrid name="Grid">
     <bean id="ObjectGridEventListener" className="{spring}partitionListener" />
            <backingMap name="Test" pluginCollectionRef="test" />
        </objectGrid>
    </objectGrids>
```

```
    <backingMapPluginCollections>
        <backingMapPluginCollection id="test">
            <bean id="Evictor" className="{spring}testLRUEvictor"/>
        </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

Fichier XML de déploiement ObjectGrid

Maintenant, créez un fichier de déploiement XML ObjectGrid simple, comme suit. Le fichier ObjectGrid est divisé en 5 partitions et aucune réplique n'est requise.

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
 xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
    <objectgridDeployment objectgridName="Grid">
        <mapSet name="mapSet" numInitialContainers="1" numberOfPartitions="5" minSyncReplicas="0"
            maxSyncReplicas="1" maxAsyncReplicas="0">
            <map ref="Test"/>
        </mapSet>
    </objectgridDeployment>
</deploymentPolicy>
```

Fichier XML Spring ObjectGrid

Maintenant, nous allons utiliser les beans d'extension gérés pas ObjectGrid Spring et les fonctions de prise en charge d'espaces de noms pour configurer les beans ObjectGrid. Le fichier XML Spring est nommé "Grid_spring.xml". Comme vous pouvez le voir, deux schémas sont inclus dans le fichier XML : spring-beans-2.0.xsd permet d'utiliser les beans gérés par Spring et objectgrid.xsd permet d'utiliser les beans prédéfinis dans l'espace de noms objectgrid.

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:objectgrid="http://www.ibm.com/schema/objectgrid"
       xsi:schemaLocation="
       http://www.ibm.com/schema/objectgrid
   http://www.ibm.com/schema/objectgrid/objectgrid.xsd
       http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <objectgrid:register id="ogregister" gridname="Grid"/>

    <objectgrid:server id="server" isCatalog="true" name="server">
        <objectgrid:catalog host="localhost" port="2809"/>
    </objectgrid:server>

    <objectgrid:container id="container"
   objectgridxml="com/ibm/ws/objectgrid/test/springshard/objectgrid.xml"
       deploymentxml="com/ibm/ws/objectgrid/test/springshard/deployment.xml"
   server="server"/>

    <objectgrid:LRUEvictor id="testLRUEvictor" numberOfLRUQueues="31"/>

    <bean id="partitionListener"
   class="com.ibm.websphere.objectgrid.springshard.ShardListener" scope="shard"/>
</beans>
```

Dans ce fichier Spring XML se trouvent 6 beans définis :

1. *objectgrid:register* : enregistre la classe de bean par défaut pour l'ObjectGrid "Grid".

2. *objectgrid:server* : définit un serveur ObjectGrid nommé "server". Ce serveur fournit également un service de catalogue, car un bean objectgrid:catalog y est imbriqué.
3. *objectgrid:catalog* : définit un point de contact de service de catalogue ObjectGrid, qui est défini sur "localhost:2809".
4. *objectgrid:container* : définit un conteneur ObjectGrid avec le fichier objectgrid XML et le fichier de déploiement XML spécifiés que nous avons évoqués précédemment. La propriété de serveur spécifie dans quel serveur ce conteneur est hébergé.
5. *objectgrid:LRUEvictor* : définit un expulseur LRUEvictor avec le nombre de files d'attente LRU à utiliser pour le définir sur 31.
6. *bean partitionListener* : définit un plug-in ShardListener. Vous devez fournir une implémentation pour ce plug-in de manière à ce qu'il ne puisse pas utiliser les beans prédéfinis. De plus, la portée du bean est définie pour "shard", ce qui signifie qu'il y a une seule instance de ShardListener par fragment ObjectGrid.

Démarrage du serveur

Le fragment ci-dessous démarre le serveur ObjectGrid, lequel héberge à la fois les services de conteneur et de catalogue. Comme nous pouvons le voir, la seule méthode à appeler pour démarrer le serveur est d'obtenir un "conteneur" de bean de la fabrique de beans. Cela simplifie la complexité de la programmation en déplaçant la plupart de la logique dans la configuration de Spring.

```
public class ShardServer extends TestCase
{
    Container container;
    org.springframework.beans.factory.BeanFactory bf;

    public void startServer(String cep)
    {
        try
        {
            bf = new org.springframework.context.support.ClassPathXmlApplicationContext(
                "/com/ibm/ws/objectgrid/test/springshard/Grid_spring.xml", ShardServer.class);
            container = (Container)bf.getBean("container");
        }
        catch (Exception e)
        {
            throw new ObjectGridRuntimeException("Cannot start OG container", e);
        }
    }

    public void stopServer()
    {
        if(container != null)
            container.teardown();
    }
}
```

Chapitre 9. Programmation de la sécurité

Les interfaces de programmation permettent de gérer les divers aspects de la sécurité dans un environnement eXtreme Scale.

API de sécurité

WebSphere eXtreme Scale adopte une architecture de sécurité ouverte. Elle offre une infrastructure de sécurité de base pour l'authentification, l'autorisation et la sécurité du transport et requiert que les utilisateurs implémentent des plug-in pour compléter l'infrastructure de sécurité.

La représentation ci-après illustre le flux de base de l'authentification et de l'autorisation d'un client pour un serveur eXtreme Scale.

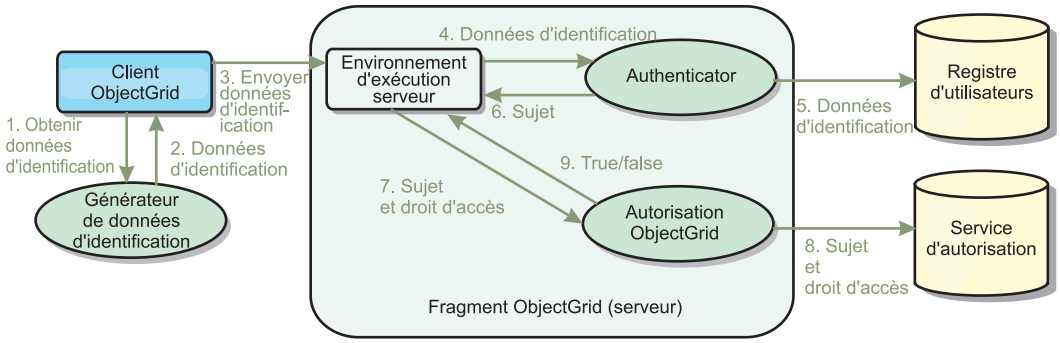


Figure 17. Flux d'authentification et d'autorisation du client

Le flux d'authentification et le flux d'autorisation se présentent comme ci-après.

Flux d'authentification

1. Le flux d'authentification commence par l'obtention de données d'identification par un client eXtreme Scale. Cette opération est effectuée par le plug-in com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator.
2. Un objet CredentialGenerator sait comment générer des données d'identification client valides (par exemple, une paire d'ID utilisateur/mot de passe, un ticket Kerberos, etc.). Ces données d'identification générées sont renvoyées au client.
3. Une fois que le client a extrait l'objet Credential à l'aide de l'objet CredentialGenerator, cet objet Credential est envoyé avec la demande eXtreme Scale au serveur eXtreme Scale.
4. Le serveur eXtreme Scale authentifie l'objet Credential avant de traiter la demande eXtreme Scale. Le serveur utilise ensuite le plug-in Authentificateur pour authentifier l'objet Credential.
5. Le plug-in Authentificateur représente une interface avec le registre d'utilisateurs. Par exemple, un serveur LDAP (Lightweight Directory Access Protocol) ou un registre d'utilisateurs de système d'exploitation. L'authentificateur consulte le registre d'utilisateurs et prend les décisions d'authentification.
6. Si l'authentification aboutit, un objet de sujet est renvoyé pour représenter ce client.

Flux d'autorisation

WebSphere eXtreme Scale adopte un mécanisme d'autorisation basé sur les droits d'accès et possède plusieurs catégories de droits d'accès représentées par différentes classes de droits d'accès. Par exemple, un objet com.ibm.websphere.objectgrid.security.MapPermission représente les droits de lecture, d'écriture, d'insertion, d'invalidation et de suppression des entrées de données d'une mappe d'objet. WebSphere eXtreme Scale prenant en charge l'autorisation JAAS (Java Authentication and Authorization Service) prête à l'emploi, vous pouvez utiliser JAAS pour gérer les autorisations en fournissant des politiques d'autorisation.

En outre, eXtreme Scale prend en charge les autorisations personnalisées. Les autorisations personnalisées sont intégrées par le plug-in com.ibm.websphere.objectgrid.security.plugins.ObjectGridAuthorization. Le flux d'autorisation du client est le suivant :

7. L'environnement d'exécution du serveur envoie l'objet de sujet et les droits requis au plug-in d'autorisation.
8. Le plug-in d'autorisation consulte le service d'autorisation et prend une décision d'autorisation. Si les droits sont octroyés à cet objet de sujet, la valeur `true` est renvoyée ; sinon, la valeur `false` est renvoyée.
9. Cette décision d'autorisation, true ou false, est renvoyée à l'environnement d'exécution du serveur.

Implémentation de la sécurité

Les rubriques de cette section expliquent comment programmer un déploiement sécurisé de WebSphere eXtreme Scale et les implémentations des plug-in. La section est organisée en fonction des diverses fonctions de sécurité. Dans chaque sous-rubrique, vous découvrirez les plug-in appropriés et la manière de les implémenter. Dans la section d'authentification, vous apprendrez à vous connecter à un environnement de déploiement WebSphere eXtreme Scale sécurisé.

Authentification du client : La rubrique sur l'authentification du client décrit comment un client WebSphere eXtreme Scale obtient des données d'identification et comment un serveur authentifie le client. Elle explique également comment un client WebSphere eXtreme Scale se connecte à un serveur WebSphere eXtreme Scale sécurisé.

Autorisation : La rubrique relative à l'autorisation explique comment utiliser ObjectGridAuthorization pour procéder à l'autorisation des clients en plus de l'autorisation JAAS.

Authentification de grille : La rubrique Authentification de grille explique comment vous pouvez utiliser SecureTokenManager pour transporter de manière sécurisée les valeurs confidentielles du serveur.

Programmation JMX (Java Management Extensions) : Si le serveur WebSphere eXtreme Scale est sécurisé, le client JMX risque de devoir envoyer des données d'identification JMX au serveur.

Programmation de l'authentification des clients

Pour l'authentification, WebSphere eXtreme Scale fournit un environnement d'exécution qui envoie les données d'identification du client vers le serveur. Le produit appelle ensuite le plug-in Authenticator.

Pour pouvoir procéder à l'authentification, WebSphere eXtreme Scale nécessite que vous implémentiez les plug-in suivants :

- Credential : le plug-in Credential représente les données d'identification d'un client (paire ID utilisateur et mot de passe, par exemple).
- CredentialGenerator : ce plug-in représente une fabrique de données d'identification chargée de générer ces données.
- Authenticator : comme son nom l'indique, ce plug-in authentifie les données d'identification du client et extrait les informations concernant ce dernier.

Plug-in Credential et CredentialGenerator

Lorsqu'un client eXtreme Scale se connecte à un serveur qui exige une authentification, le client est requis de fournir des données d'identification. Les données d'identification du client sont représentées par une interface com.ibm.websphere.objectgrid.security.plugins.Credential. Ces données d'identification peuvent être une paire nom-mot de passe, un ticket Kerberos, un certificat client ou des données au format convenu par le client et le serveur. Pour plus de détails, voir les informations concernant l'API Credential dans la documentation de l'API. Cette interface définit explicitement les méthodes equals(Object) et hashCode. Ces deux méthodes sont importantes car les objets Subject authentifiés sont mis en cache par le biais de l'objet Credential en tant que clé sur le serveur. WebSphere eXtreme Scale fournit également un plug-in pour générer des données d'identification. Ce plug-in est représenté par l'interface com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator et il est utile lorsque les données d'identification peuvent expirer. Dans ce cas, la méthode getCredential est appelée pour renouveler ces données.

L'interface Credential définit explicitement les méthodes equals(Object) et hashCode. Ces deux méthodes sont importantes car les objets Subject authentifiés sont mis en cache par le biais de l'objet Credential en tant que clé sur le serveur.

Vous pouvez également générer des données d'identification avec le plug-in fourni. Ce plug-in est représenté par l'interface com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator et il est utile lorsque les données d'identification peuvent expirer. Dans ce cas, la méthode getCredential est appelée pour renouveler ces données. Pour plus d'informations, voir la documentation de l'API.

Trois implémentations sont fournies par défaut pour les interfaces Credential :

- l'implémentation com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential qui contient une paire ID utilisateur/mot de passe
- l'implémentation com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential qui contient des clés d'authentification et d'autorisation spécifiques à WebSphere Application Server. Ces clés peuvent être utilisées pour propager les attributs de sécurité sur les serveurs d'applications appartenant au même domaine de sécurité

WebSphere eXtreme Scale fournit également un plug-in permettant de générer des données d'identification. Ce plug-in est représenté par l'interface com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator. WebSphere eXtreme Scale est livré avec deux implémentations par défaut :

- com.ibm.websphere.objectgrid.security.plugins.builtins Le constructeur UserPasswordCredentialGenerator accepte un ID utilisateur et un mot de passe. Lorsque la méthode getCredential est appelée, elle retourne un objet UserPasswordCredential qui contient l'ID utilisateur et le mot de passe
- com.ibm.websphere.objectgrid.security.plugins.builtins WSTokenCredentialGenerator représente un générateur de données d'identification (jeton de sécurité) lors d'une exécution dans WebSphere Application Server. Lorsque la méthode getCredential est appelée, le Subject associé à l'unité d'exécution en cours est extrait. Puis les informations de sécurité présentes dans cet objet Subject sont converties en objet WSTokenCredential. Vous pouvez spécifier s'il y a lieu d'extraire de l'unité d'exécution un sujet runAs ou un sujet caller à l'aide de la constante WSTokenCredentialGenerator.RUN_AS_SUBJECT ou de la constante WSTokenCredentialGenerator.CALLER_SUBJECT.

UserPasswordCredential et UserPasswordCredentialGenerator

A des fins de test, WebSphere eXtreme Scale fournit les implémentations de plug-in suivantes :

1. `com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential`
2. `com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator`

Les données d'identification utilisateur/mot de passe stockent un ID utilisateur et un mot de passe. Le générateur de ces données d'identification contient alors cet ID utilisateur et ce mot de passe.

L'exemple de code suivant montre comment implémenter ces deux plug-in.

UserPasswordCredential.java

```
// Cet exemple de programme est fourni TEL QUEL et peut être utilisé, exécuté, copié et modifié
// gratuitement par le client
// (a) à des fins d'études,
// (b) afin de développer des applications conçues pour être exécutées avec un produit IBM WebSphere,
// soit pour un usage interne soit pour une redistribution par le client, en tant que partie de
// l'application, au sein des produits du client.
//   Eléments sous licence - Propriété d'IBM
// 5724-J34 © COPYRIGHT International Business Machines Corp. 2007
package com.ibm.websphere.objectgrid.security.plugins.builtins;

import com.ibm.websphere.objectgrid.security.plugins.Credential;

/**
 * Cette classe représente des données d'identification contenant un ID utilisateur et un mot de passe.
 *
 * @ibm-api
 * @since WAS XD 6.0.1
 *
 * @see Credential
 * @see UserPasswordCredentialGenerator#getCredential()
 */
public class UserPasswordCredential implements Credential {

    private static final long serialVersionUID = 140904482554100722 8L;

    private String ivUserName;

    private String ivPassword;

    /**
     * Crée un UserPasswordCredential avec le nom d'utilisateur
     * et le mot de passe spécifiés.
     *
     * @param userName Le nom d'utilisateur pour ces données d'identification
     * @param password Le mot de passe pour ces données d'identification
     *
     * @throws IllegalArgumentException si userName or password sont <code>null</code>
     */
    public UserPasswordCredential(String userName, String password) {
        super();
        if (userName == null || password == null) {
            throw new IllegalArgumentException("User name and password cannot be null.");
        }
```

```
    this.ivUserName = userName;
    this.ivPassword = password;
}

/**
 * Obtention du nom d'utilisateur pour ces données d'identification.
 *
 * @return   l'argument username qui a été passé au constructeur
 *           ou la méthode <code>setUserName(String)</code>
 *           de cette classe
 *
 * @see #setUserName(String)
 */
public String getUserName() {
    return ivUserName;
}

/**
 * Définit le nom d'utilisateur pour ces données d'identification.
 *
 * @param userName Le nom d'utilisateur à définir.
 *
 * @throws IllegalArgumentException si userName est <code>null</code>
 */
public void setUserName(String userName) {
    if (userName == null) {
        throw new IllegalArgumentException("User name cannot be null.");
    }
    this.ivUserName = userName;
}

/**
 * Obtention du mot de passe pour ces données d'identification.
 *
 * @return   l'argument password qui a été passé au constructeur
 *           ou la méthode <code>setPassword(String)</code>
 *           de cette classe
 *
 * @see #setPassword(String)
 */
public String getPassword() {
    return ivPassword;
}

/**
 * Définit le mot de passe pour ces données d'identification
 *
 * @param password Le mot de passe à définir.
 *
 * @throws IllegalArgumentException si password est <code>null</code>
 */
public void setPassword(String password) {
    if (password == null) {
        throw new IllegalArgumentException("Password cannot be null.");
    }
    this.ivPassword = password;
}

/**
 * Vérifie l'égalité des deux objets UserPasswordCredential.
 * <p>
 * Deux objets UserPasswordCredential sont égaux si et seulement si leurs noms d'utilisateur
 * et leurs mots de passe sont égaux.
 *
 * @param o l'objet dont nous testons l'égalité avec cet objet.
 *
 * @return <code>true</code> si les deux objets UserPasswordCredential sont équivalents.
 *
 * @see Credential#equals(Object)
 */
public boolean equals (Object o)  {
    if (this == o) {
        return true;
    }
    if (o instanceof UserPasswordCredential) {
        UserPasswordCredential other = (UserPasswordCredential) o;
        return other.ivPassword.equals(ivPassword) && other.ivUserName.equals(ivUserName);
    }

    return false;
}

/**
 * Retourne le code de hachage de l'objet UserPasswordCredential.
 *
 * @return le code de hachage de cet objet
 *
 * @see Credential#hashCode()
 */
```

```
    public int hashCode ()  {
        return ivUserName.hashCode() + ivPassword.hashCode();
    }
}
```

UserPasswordCredentialGenerator.java

```
// Cet exemple de programme est fourni TEL QUEL et peut être utilisé, exécuté, copié et modifié
// gratuitement par le client
// (a) à des fins d'études,
// (b) afin de développer des applications conçues pour être exécutées avec un produit IBM WebSphere,
// soit pour un usage interne soit pour une redistribution par le client, en tant que partie de
// l'application, au sein des produits du client.
//   Eléments sous licence - Propriété d'IBM
// 5724-J34 © COPYRIGHT International Business Machines Corp. 2007
package com.ibm.websphere.objectgrid.security.plugins.builtins;

import java.util.StringTokenizer;

import com.ibm.websphere.objectgrid.security.plugins.Credential;
import com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator;

/**
 * Ce générateur de données d'identification crée des objets <code>UserPasswordCredential</code>.
 * <p>
 * UserPasswordCredentialGenerator a une relation un à un
 * avec UserPasswordCredential car il ne peut créer qu'un UserPasswordCredential
 * représentant une seule identité.
 *
 * @since WAS XD 6.0.1
 * @ibm-api
 *
 * @see CredentialGenerator
 * @see UserPasswordCredential
 */
public class UserPasswordCredentialGenerator implements CredentialGenerator {

    private String ivUser;

    private String ivPwd;

    /**
     * Crée un UserPasswordCredentialGenerator sans nom d'utilisateur ni mot de passe.
     *
     * @see #setProperties(String)
     */
    public UserPasswordCredentialGenerator() {
        super();
    }

    /**
     * Crée un UserPasswordCredentialGenerator avec un nom d'utilisateur et un mot de passe
     * spécifiés
     *
     * @param user the user name
     * @param pwd the password
     */
    public UserPasswordCredentialGenerator(String user, String pwd) {
        ivUser = user;
        ivPwd = pwd;
    }

    /**
     * Crée un nouvel objet <code>UserPasswordCredential</code>
     * avec le nom d'utilisateur et le mot de passe de cet objet.
     *
     * @return une nouvelle instance d'<code>UserPasswordCredential</code>
     *
     * @see CredentialGenerator#getCredential()
     * @see UserPasswordCredential
     */
    public Credential getCredential() {
        return new UserPasswordCredential(ivUser, ivPwd);
    }

    /**
     * Obtient le mot de passe pour ce générateur de données d'identification.
     *
     * @return   l'argument password qui a été passé au constructeur
     */
    public String getPassword() {
        return ivPwd;
    }

    /**
     * Obtient le nom d'utilisateur pour ces données d'identification
     *
     * @return   l'argument user qui a été passé au constructeur
     *           de cette classe
     */
    public String getUserName() {
        return ivUser;
    }
```

```
    /**
     * Définit des propriétés supplémentaires, un nom d'utilisateur et un mot de passe
     *
     * @param properties chaîne properties avec un nom d'utilisateur
     *                   et un mot de passe séparés par un espace.
     *
     * @throws IllegalArgumentException si le format n'est pas valide
     */
    public void setProperties(String properties) {
        StringTokenizer token = new StringTokenizer(properties, " ");
        if (token.countTokens() != 2) {
            throw new IllegalArgumentException(
                "Les propriétés doivent comporter un nom d'utilisateur et un mot de passe
séparés par un espace.");
        }

        ivUser = token.nextToken();
        ivPwd = token.nextToken();
    }
    /**
     * Vérifie l'égalité des deux objets UserPasswordCredentialGenerator.
     * <p>
     * Deux objets UserPasswordCredentialGenerator sont égaux si et seulement si leurs noms d'utilisateur
     * et leurs mots de passe sont égaux.
     *
     * @param obj the object we are testing for equality with this object.
     *
     * @return <code>true</code> if both UserPasswordCredentialGenerator objects
     *         are equivalent.
     */
    public boolean equals(Object obj) {
        if (obj == this) {
            return true;
        }

        if (obj != null && obj instanceof UserPasswordCredentialGenerator) {
            UserPasswordCredentialGenerator other = (UserPasswordCredentialGenerator) obj;

            boolean bothUserNull = false;
            boolean bothPwdNull = false;

            if (ivUser == null) {
                if (other.ivUser == null) {
                    bothUserNull = true;
                } else {
                    return false;
                }
            }

            if (ivPwd == null) {
                if (other.ivPwd == null) {
                    bothPwdNull = true;
                } else {
                    return false;
                }
            }

            return (bothUserNull || ivUser.equals(other.ivUser)) && (bothPwdNull || ivPwd.equals(other.ivPwd));
        }

        return false;
    }

    /**
     * Returns the hashcode of the UserPasswordCredentialGenerator object.
     *
     * @return the hash code of this object
     */
    public int hashCode ()  {

        return ivUser.hashCode() + ivPwd.hashCode();
    }

}
```

La classe UserPasswordCredential contient deux attributs: userName et password. UserPasswordCredentialGenerator sert de fabrique contenant les objets UserPasswordCredential.

WSTokenCredential et WSTokenCredentialGenerator

Lorsque les clients WebSphere eXtreme Scale sont tous déployés dans WebSphere Application Server, l'application client peut utiliser ces deux implémentations pré-intégrées pour peu que les conditions suivantes soient réunies :

1. La sécurité globale de WebSphere Application Server est activée.
2. Tous les clients et les serveurs WebSphere eXtreme Scale s'exécutent sur des machines virtuelles Java WebSphere Application Server.
3. Les serveurs d'applications se trouvent tous dans le même domaine de sécurité.
4. Le client est déjà authentifiée dans WebSphere Application Server.

Dans ce cas de figure, le client peut utiliser la classe com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator pour générer des données d'identification. Le serveur utilise une classe d'implémentation WSAuthenticator pour authentifier ces données.

Ce scénario exploite le fait que le client eXtreme Scale a déjà été authentifié. Du fait que les serveurs d'applications sur lesquels sont les serveurs se trouvent dans le même domaine de sécurité que ceux qui hébergent les clients, les jetons de sécurité peuvent être propagés du client vers le serveur de sorte que le même registre d'utilisateurs n'a pas besoin d'être authentifié à nouveau.

Remarque : Ne partez pas du principe qu'un CredentialGenerator génère toujours les mêmes données d'identification. Dans le cas de données pouvant expirer et réactualisables, le CredentialGenerator doit être capable de générer les données d'identification valides les plus récentes pour garantir l'authentification. L'utilisation de tickets Kerberos comme objet Credential en est un bon exemple. Lorsque le ticket Kerberos s'actualise, le CredentialGenerator doit extraire le ticket actualisé lorsque la méthode CredentialGenerator.getCredential est appelée.

Plug-in Authenticator

Après que le client eXtreme Scale a récupéré l'objet Credential par le biais de l'objet CredentialGenerator, cet objet Credential est envoyé en même temps de la requête client au serveur eXtreme Scale. Le serveur authentifie l'objet Credential avant de traiter la demande. Si l'objet Credential est authentifié, un objet Subject est renvoyé pour représenter ce client.

Cet objet Subject est alors mis en cache et expire lorsque de délai d'expiration de la session est atteint. Ce délai peut être défini par le biais de la propriété loginSessionExpirationTime dans le fichier XML du cluster. Par exemple, le paramètre `loginSessionExpirationTime="300"` entraîne l'expiration de l'objet Subject dans 300 secondes.

Cet objet Subject est alors utilisé pour autorisé la requête, ce qui sera illustré plus loin. Un serveur eXtreme Scale utilise le plug-in Authenticator pour authentifier l'objet Credential. Pour plus de détails, voir les explications concernant Authenticator dans la documentation de l'API.

C'est dans le plug-in Authenticator que l'environnement d'exécution d'eXtreme Scale authentifie l'objet Credential issu du registre des utilisateurs clients, un serveur LDAP, par exemple.

WebSphere eXtreme Scale ne fournit pas de configuration de registre d'utilisateurs immédiatement utilisable. Le soin de configurer et de gérer ce registre a volontairement été laissé en dehors de WebSphere eXtreme Scale pour des raisons de simplicité et de flexibilité. Ce plug-in implémente la connexion au registre des utilisateurs et l'authentification auprès de ce registre. Par exemple, une implémentation d'Authenticator extraira des données d'identification l'ID utilisateur et le mot de passe, les utilisera pour la connexion et la validation auprès

d'un serveur LDAP et créera un objet Subject résultant de l'authentification. L'implémentation peut utiliser des modules de connexion JAAS. Un objet Subject résultant de l'authentification est renvoyé.

Vous remarquerez que cette méthode crée deux exceptions : InvalidCredentialException et ExpiredCredentialException. L'exception InvalidCredentialException indique que les données d'identification ne sont pas valides. L'exception ExpiredCredentialException signale que les données d'identification ont expiré. Si la méthode authenticate donne lieu à l'une de ces deux exceptions, ces exceptions sont renvoyées au client. Mais l'environnement d'exécution du client gère différemment ces deux exceptions :

- S'il s'agit d'une exception InvalidCredentialException, l'environnement d'exécution du client affiche cette exception. Votre application doit gérer l'exception. Vous pouvez corriger le CredentialGenerator, par exemple, et retenter l'opération.
- Si l'erreur est une exception ExpiredCredentialException et que le nombre des nouvelles tentatives est différent de 0, l'environnement d'exécution du client appelle à nouveau la méthode CredentialGenerator.getCredential et envoie au serveur le nouvel objet Credential. Si l'authentification des nouvelles données d'identification réussit, le serveur traite la demande. Si elle échoue, l'exception est renvoyée au client. Si le nombre des tentatives d'authentification atteint la valeur définie et que le client continue à recevoir une exception ExpiredCredentialException, il résulte une exception ExpiredCredentialException exception. Votre application doit gérer l'erreur.

L'interface Authenticator apporte une grande flexibilité. Vous pouvez implémenter cette interface de la manière qui vous est propre. Vous pouvez, par exemple, l'implémenter pour qu'elle prenne en charge deux registres d'utilisateurs différents.

WebSphere eXtreme Scale fournit des exemples d'implémentations du plug-in Authenticator. Hormis celle du plug-in Authenticator de WebSphere Application Server, les autres implémentations ne sont que des exemples fournies à des fins de tests.

KeyStoreLoginAuthenticator

Cet exemple utilise une implémentation pré-intégrée dans eXtreme Scale : KeyStoreLoginAuthenticator, qui n'est là qu'à des fins de test et qu'à titre d'exemple (un fichier de clés est un registre simple d'utilisateurs, qui ne doit pas être utilisé dans le cadre d'un environnement de production). Là aussi, la classe est affichée pour montrer comment implémenter un authentificateur.

KeyStoreLoginAuthenticator.java

```
// Cet exemple de programme est fourni TEL QUEL et peut être utilisé, exécuté, copié et modifié
// gratuitement par le client
// (a) à des fins d'études,
// (b) afin de développer des applications conçues pour être exécutées avec un produit IBM WebSphere,
// soit pour un usage interne soit pour une redistribution par le client, en tant que partie de
// l'application, au sein des produits du client.
//   Eléments sous licence - Propriété d'IBM
// 5724-J34 © COPYRIGHT International Business Machines Corp. 2007

package com.ibm.websphere.objectgrid.security.plugins.builtins;

import javax.security.auth.Subject;
import javax.security.auth.login.LoginContext;
import javax.security.auth.login.LoginException;

import com.ibm.websphere.objectgrid.security.plugins.Authenticator;
import com.ibm.websphere.objectgrid.security.plugins.Credential;
import com.ibm.websphere.objectgrid.security.plugins.ExpiredCredentialException;
import com.ibm.websphere.objectgrid.security.plugins.InvalidCredentialException;
import com.ibm.ws.objectgrid.Constants;
import com.ibm.ws.objectgrid.ObjectGridManagerImpl;
import com.ibm.ws.objectgrid.security.auth.callback.UserPasswordCallbackHandlerImpl;
```

```
/**
 * Cette classe est une implémentation de l'interface <code>Authenticator</code>
 * lorsqu'un nom d'utilisateur et un mot de passe sont utilisés comme données d'identification.
 * <p>
 * Lorsqu'on utilise l'authentification par ID utilisateur et mot de passe, les données d'identification
 * passées à la méthode <code>authenticate(Credential)</code> est un objet
 * UserPasswordCredential.
 * <p>
 * Cette implémentation va utiliser un <code>KeyStoreLoginModule</code> pour authentifier
 * l'utilisateur dans le magasin de clés à l'aide d'un module de connexion JAAS "KeyStoreLogin".
 * Le magasin de clés peut être configuré en option de la classe
 * <code>KeyStoreLoginModule</code>
 * Voir la classe <code>KeyStoreLoginModule</code> pour plus de détails
 * sur la manière de constituer le fichier de configuration de connexion JAAS.
 * <p>
 * Cette classe n'est là qu'à titre d'exemple et à des fins de tests rapides. Les utilisateurs doivent
 * écrire leur propre implémentation d'Authenticator qui correspondra mieux
 * à leur environnement.
 *
 * @ibm-api
 * @since WAS XD 6.0.1
 *
 * @see Authenticator
 * @see KeyStoreLoginModule
 * @see UserPasswordCredential
 */
public class KeyStoreLoginAuthenticator implements Authenticator {

    /**
     * Crée un nouveau KeyStoreLoginAuthenticator.
     */
    public KeyStoreLoginAuthenticator() {
        super();
    }

    /**
     * Authentifie un <code>UserPasswordCredential</code>.
     * <p>
     * Utilise le nom d'utilisateur et le mot de passe de l'UserPasswordCredential spécifié
     * pou se connecter au KeyStoreLoginModule nommé "KeyStoreLogin".
     *
     * @throws InvalidCredentialException si les données d'identification ne sont pas
     *         UserPasswordCredential ou si une erreur se produit pendant le traitement
     *         de l'UserPasswordCredential fourni
     *
     * @throws ExpiredCredentialException si les données d'identification ont expiré. Cette exception
     *         n'est pas utilisée par cette implémentation
     *
     * @see Authenticator#authenticate(Credential)
     * @see KeyStoreLoginModule
     */
    public Subject authenticate(Credential credential) throws InvalidCredentialException,
   ExpiredCredentialException {

        if (credential == null) {
            throw new InvalidCredentialException("Supplied credential is null");
        }

        if ( ! (credential instanceof UserPasswordCredential) ) {
            throw new InvalidCredentialException("Supplied credential is not a UserPasswordCredential");
        }

        UserPasswordCredential cred = (UserPasswordCredential) credential;
        LoginContext lc = null;
        try {
            lc = new LoginContext("KeyStoreLogin",
                    new UserPasswordCallbackHandlerImpl(cred.getUserName(), cred.getPassword().toCharArray()));

            lc.login();

            Subject subject = lc.getSubject();

            return subject;
        }
        catch (LoginException le) {
            throw new InvalidCredentialException(le);
        }
        catch (IllegalArgumentException ile) {
            throw new InvalidCredentialException(ile);
        }
    }
}
```

KeyStoreLoginModule.java

```
// Cet exemple de programme est fourni TEL QUEL et peut être utilisé, exécuté, copié et modifié
// gratuitement par le client
// (a) à des fins d'études,
// (b) afin de développer des applications conçues pour être exécutées avec un produit IBM WebSphere,
// soit pour un usage interne soit pour une redistribution par le client, en tant que partie de
// l'application, au sein des produits du client.
//   Eléments sous licence - Propriété d'IBM
```

```
// 5724-J34 © COPYRIGHT International Business Machines Corp. 2007
package com.ibm.websphere.objectgrid.security.plugins.builtins;

import java.io.File;
import java.io.FileInputStream;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.UnrecoverableKeyException;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

import javax.security.auth.Subject;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.NameCallback;
import javax.security.auth.callback.PasswordCallback;
import javax.security.auth.login.LoginException;
import javax.security.auth.spi.LoginModule;
import javax.security.auth.x500.X500Principal;
import javax.security.auth.x500.X500PrivateCredential;

import com.ibm.websphere.objectgrid.ObjectGridRuntimeException;
import com.ibm.ws.objectgrid.Constants;
import com.ibm.ws.objectgrid.ObjectGridManagerImpl;
import com.ibm.ws.objectgrid.util.ObjectGridUtil;

/**
 * Un KeyStoreLoginModule est un module de connexion d'authentification JAAS
 * auprès d'un magasin de clés.
 * <p>
 * Une configuration de connexion doit fournir une option "<code>keyStoreFile</code>"
 * pour indiquer où se trouve le fichier de clés. Si la valeur <code>keyStoreFile</code>
 * contient une propriété système de la forme <code>${system.property}</code>,
 * il sera étendu à la valeur de cette propriété système.
 * <p>
 * S'il n'est pas fourni d'option "<code>keyStoreFile</code>", le nom par défaut du fichier de clés
 * est <code>"${java.home}${/}.keystore"</code>.
 * <p>
 * Voici un exemple de configuration du module de connexion :
 * <pre><code>
 *    KeyStoreLogin {
 *        com.ibm.websphere.objectgrid.security.plugins.builtins.KeystoreLoginModule required
 *            keyStoreFile="${user.dir}${/}security${/}.keystore";
 *    };
 * </code></pre>
 *
 * @ibm-api
 * @since WAS XD 6.0.1
 *
 * @see LoginModule
 */
public class KeyStoreLoginModule implements LoginModule {

    private static final String CLASS_NAME = KeyStoreLoginModule.class.getName();

    /**
     * Nom de la propriété du fichier de clés
     */
    public static final String KEY_STORE_FILE_PROPERTY_NAME = "keyStoreFile";

    /**
     * Type du fichiers de clés. Seul JKS est pris en charge
     */
    public static final String KEYSTORE_TYPE = "JKS";

    /**
     * Le nom par défaut du fichier de clés
     */
    public static final String DEFAULT_KEY_STORE_FILE = "${java.home}${/}.keystore";

    private CallbackHandler handler;

    private Subject subject;

    private boolean debug = false;

    private Set principals = new HashSet();

    private Set publicCreds = new HashSet();

    private Set privateCreds = new HashSet();

    protected KeyStore keyStore;
```

```
/**
 * Crée un nouveau KeyStoreLoginModule.
 */
public KeyStoreLoginModule() {
}

/**
 * Initialise le module de connexion.
 *
 * @see LoginModule#initialize(Subject, CallbackHandler, Map, Map)
 */
public void initialize(Subject sub, CallbackHandler callbackHandler,
        Map mapSharedState, Map mapOptions) {

    // initialisation de toutes les options configurées
    debug = "true".equalsIgnoreCase((String) mapOptions.get("debug"));

    if (sub == null)
        throw new IllegalArgumentException("Subject is not specified");

    if (callbackHandler == null)
        throw new IllegalArgumentException(
        "CallbackHander is not specified");

    // Obtention du chemin du magasin de clés
    String sKeyStorePath = (String) mapOptions
        .get(KEY_STORE_FILE_PROPERTY_NAME);

    // S'il n'y a pas de chemin du magasin de clés, le chemin par défaut est le fichier .keystore
    // dans le répertoire de base de java
    if (sKeyStorePath == null) {
        sKeyStorePath = DEFAULT_KEY_STORE_FILE;
    }

    // Replace the system enviroment variable
    sKeyStorePath = ObjectGridUtil.replaceVar(sKeyStorePath);

    File fileKeyStore = new File(sKeyStorePath);

    try {
        KeyStore store = KeyStore.getInstance("JKS");
        store.load(new FileInputStream(fileKeyStore), null);

        // Enregistrement du magasin de clés
        keyStore = store;

        if (debug) {
            System.out.println("[KeyStoreLoginModule] initialize: Successfully loaded key store");
        }
    }
    catch (Exception e) {
        ObjectGridRuntimeException re = new ObjectGridRuntimeException(
                "Failed to load keystore: " + fileKeyStore.getAbsolutePath());
        re.initCause(e);
        if (debug) {
            System.out.println("[KeyStoreLoginModule] initialize: Key store loading failed with exception "
                    + e.getMessage());
        }
    }

    this.subject = sub;
    this.handler = callbackHandler;
}

/**
 * Authentification d'un utilisateur à partir du fichier de clés.
 *
 * @see LoginModule#login()
 */
public boolean login() throws LoginException {

    if (debug) {
        System.out.println("[KeyStoreLoginModule] login: entry");
    }

    String name = null;
    char pwd[] = null;

    if (keyStore == null || subject == null || handler == null) {
        throw new LoginException("Module initialization failed");
    }

    NameCallback nameCallback = new NameCallback("Username:");
    PasswordCallback pwdCallback = new PasswordCallback("Password:", false);

    try {
        handler.handle(new Callback[] { nameCallback, pwdCallback });
    }
    catch (Exception e) {
        throw new LoginException("Callback failed: " + e);
```

```
    }

    name = nameCallback.getName();
    char[] tempPwd = pwdCallback.getPassword();

    if (tempPwd == null) {
        // treat a NULL password as an empty password
        tempPwd = new char[0];
    }
    pwd = new char[tempPwd.length];
    System.arraycopy(tempPwd, 0, pwd, 0, tempPwd.length);

    pwdCallback.clearPassword();

    if (debug) {
        System.out.println("[KeyStoreLoginModule] login: "
                + "user entered user name: " + name);
    }

    // Validation du nom d'utilisateur et du mot de passe
    try {
        validate(name, pwd);
    }
    catch (SecurityException se) {
        principals.clear();
        publicCreds.clear();
        privateCreds.clear();
        LoginException le = new LoginException(
        "Exception encountered during login");
        le.initCause(se);

        throw le;
    }

    if (debug) {
        System.out.println("[KeyStoreLoginModule] login: exit");
    }
    return true;
}

/**
 * Indique si l'utilisateur est acceptê.
 * <p>
 * Cette méthode n'est appelée que si l'utilisateur est authentifié par tous les modules
 * du fichier de configuration de connexion. Les objets principaux seront ajoutés
 * au subject stockê.
 *
 * @return false si pour une raison ou pour une autre les principaux ne peuvent être ajoutés; true
 *         autrement
 *
 * @exception LoginException
 *                Une LoginException est levée si le est en lecture seule
 *                ou si des exceptions irrécupérables sont rencontrées.
 *
 * @see LoginModule#commit()
 */
public boolean commit() throws LoginException {
    if (debug) {
        System.out.println("[KeyStoreLoginModule] commit: entry");
    }

    if (principals.isEmpty()) {
        throw new IllegalStateException("Commit is called out of sequence");
    }

    if (subject.isReadOnly()) {
        throw new LoginException("Subject is Readonly");
    }

    subject.getPrincipals().addAll(principals);
    subject.getPublicCredentials().addAll(publicCreds);
    subject.getPrivateCredentials().addAll(privateCreds);

    principals.clear();
    publicCreds.clear();
    privateCreds.clear();

    if (debug) {
        System.out.println("[KeyStoreLoginModule] commit: exit");
    }
    return true;
}

/**
 * Indique que l'utilisateur n'est pas accepté
 *
 * @see LoginModule#abort()
 */
public boolean abort() throws LoginException {
    boolean b = logout();
    return b;
```

```
}

/**
 * Déconnecte l'utilisateur. Efface toutes les mappes.
 *
 * @see LoginModule#logout()
 */
public boolean logout() throws LoginException {


    // Effacement des variables d'instance
    principals.clear();
    publicCreds.clear();
    privateCreds.clear();

    // Efacement des mappes dans le subject
    if (!subject.isReadOnly()) {
        if (subject.getPrincipals() != null) {
            subject.getPrincipals().clear();
        }

        if (subject.getPublicCredentials() != null) {
            subject.getPublicCredentials().clear();
        }

        if (subject.getPrivateCredentials() != null) {
            subject.getPrivateCredentials().clear();
        }
    }
    return true;
}

/**
 * Validation du nom d'utilisateur et du mot de passe d'après le magasin de clés.
 *
 * @param userName user name
 * @param password password
 * @throws SecurityException if any exceptions encountered
 */
private void validate(String userName, char password[])
    throws SecurityException {

    PrivateKey privateKey = null;

    // Obtention de la clé privée depuis le magasin de clés
    try {
        privateKey = (PrivateKey) keyStore.getKey(userName, password);
    }
    catch (NoSuchAlgorithmException nsae) {
        SecurityException se = new SecurityException();
        se.initCause(nsae);
        throw se;
    }
    catch (KeyStoreException kse) {
        SecurityException se = new SecurityException();
        se.initCause(kse);
        throw se;
    }
    catch (UnrecoverableKeyException uke) {
        SecurityException se = new SecurityException();
        se.initCause(uke);
        throw se;
    }

    if (privateKey == null) {
        throw new SecurityException("Invalid name: " + userName);
    }

    // Vérification des certificats
    Certificate certs[] = null;
    try {
        certs = keyStore.getCertificateChain(userName);
    }
    catch (KeyStoreException kse) {
        SecurityException se = new SecurityException();
        se.initCause(kse);
        throw se;
    }

    if (debug) {
        System.out.println("  Print out the certificates:");
        for (int i = 0; i < certs.length; i++) {
            System.out.println("  certificate " + i);
            System.out.println("    " + certs[i]);
        }
    }

    if (certs != null && certs.length > 0) {

        // Si le premier certificat est un X509Certificate
        if (certs[0] instanceof X509Certificate) {
```

```
                try {
                    // Obtention du premier certificat qui représente l'utilisateur
                    X509Certificate certX509 = (X509Certificate) certs[0];

                    // Création d'un principal
                    X500Principal principal = new X500Principal(certX509
                            .getIssuerDN()
                            .getName());
                    principals.add(principal);

                    if (debug) {
                        System.out.println("  Principal added: " + principal);
                    }
                    // Création de l'objet de chemin de certification et ajout de cet objet
                    // à l'ensemble des données d'identification publiques
                    CertificateFactory factory = CertificateFactory
                        .getInstance("X.509");
                    java.security.cert.CertPath certPath = factory
                        .generateCertPath(Arrays.asList(certs));
                    publicCreds.add(certPath);

                    // Ajout des données d'identification privées au jeu de données d'identification privées
                    privateCreds.add(new X500PrivateCredential(certX509,
                            privateKey, userName));

                }
                catch (CertificateException ce) {
                    SecurityException se = new SecurityException();
                    se.initCause(ce);
                    throw se;
                }
            }
            else {
                // Le premier certificat n'est pas un X509Certificate
                // Nous n'ajoutons le certificat qu'au jeu des données d'identification publiques
                // et la clé privée au jeu de données d'identification privées.
                publicCreds.add(certs[0]);
                privateCreds.add(privateKey);
            }
        }
    }
}
```

Utiliser le plug-in d'authentificateur pour LDAP

L'implémentation de com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPAuthenticator est fournie par défaut pour permettre de gérer l'authentification par nom d'utilisateur et mot de passe auprès d'un serveur LDAP. Cette implémentation utilise le module de connexion LDAPLogin pour permettre à l'utilisateur d'ouvrir une session sur un serveur LDAP. Le fragment de code suivant montre comment est implémentée la méthode authenticate :

```
/**
* @see com.ibm.ws.objectgrid.security.plugins.Authenticator#
* authenticate(LDAPLogin)
*/
public Subject authenticate(Credential credential) throws
InvalidCredentialException, ExpiredCredentialException {

    UserPasswordCredential cred = (UserPasswordCredential) credential;
    LoginContext lc = null;
    try {
        lc = new LoginContext("LDAPLogin",
            new UserPasswordCallbackHandlerImpl(cred.getUserName(),
            cred.getPassword().toCharArray()));

        lc.login();

        Subject subject = lc.getSubject();

        return subject;
    }
    catch (LoginException le) {
        throw new InvalidCredentialException(le);
    }
    catch (IllegalArgumentException ile) {
        throw new InvalidCredentialException(ile);
    }
}
```

Par ailleurs, eXtreme Scale est livré à cette fin avec le module de connexion com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPLoginModule. Vous devez fournir les deux options suivantes dans le fichier de configuration des connexions JAAS.

- providerURL : l'URL du fournisseur du serveur LDAP
- factoryClass : la classe d'implémentation de fabrique de contexte LDAP

Le module LDAPLoginModule appelle la méthode com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPAuthenticationHelper.authenticate. Le fragment de code suivant indique comment implémenter la méthode authenticate de LDAPAuthenticationHelper.

```
/**
* Authenticate the user to the LDAP directory.
* @param user the user ID, e.g., uid=xxxxxx,c=us,ou=bluepages,o=ibm.com
* @param pwd the password
*
* @throws NamingException
*/
public String[] authenticate(String user, String pwd)
throws NamingException {
    Hashtable env = new Hashtable();
    env.put(Context.INITIAL_CONTEXT_FACTORY, factoryClass);
    env.put(Context.PROVIDER_URL, providerURL);
    env.put(Context.SECURITY_PRINCIPAL, user);
    env.put(Context.SECURITY_CREDENTIALS, pwd);
    env.put(Context.SECURITY_AUTHENTICATION, "simple");

    InitialContext initialContext = new InitialContext(env);

    // Look up for the user
    DirContext dirCtx = (DirContext) initialContext.lookup(user);

    String uid = null;
    int iComma = user.indexOf(",");
    int iEqual = user.indexOf("=");
    if (iComma > 0 && iComma > 0) {
        uid = user.substring(iEqual + 1, iComma);
    }
    else {
        uid = user;
    }

    Attributes attributes = dirCtx.getAttributes("");

    // Check the UID
    String thisUID = (String) (attributes.get(UID).get());

    String thisDept = (String) (attributes.get(HR_DEPT).get());

    if (thisUID.equals(uid)) {
        return new String[] { thisUID, thisDept };
    }
    else {
        return null;
    }
}
```

Si l'authentification réussit, l'ID et le mot de passe sont considérés comme valides. Puis le module de connexion obtient de cette méthode authenticate l'ID et le département. Le module de connexion crée deux principaux : SimpleUserPrincipal et SimpleDeptPrincipal. Vous pouvez utiliser le Subject authentifié pour l'autorisation de groupe (dans notre cas, le département est un groupe) et pour l'autorisation individuelle.

L'exemple qui suit montre une configuration de module de connexion utilisée pour ouvrir une session sur le serveur LDAP :

```
LDAPLogin { com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPLoginModule required
    providerURL="ldap://directory.acme.com:389/"
    factoryClass="com.sun.jndi.ldap.LdapCtxFactory";
};
```

Dans cette configuration, le serveur LDAP pointe sur `ldap://directory.acme.com:389/server`. Vous utiliserez ici l'URL de votre serveur LDAP. Ce module de connexion utilise l'ID et le mot de passe fournis pour se connecter au serveur LDAP. Cette implémentation n'est là qu'à des fins de test.

Utiliser le plug-in d'authentificateur pour WebSphere Application Server

eXtreme Scale intègre également l'implémentation de com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator qui permet d'utiliser l'infrastructure de sécurité de WebSphere Application Server. Cette implémentation pré-intégrée est utilisable lorsque les conditions suivantes sont réunies.

1. La sécurité globale de WebSphere Application Server est activée.
2. Tous les clients et les serveurs eXtreme Scale sont lancés sur des machines virtuelles Java WebSphere Application Server.
3. Ces serveurs d'applications se trouvent tous dans le même domaine de sécurité.
4. Le client eXtreme Scale est déjà authentifié dans WebSphere Application Server.

Le client peut utiliser la classe com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator pour générer des données d'identification. Le serveur utilise cette classe d'implémentation Authenticator pour authentifier ces données. Si la clé est authentifiée, un objet Subject est renvoyé.

Ce scénario exploite le fait que le client a déjà été authentifié. Du fait que les serveurs d'applications sur lesquels sont les serveurs se trouvent dans le même domaine de sécurité que ceux qui hébergent les clients, les jetons de sécurité peuvent être propagés du client vers le serveur de sorte que le même registre d'utilisateurs n'a pas besoin d'être authentifié à nouveau.

Utiliser le plug-in d'authentificateur Tivoli Access Manager plug-in

Tivoli Access Manager connaît une large utilisation comme serveur de sécurité. Vous pouvez également implémenter Authenticator à l'aide des modules de connexion fournis par Tivoli Access Manager.

Pour authentifier un utilisateur pour Tivoli Access Manager, appliquez le module de connexion com.tivoli.mts.PDLoginModule, lequel requiert que l'application appelante fournisse les informations suivantes :

1. un nom de principal, spécifié soit sous la forme d'un nom court, soit d'un nom X.500 (DN)
2. un mot de passe

Le module de connexion authentifie le principal et retourne les données d'identification Tivoli Access Manager. Le module de connexion attend de l'application appelante les informations suivantes :

1. le nom d'utilisateur, via un objet javax.security.auth.callback.NameCallback
2. le mot de passe, via un objet javax.security.auth.callback.PasswordCallback

Lorsque les données d'identification Tivoli Access Manager ont pu être récupérées, le JAAS LoginModule crée un Subject et un PDPrincipal. Aucune intégration de l'authentification pour Tivoli Access Manager n'est fournie car celle-ci l'est avec le module PDLoginModule. Pour plus de détails, voir le manuel IBM Tivoli Access Manager Authorization Java Classes Developer Reference.

Se connecter de manière sécurisée à WebSphere eXtreme Scale

Pour connecter de manière sécurisée un client eXtreme Scale à un serveur, vous pouvez utiliser n'importe quelle méthode connect de l'interface ObjectGridManager qui accepte un objet ClientSecurityConfiguration. Voici un exemple succinct.

```
public ClientClusterContext connect(String catalogServerAddresses,
   ClientSecurityConfiguration securityProps,
     URL overRideObjectGridXml) throws ConnectException;
```

Cette méthode reçoit un paramètre du type ClientSecurityConfiguration, qui est une interface représentant une configuration de la sécurité d'un client. Vous pouvez utiliser l'API publique com.ibm.websphere.objectgrid.security.config.ClientSecurityConfigurationFactory pour créer une instance avec des valeurs par défaut. Vous pouvez également créer une instance en passant le fichier de propriétés du client WebSphere eXtreme Scale. Ce fichier contient les propriétés suivantes qui intéressent l'authentification. La valeur marquée par un signe plus (+) est la valeur par défaut.

- `securityEnabled (true, false+):` Cette propriété indique si la sécurité est activée. Lorsqu'un client se connecte à un serveur, la valeur securityEnable côté client et côté serveur doit être identique des deux côtés : égale à `true` ou à `false` dans les deux cas. Par exemple, si la sécurité du serveur connecté est activée, la valeur de la propriété doit être associée à `true` du côté client pour que le client puisse se connecter au serveur.
- authenticationRetryCount (an integer value, 0+) : Cette propriété détermine le nombre de tentatives de connexion à effectuer lorsque des données d'identification ont expiré. Une valeur de 0 indique qu'aucune nouvelle tentative n'est effectuée. La tentative d'authentification ne s'applique qu'au cas où les données d'identification sont arrivées à expiration. Si les données ne sont pas valides, aucune nouvelle tentative n'est effectuée. C'est à votre application de relancer les tentatives.

Après avoir créé un objet com.ibm.websphere.objectgrid.security.config.ClientSecurityConfiguration, définissez l'objet credentialGenerator sur le client en utilisant la méthode suivante :

```
/**
* Set the {@link CredentialGenerator} object for this client.
* @param generator the CredentialGenerator object associated with this client
*/
void setCredentialGenerator(CredentialGenerator generator);
```

Vous pouvez définir l'objet CredentialGenerator également dans le fichier de propriétés du client WebSphere eXtreme Scale :

- credentialGeneratorClass : Le nom de l'implémentation de classe de l'objet CredentialGenerator. Cette classe doit avoir un constructeur par défaut.
- credentialGeneratorProps : Les propriétés de la classe CredentialGenerator. Si la valeur n'est pas null, elle est définie à l'aide de la méthode setProperties(String) comme l'objet CredentialGenerator qui est construit.

Voici un exemple d'instanciation de ClientSecurityConfiguration, utilisée ensuite pour se connecter au serveur.

```
/**
* Get a secure ClientClusterContext
* @return a secure ClientClusterContext object
*/
protected ClientClusterContext connect() throws ConnectException {
ClientSecurityConfiguration csConfig = ClientSecurityConfigurationFactory
.getClientSecurityConfiguration("/properties/security.ogclient.props");

UserPasswordCredentialGenerator gen= new
UserPasswordCredentialGenerator("manager", "manager1");

csConfig.setCredentialGenerator(gen);

return objectGridManager.connect(csConfig, null);
}
```

Lorsque la méthode connect est appelée, le client WebSphere eXtreme Scale appelle la méthode CredentialGenerator.getCredential pour obtenir les données d'identification du client. Ces données sont envoyées pour authentification au serveur avec la demande de connexion.

Utiliser une instance CredentialGenerator différente par session

Dans certains cas, un client WebSphere eXtreme Scale ne représente qu'une seule identité de client, alors que, dans d'autres, il représentera des identités multiples. Voici un scénario pour ce dernier cas : un client WebSphere eXtreme Scale est créé et partagé sur un serveur Web. Tous les servlets de ce serveur Web utilisent cet unique client WebSphere eXtreme Scale. Chaque servlet représentant un client Web différent, vous devez utiliser des données d'identification différentes lorsque vous envoyez des demandes aux serveurs WebSphere eXtreme Scale.

WebSphere eXtreme Scale permet le changement des données d'identification au niveau session. Chaque session peut en effet utiliser un objet CredentialGenerator différent. En conséquence de quoi, les scénarios précédents peuvent être implémentés en laissant le servlet obtenir une session avec un objet CredentialGenerator différent. L'exemple qui suit illustre la méthode ObjectGrid.getSession(CredentialGenerator) dans l'interface ObjectGridManager.

```
/**
     * Obtention d'une session à l'aide de <code>CredentialGenerator</code>.
     * <p>
     * Cette méthode ne peut être appelée que par le client ObjectGrid
     * dans environnement client/serveur ObjectGrid. Si ObjectGrid est utilisé dans un modèle local, c'est-à-dire
     * au sein de la même JVM sans aucun client ou serveur existant, <code>getSession(Subject)</code>
     * ou le plug-in <code>SubjectSource</code> doivent être utilisés pour sécuriser l'ObjectGrid.
     *
     * <p>Si la méthode <code>initialize()</code> n'a pas été appelée avant
     * le premier appel à <code>getSession</code>, une initialisation implicite
     * va se produire. C'est la garantie que toute la configuration est effectuée
     * avant que toute utilisation d'exécution ne soit requise.</p>
     *
     * @param credGen A <code>CredentialGenerator</code> pour générer des données d'identification
     *                pour la session retournée.
     *
     * @return Une instance de <code>Session</code>
     *
     * @throws ObjectGridException si une erreur se produit durant le traitement
     * @throws TransactionCallbackException si le <code>TransactionCallback</code>
     *         lève une exception
     * @throws IllegalStateException si cette méthode est appelée après
     *         l'appel à la méthode <code>destroy()</code>.
     *
     * @see #destroy()
     * @see #initialize()
     * @see CredentialGenerator
     * @see Session
     * @since WAS XD 6.0.1
 */
Session getSession(CredentialGenerator credGen) throws
ObjectGridException, TransactionCallbackException;
```

Voici un exemple :

```
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();

CredentialGenerator credGenManager = new UserPasswordCredentialGenerator("manager", "xxxxxx");
CredentialGenerator credGenEmployee = new UserPasswordCredentialGenerator("employee", "xxxxxx");

ObjectGrid og = ogManager.getObjectGrid(ctx, "accounting");

// L'on obtient une session avec CredentialGenerator;
Session session = og.getSession(credGenManager );

// L'on obtien la mappe employee
ObjectMap om = session.getMap("employee");

// l'on démarre une transaction.
session.begin();

Object rec1 = map.get("xxxxxx");

session.commit();

// L'on obtient une autre Gsession avec un autre CredentialGenerator;
session = og.getSession(credGenEmployee );

// L'on obtient la mappe employee
om = session.getMap("employee");

// L'on démarre une transaction.
session.begin();

Object rec2 = map.get("xxxxx");

session.commit();
```

Si vous utilisez la méthode ObjectGrid.getSession pour obtenir un objet Session, la session utilisera l'objet CredentialGenerator défini dans l'objet ClientConfigurationSecurity. La méthode ObjectGrid.getSession(CredentialGenerator) remplace le CredentialGenerator défini dans l'objet ClientSecurityConfiguration.

Et les performances ne pourront qu'y gagner si vous pouvez réutiliser l'objet Session. Cela dit, l'appel à la méthode ObjectGrid.getSession(CredentialGenerator) ne coûte guère. Le temps système est essentiellement dû à l'augmentation du temps passé à récupérer de la place. Veillez à bien libérer les références une fois que vous en avez fini avec les objets Session. En général, si votre objet Session peut partager l'identité, essayez de le réutiliser. Sinon, utilisez la méthode ObjectGrid.getSession(CredentialGenerator).

Programmation des autorisations de clients

WebSphere eXtreme Scale prend en charge l'autorisation JAAS (Java Authentication and Authorization Service) livrée prête à l'emploi ainsi que les autorisations personnalisées via l'interface ObjectGridAuthorization.

Le plug-in ObjectGridAuthorization permet d'autoriser ObjectGrid, ObjectMap et JavaMap à accéder de manière personnalisée aux principaux représentés par un objet Subject. La plupart du temps, ce plug-in est implémenté pour extraire les principaux d'un objet Subject et vérifier ensuite si les droits d'accès spécifiés sont ou non accordés à ces principaux.

Les droits d'accès passés à la méthode checkPermission(Subject, Permission) peuvent être de l'un des types suivants :

- MapPermission
- ObjectGridPermission
- ServerMapPermission
- AgentPermission

Pour plus de détails, voir la documentation de l'API ObjectGridAuthorization.

MapPermission

La classe publique com.ibm.websphere.objectgrid.security.MapPermission représente les droits d'accès sur les ressources ObjectGrid, particulièrement les méthodes des interfaces ObjectMap ou JavaMap. WebSphere eXtreme Scale définit les chaînes suivantes de droits d'accès permettant d'accéder aux méthodes d'ObjectMap et de JavaMap :

- **read :** permission de lire les données de la mappe. La constante entière est définie comme `MapPermission.READ`.
- **write :** permission de modifier les données de la mappe. La constante entière est définie comme `MapPermission.WRITE`.
- **insert ::** permission d'insérer les données dans la mappe. La constante entière est définie comme `MapPermission.INSERT`.
- **remove :** permission de supprimer les données de la mappe. La constante entière est définie comme `MapPermission.REMOVE`.
- **invalidate :** permission d'invalider les données de la mappe. La constante entière est définie comme `MapPermission.INVALIDATE`.
- **all :** toutes les permissions ci-dessus (read, write, insert, remote et invalidate). La constante entière est définie comme `MapPermission.ALL`.

Pour plus de détails, voir la documentation de l'API MapPermission.

Vous pouvez construire un objet MapPermission en passant le nom qualifié complet de la mappe ObjectGrid (au format [ObjectGrid_name].[ObjectMap_name]) et la chaîne de droit d'accès ou la valeur de type entier. Une chaîne de droit d'accès peut se présenter sous la forme d'une chaîne des permissions évoquées plus haut (read, insert ou all, par exemple), séparées par des virgules. Une valeur de permission de type entier pourra être n'importe laquelle des constantes entières mentionnées plus haut ou une valeur mathématiques de plusieurs constantes entières, comme MapPermission.READ|MapPermission.WRITE, par exemple.

L'autorisation se produit lors de l'appel d'une méthode ObjectMap ou JavaMap. L'environnement d'exécution d'eXtreme Scale vérifie les différents droits d'accès pour les différentes méthodes. Le refus des droits requis au client génère une exception AccessControlException.

Tableau 13. Liste des méthodes et de la MapPermission requise

Permission	ObjectMap/JavaMap
read	containsKey(Object) booléen
	equals(Object) booléen
	Object get(Object)
	Object get(Object, Serializable)
	List getAll(List)
	List getAll(List keyList, Serializable)
	List getAllForUpdate(List)
	List getAllForUpdate(List, Serializable)
	Object getForUpdate(Object)
	Object getForUpdate(Object, Serializable)
	public Object getNextKey(long)
write	Object put(Object key, Object value)
	void put(Object, Object, Serializable)
	void putAll(Map)
	void putAll(Map, Serializable)
	void update(Object, Object)
	void update(Object, Object, Serializable)
insert	public void insert (Object, Object)
	void insert(Object, Object, Serializable)
remove	Object remove (Object)
	void removeAll(Collection)
	void clear()
invalidate	public void invalidate (Object, boolean)
	void invalidateAll(Collection, boolean)
	void invalidateUsingKeyword(Serializable)
	int setTimeToLive(int)

L'autorisation se base uniquement sur la méthode utilisée et non sur l'action effective de cette méthode. Exemple : une méthode put peut aussi bien insérer qu'actualiser un enregistrement en fonction de l'existence de cet enregistrement. Mais, pour l'autorisation, aucune différence ne sera faite entre l'insertion et l'actualisation.

Un type d'opération peut être réalisé par la combinaison d'autres types. Une actualisation, par exemple, pourra être réalisée par un remove et par un insert. Vous devez envisager ces combinaisons possibles lorsque vous concevez vos règles d'autorisation.

ObjectGridPermission

Un com.ibm.websphere.objectgrid.security.ObjectGridPermission représente les droits d'accès à l'ObjectGrid :

- Query : permission de créer une requête sur des objets ou sur des entités. La constante entière est définie comme ObjectGridPermission.QUERY.

- Dynamic map : permission de créer une mappe dynamique à partir du modèle de mappe. La constante entière est définie comme ObjectGridPermission.DYNAMIC_MAP.

Pour plus de détails, voir la documentation de l'API ObjectGridPermission.

Le tableau suivant récapitule les méthodes et l'ObjectGridPermission requise :

Tableau 14. Liste des méthodes et de l'ObjectGridPermission requise

Action	Méthodes
query	com.ibm.websphere.objectgrid.Session.createObjectQuery(String)
query	com.ibm.websphere.objectgrid.em.EntityManager.createQuery(String)
dynamicmap	com.ibm.websphere.objectgrid.Session.getMap(String)

ServerMapPermission

Une ServerMapPermission représente les droits d'accès sur un ObjectMap hébergé sur un serveur. Le nom du droit d'accès n'est autre que le nom complet de la mappe ObjectGrid. Cette autorisation comporte les actions suivantes :

- replicate : permission de répliquer une mappe de serveur dans le cache local
- dynamicIndex : permission pour un client de créer ou de supprimer un index dynamique sur un serveur

Pour plus de détails, voir la documentation de l'API ServerMapPermission. Le tableau suivant répertorie les méthodes détaillées, qui requièrent différents ServerMapPermission :

Tableau 15. Droits d'accès à un ObjectMap hébergé sur serveur

Action	Méthodes
replicate	com.ibm.websphere.objectgrid.ClientReplicableMap.enableClientReplication(Mode, int[], ReplicationMapListener)
dynamicIndex	com.ibm.websphere.objectgrid.BackingMap.createDynamicIndex(String, boolean, String, DynamicIndexCallback)
dynamicIndex	com.ibm.websphere.objectgrid.BackingMap.removeDynamicIndex(String)

AgentPermission

Un AgentPermission représente les droits d'accès aux agents de grille de données. Le nom du droit d'accès n'est autre que le nom complet de la mappe ObjectGrid et l'action est une chaîne, délimitée par des virgules, constituée par les noms des classes d'implémentation ou des packages d'agents.

Pour plus d'informations, voir la documentation de l'API AgentPermission.

Les méthodes suivantes de la classe com.ibm.websphere.objectgrid.datagrid.AgentManager requièrent AgentPermission.

```
com.ibm.websphere.objectgrid.datagrid.AgentManager#callMapAgent(MapGridAgent, Collection)
com.ibm.websphere.objectgrid.datagrid.AgentManager#callMapAgent(MapGridAgent)
com.ibm.websphere.objectgrid.datagrid.AgentManager#callReduceAgent(ReduceGridAgent, Collection)
com.ibm.websphere.objectgrid.datagrid.AgentManager#callReduceAgent(ReduceGridAgent, Collection)
```

Mécanismes d'autorisation

WebSphere eXtreme Scale prend en charge deux sortes de mécanismes d'autorisation : l'autorisation JAAS (Java Authentication and Authorization Service) et l'autorisation personnalisée. Ces mécanismes s'appliquent à la totalité des autorisations. L'autorisation JAAS augmente les règles Java de sécurité en les

dotant de contrôles d'accès centrés sur l'utilisateur. Les droits d'accès peuvent être accordés non seulement en fonction du code en cours d'exécution, mais également en fonction de qui exécute ce code. L'autorisation JAAS fait partie du SDK version 1.4 et plus récent.

En outre, WebSphere eXtreme Scale prend également en charge l'autorisation personnalisée grâce au plug-in suivant :

- ObjectGridAuthorization : moyen personnalisé d'autoriser l'accès à tous les artefacts.

Si vous ne souhaitez pas utiliser l'autorisation JAAS, vous pouvez implémenter votre propre mécanisme d'autorisation. Le recours à un mécanisme d'autorisation personnalisée permet d'utiliser la base de données des règles, le serveur de règles ou Tivoli Access Manager pour gérer les autorisations.

Il existe deux moyens de configurer le mécanisme d'autorisation :

- Configuration XML

  Vous pouvez utiliser le fichier XML ObjectGrid pour définir un ObjectGrid et pour choisir le mécanisme d'autorisation : AUTHORIZATION_MECHANISM_JAAS ou AUTHORIZATION_MECHANISM_CUSTOM. Voici le fichier secure-objectgrid-definition.xml qui est utilisé l'exemple d'application d'entreprise ObjectGridSample :

  ```
  <objectGrids>
   <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
    authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS">
    <bean id="TransactionCallback"
  classname="com.ibm.websphere.samples.objectgrid.HeapTransactionCallback" />
  ...
  </objectGrids>
  ```

- Configuration par programmation

  Si vous souhaitez créer un ObjectGrid à l'aide de la méthode ObjectGrid.setAuthorizationMechanism(int), vous pouvez appeler la méthode suivante pour définir le mécanisme d'autorisation. L'appel de cette méthode s'applique uniquement au modèle de programmation WebSphere eXtreme Scale local lorsque vous instanciez directement l'instance ObjectGrid :

  ```
  /**
   * Set the authorization Mechanism. The default is
   * com.ibm.websphere.objectgrid.security.SecurityConstants.
   * AUTHORIZATION_MECHANISM_JAAS.
   * @param authMechanism the map authorization mechanism
  */
  void setAuthorizationMechanism(int authMechanism);
  ```

Autorisation JAAS

Un objet javax.security.auth.Subject représente un utilisateur authentifié. Un Subject se compose d'un ensemble de principaux, dont chacun représente une identité pour cet utilisateur. Exemple : un Subject peut avoir un principal name (Jean Dupont, par exemple) et un principal group (manager, par exemple).

L'autorisation JAAS permet d'accorder des droits d'accès à des principaux spécifiques. WebSphere eXtreme Scale associe le Subject au contexte actuel de contrôle d'accès. A chaque appel aux méthodes ObjectMap ou Javamap, l'environnement d'exécution Java détermine automatiquement si la règle accorde le

droit d'accès requis uniquement à un principal spécifique et, si c'est le cas, l'opération n'est autorisée que si le Subject associé au contexte de contrôle d'accès contient bien le principal désigné.

Vous devez être familiarisé avec la syntaxe du fichier de règles. Vous trouverez une description détaillée de l'autorisation JAAS dans le Guide de référence JAAS.

Un codebase spécial de WebSphere eXtreme Scale permet de vérifier l'autorisation JAAS sur les appels aux méthodes ObjectMap et JavaMap. Il s'agit du codebase spécial `http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction`. Utilisez-le pour l'octroi de droits d'accès d'ObjectMap ou de JavaMap à des principaux. Ce code spécial a été créé parce que le fichier JAR d'eXtreme Scale dispose de tous les droits d'accès.

Le modèle de la règle permettant d'accorder la permission MapPermission est :

```
grant codeBase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
   <champs principaux>{
    permission com.ibm.websphere.objectgrid.security.MapPermission
               "[ObjectGrid_name].[ObjectMap_name]", "action";
    ....
    permission com.ibm.websphere.objectgrid.security.MapPermission
               "[ObjectGrid_name].[ObjectMap_name]", "action";
  };
```

Un champ Principal se présente comme dans l'exemple suivant :

```
principal Principal_class "principal_name"
```

Dans cette règle, seuls les droits d'accès insert et read sont accordés à ces quatre mappes vers un certain principal. L'autre fichier de règles, fullAccessAuth.policy, accorde à ces mappes tous les droits d'accès à un principal. Avant d'exécuter l'application, modifiez le principal_name et la classe principal. La valeur de principal_name dépend du registre des utilisateurs. Par exemple, si le système d'exploitation local sert de registre des utilisateurs, que le nom de la machine est MACH1 et l'ID utilisateur user1, le principal_name sera MACH1/user1.

Avant d'être définie de l'une des deux manières suivantes, la règle d'autorisation JAAS peut être placée directement dans le fichier de règles Java ou dans un fichier d'autorisation JAAS distinct :

- à l'aide de l'argument JVM suivant :

  ```
  -Djava.security.auth.policy=file:[JAAS_AUTH_POLICY_FILE]
  ```

- à l'aide de la propriété suivante dans le fichier java.security :

  ```
  -Dauth.policy.url.x=file:[JAAS_AUTH_POLICY_FILE]
  ```

Autorisation ObjectGrid personnalisée

Le plug-in ObjectGridAuthorization permet d'autoriser ObjectGrid, ObjectMap et JavaMap à accéder de manière personnalisée aux principaux représentés par un objet Subject. La plupart du temps, ce plug-in est implémenté pour extraire les principaux d'un objet Subject et vérifier ensuite si les permissions spécifiées sont ou non accordées à ces principaux.

Les droits d'accès passés à la méthode checkPermission(Subject, Permission) peuvent être de l'un des types suivants :

- MapPermission
- ObjectGridPermission

- AgentPermission
- ServerMapPermission

Pour plus de détails, voir la documentation de l'API ObjectGridAuthorization.

Le plug-in ObjectGridAuthorization peut être configuré de l'une des manières suivantes :

- Configuration XML

  Vous pouvez utiliser le fichier XML ObjectGrid pour définir un plug-in ObjectAuthorization. Par exemple :

```
<objectGrids>
 <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
  authorizationMechanism="AUTHORIZATION_MECHANISM_CUSTOM">
...
  <bean id="ObjectGridAuthorization"
className="com.acme.ObjectGridAuthorizationImpl" />
</objectGrids>
```

- Configuration par programmation

  Si vous souhaitez créer un ObjectGrid à l'aide de la méthode ObjectGrid.setObjectGridAuthorization(ObjectGridAuthorization) de l'API, vous pouvez appeler la méthode suivante pour définir le plug-in d'autorisation. Cette méthode ne s'applique qu'au modèle de programmation eXtreme Scale local lorsqu'on instancie directement l'instance ObjectGrid.

```
/**
     * Sets the <code>ObjectGridAuthorization</code> for this ObjectGrid instance.
     * <p>
     * Passing <code>null</code> to this method removes a previously set
     * <code>ObjectGridAuthorization</code> object from an earlier invocation of this method
     * and indicates that this <code>ObjectGrid</code> is not associated with a
     * <code>ObjectGridAuthorization</code> object.
     * <p>
     * This method should only be used when ObjectGrid security is enabled. If
     * the ObjectGrid security is disabled, the provided <code>ObjectGridAuthorization</code> object
     * will not be used.
     * <p>
     * A <code>ObjectGridAuthorization</code> plugin can be used to authorize
     * access to the ObjectGrid and maps. Please refer to <code>ObjectGridAuthorization</code> for more details.
     *
     * <p>
     * As of XD 6.1, the <code>setMapAuthorization</code> is deprecated and
     * <code>setObjectGridAuthorization</code> is recommended for use. However,
     * if both <code>MapAuthorization</code> plugin and <code>ObjectGridAuthorization</code> plugin
     * are used, ObjectGrid will use the provided <code>MapAuthorization</code> to authorize map accesses,
     * even though it is deprecated.
     * <p>
     * Note, to avoid an <code>IllegalStateException</code>, this method must be
     * called prior to the <code>initialize()</code> method.  Also, keep in mind
     * that the <code>getSession</code> methods implicitly call the
     * <code>initialize()</code> method if it has yet to be called by the
     * application.
     *
     * @param ogAuthorization the <code>ObjectGridAuthorization</code> plugin
     *
     * @throws IllegalStateException if this method is called after the
     *         <code>initialize()</code> method is called.
     *
     * @see #initialize()
     * @see ObjectGridAuthorization
     * @since WAS XD 6.1
     */
    void setObjectGridAuthorization(ObjectGridAuthorization ogAuthorization);
```

Implémenter ObjectGridAuthorization

La méthode booléenne checkPermission(Subject subject, Permission permission) de l'interface ObjectGridAuthorization est appelée par l'environnement d'exécution WebSphere eXtreme Scale afin de vérifier si l'objet Subject qui est passé dispose de la permission passed-in. L'implémentation de l'interface ObjectGridAuthorization retourne true si l'objet dispose bien de cette permission et false dans le cas contraire.

La plupart du temps, ce plug-in est implémenté pour extraire les principaux d'un objet Subject et vérifier ensuite, en consultant des règles spécifiques, si les droits d'accès spécifiés sont ou non accordés à ces principaux. Ce sont les utilisateurs qui définissent ces règles. Les règles peuvent ainsi aussi bien être définies dans une base de données, dans un fichier texte ou sur un serveur de règles Tivoli Access Manager.

Nous pouvons, par exemple, utiliser un serveur de règles Tivoli Access Manager pour gérer la règle d'autorisation et utiliser son API pour autoriser l'accès. Pour savoir comment utiliser les API Tivoli Access Manager Authorization, voir le guide IBM Tivoli Access Manager Authorization Java Classes Developer Reference.

Cet exemple d'implémentation part des présuppositions suivantes :

- l'on ne vérifie que l'autorisation pour MapPermission. Pour les autres droits d'accès, toujours retourner `true`
- l'objet Subject contient un principal com.tivoli.mts.PDPrincipal
- le serveur de règles Tivoli Access Manager a défini les droits d'accès suivants pour le nom de l'objet ObjectMap ou JavaMap. L'objet qui est défini sur le serveur de règles doit avoir le même nom que l'ObjectMap ou le JavaMap au format [ObjectGrid_name].[ObjectMap_name]. Le droit d'accès est le premier caractère des chaînes de droits d'accès qui sont définies dans la MapPermission. Exemple : "r" représente le droit d'accès read à la mappe ObjectMap

Le fragment de code suivant montre comment implémenter la méthode checkPermission :

```
/**
* @see com.ibm.websphere.objectgrid.security.plugins.
*   MapAuthorization#checkPermission
* (javax.security.auth.Subject, com.ibm.websphere.objectgrid.security.
*   MapPermission)
*/
public boolean checkPermission(final Subject subject,
 Permission p) {

  // For non-MapPermission, we always authorize.
  if (!(p instanceof MapPermission)){
    return true;
  }

  MapPermission permission = (MapPermission) p;

  String[] str = permission.getParsedNames();

  StringBuffer pdPermissionStr = new StringBuffer(5);
  for (int i=0; i<str.length; i++) {
    pdPermissionStr.append(str[i].substring(0,1));
  }

  PDPermission pdPerm = new PDPermission(permission.getName(),
  pdPermissionStr.toString());

  Set principals = subject.getPrincipals();

  Iterator iter= principals.iterator();
  while (iter.hasNext()) {
    try {
      PDPrincipal principal = (PDPrincipal) iter.next();
      if (principal.implies(pdPerm)) {
        return true;
      }
    }
```

```
    catch (ClassCastException cce) {
      // Handle exception
    }
  }
  return false;
}
```

Authentification de grille

Vous pouvez utiliser le plug-in du gestionnaire de jetons de sécurité pour activer l'authentification serveur à serveur, ce qui signifie d'implémenter l'interface SecureTokenManager.

La méthode generateToken(Object) prend un objet protect, puis génère un jeton pouvant être compris par les autres. La méthode verifyTokens(byte[]) procède en sens contraire : elle reconvertit le jeton en objet d'origine.

Une implémentation SecureTokenManager simple utilise un algorithme de codage de base, tel qu'un algorithme XOR, pour coder l'objet dans un formulaire sérialisé et utiliser l'algorithme de décodage correspondant pour décoder le jeton. Cette implémentation n'est pas sécurisée et peut être facilement interrompue.

Implémentation par défaut de WebSphere eXtreme Scale

WebSphere eXtreme Scale met immédiatement à disposition une implémentation de cette interface. Cette implémentation par défaut utilise une paire de clés pour signer et vérifier la signature et une clé confidentielle pour chiffrer le contenu. Chaque serveur comporte un fichier de clés de type JCKES contenant la paire de clés, une clé privée et une clé publique ainsi qu'une clé confidentielle. Pour stocker les clés confidentielles, le fichier de clés doit être de type JCKES. En effet, ces clés servent à chiffrer et signer ou vérifier la chaîne secrète côté expéditeur. De plus, le jeton est associé à un délai d'expiration. Côté récepteur, les données sont vérifiées, déchiffrées et comparées à la chaîne secrète du récepteur. Des protocoles de communication SSL (Secure Sockets Layer) ne sont pas requis entre une paire de serveurs pour l'authentification car les clés privées et les clés publiques ont la même finalité. Toutefois, si la communication du serveur n'est pas chiffrée, les données peuvent être dérobées à la seule vue de la communication. Le jeton venant à expiration, la menace d'attaque par relecture est minimisée. Ce risque est considérablement réduit si tous les serveurs sont déployés derrière un pare-feu.

L'inconvénient de cette approche : les administrateurs de WebSphere eXtreme Scale doivent générer des clés et les transporter vers tous les serveurs, ce qui peut provoquer des failles de sécurité lors du transport.

Sécurité locale

WebSphere eXtreme Scale fournit plusieurs points de contact de sécurité permettant d'intégrer les mécanismes personnalisés. Dans le modèle de programmation local, la principale fonction de sécurité est l'autorisation, qui n'est associée à aucune prise en charge de l'authentification. Vous devez vous authentifier en dehors de WebSphere Application Server. Toutefois, des plug-in permettant d'obtenir et de valider des objets Subject sont fournis.

Activation de la sécurité

La liste ci-dessous indique les deux méthodes d'activation de la sécurité locale :

- **Configuration du fichier XML** Vous pouvez utiliser le fichier XML ObjectGrid pour définir une ObjectGrid et activer la sécurité pour celle-ci. Le fichier secure-objectgrid-definition.xml utilisé dans l'exemple d'application d'entreprise ObjectGridSample est présenté dans l'exemple suivant. Dans ce fichier XML, la sécurité est activée en définissant l'attribut securityEnabled sur true.

```
<objectGrids>
    <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
        authorizationMechanism="AUTHORIZATION_MECHANISM_JASS">
    ...
</objectGrids>
```

- **Programmation** Pour créer une ObjectGrid à l'aide de la méthode API ObjectGrid.setSecurityEnabled(), appelez la méthode suivante dans l'interface ObjectGrid et activez la sécurité.

```
/**
 * Enable the ObjectGrid security
 */
void setSecurityEnabled();
```

Authentification

Dans le modèle de programmation local, eXtreme Scale ne fournit aucun mécanisme d'authentification mais repose sur l'environnement, serveurs d'applications ou applications, pour l'authentification. Lors de l'utilisation d'eXtreme Scale dans WebSphere Application Server ou WebSphere Extended Deployment, les applications peuvent utiliser le mécanisme d'authentification de sécurité WebSphere Application Server. Lors de l'exécution d'eXtreme Scale dans un environnement Java 2 Platform, Standard Edition (J2SE), l'application doit gérer les authentifications à l'aide de l'authentification JAAS (Java Authentication and Authorization Service) ou d'autres mécanismes d'authentification. Pour plus d'informations sur l'utilisation de l'authentification JAAS, reportez-vous auguide de référence JAAS. Le contrat entre une application et une instance ObjectGrid est l'objet javax.security.auth.Subject. Après l'authentification du client par le serveur d'applications ou l'application, cette dernière peut extraire l'objet javax.security.auth.Subject authentifié et utiliser cet objet Subject pour obtenir une session de l'instance ObjectGrid en appelant la méthode ObjectGrid.getSession(Subject). Cet objet Subject est utilisé pour autoriser l'accès aux données de mappe. Ce contrat est appelé mécanisme de passage du sujet. L'exemple ci-dessous illustre l'API ObjectGrid.getSession(Subject).

```
/**
 * Cette API permet au cache d'utiliser un sujet spécifique et non celui
 * configuré sur l'ObjectGrid pour obtenir une session.
 * @param subject
 * @return An instance of Session
 * @throws ObjectGridException
 * @throws TransactionCallbackException
 * @throws InvalidSubjectException the subject passed in is not valid based
 * on the SubjectValidation mechanism.
 */
public Session getSession(Subject subject)
throws ObjectGridException, TransactionCallbackException, InvalidSubjectException;
```

La méthode ObjectGrid.getSession() de l'interface ObjectGrid peut également être utilisée pour obtenir un objet Session :

```
/**
 * Cette méthode renvoie un objet Session utilisable par une seule unité d'exécution à la fois.
 * Vous ne pouvez pas partager cet objet Session entre des unités d'exécution sans l'entourer d'une
 * section critique. Alors que l'infrastructure principale permet à l'objet de se déplacer entre les
 * unités d'exécution, TransactionCallback et Loader peuvent entraver cette utilisation,
 * notamment dans les environnements J2EEs. Lorsque la sécurité est activée, cette méthode utilise
 * SubjectSource pour obtenir l'objet Subject.
```

```
 *
 * Si la méthode initialize n'est pas appelée avant la première
 * invocation getSession, une initialisation implicite a lieu.  Cette
 * initialisation fait en sorte que toute la configuration soit terminée avant
 * que l'utilisation de l'exécution soit requise.
 *
 * @see #initialize()
 * @return An instance of Session
 * @throws ObjectGridException
 * @throws TransactionCallbackException
 * @throws IllegalStateException if this method is called after the
 *         destroy() method is called.
 */
public Session getSession()
throws ObjectGridException, TransactionCallbackException;
```

Comme le spécifie la documentation d'API, lorsque la sécurité est activée, cette méthode utilise le plug-in SubjectSource pour obtenir un objet Subject. Le plug-in SubjectSource est l'un des plug-in de sécurité définis dans eXtreme Scale pour prendre en charge la propagation des objets Subject. Pour plus d'informations, consultez les plug-in de sécurité. La méthode getSession(Subject) peut être appelée sur l'instance ObjectGrid locale uniquement. Si vous appelez la méthode getSession(Subject) côté client dans une configuration répartie eXtreme Scale, une exception IllegalStateException est émise.

Plug-in de sécurité

WebSphere eXtreme Scale fournit deux plug-in de sécurité liés au mécanisme de passage de sujet : les plug-in SubjectSource et SubjectValidation.

Plug-in SubjectSource

Le plug-in SubjectSource représenté par l'interface com.ibm.websphere.objectgrid.security.plugins.SubjectSource est un plug-in utilisé pour obtenir un objet Subject d'un environnement d'exécution eXtreme Scale. Cet environnement peut être une application utilisant l'ObjectGrid ou un serveur d'applications qui héberge l'application. Pensez à utiliser le plug-in SubjectSource comme alternative au mécanisme de passage du sujet. A l'aide du mécanisme de passage du sujet, l'application extrait l'objet et l'utilise pour obtenir l'objet de session ObjectGrid. Avec le plug-in SubjectSource, l'exécution eXtreme Scale extrait l'objet Subject et l'utilise pour obtenir l'objet de session. Le mécanisme de passage du sujet donne le contrôle des objets Subject aux applications alors que le mécanisme de plug-in SubjectSource décharge les applications de l'extraction de l'objet Subject. Vous pouvez utiliser le plug-in SubjectSource pour obtenir un objet Subject représentant un client eXtreme Scale utilisé pour l'autorisation. Lorsque la méthode ObjectGrid.getSession est appelée, le sujet getSubject émet une exception ObjectGridSecurityException si la sécurité est activée. WebSphere eXtreme Scale offre une implémentation par défaut de ce plug-in : com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectSourceImpl. Cette implémentation peut être utilisée pour extraire un sujet caller ou un sujet RunAs à partir de l'unité d'exécution lorsqu'une application est exécutée dans WebSphere Application Server. Vous pouvez configurer cette classe dans le fichier XML du descripteur d'ObjectGrid comme la classe d'implémentation SubjectSource lors de l'utilisation d'eXtreme Scale dans WebSphere Application Server. Le fragment de code suivant illustre le flux principal de la méthodeWSSubjectSourceImpl.getSubject.

```
Subject s = null;
try {
  if (finalType == RUN_AS_SUBJECT) {
    // get the RunAs subject
    s = com.ibm.websphere.security.auth.WSSubject.getRunAsSubject();
```

```
  }
  else if (finalType == CALLER_SUBJECT) {
    // obtenez le callersubject
    s = com.ibm.websphere.security.auth.WSSubject.getCallerSubject();
  }
}
catch (WSSecurityException wse) {
  throw new ObjectGridSecurityException(wse);
}

return s;
```

Pour plus de détails, reportez-vous à la documentation d'API sur le plug-in SubjectSource et l'implémentation WSSubjectSourceImpl.

Plug-in SubjectValidation

Le plug-in SubjectValidation, représenté par l'interface com.ibm.websphere.objectgrid.security.plugins.SubjectValidation, est un autre plug-in de sécurité. Ce plug-in permet de valider qu'un javax.security.auth.Subject qui est transmis à l'ObjectGrid ou extrait par le plug-in SubjectSource est un Subject valide qui n'a pas été falsifié.

La méthode SubjectValidation.validateSubject(Subject) de l'interface SubjectValidation prend un objet Subject et renvoie un objet Subject. Vos implémentations déterminent si un objet Subject est considéré comme valide ou quel objet Subject est renvoyé. Si l'objet Subject n'est pas valide, une exception InvalidSubjectException est émise.

Vous pouvez utiliser ce plug-in si vous ne faites pas confiance à l'objet Subject transmis à cette méthode. Ce cas est rare si l'on considère que vous faites confiance aux développeurs d'applications qui s'occupent du code permettant d'extraire l'objet Subject.

Une implémentation de ce plug-in nécessite l'assistance du créateur d'objets Subject, car le créateur est le seul à savoir si l'objet Subject a été ou non falsifié. Mais il peut arriver qu'un créateur de Subject n'ait pas les moyens de savoir si le Subject a été falsifié. Dans ce cas, ce plug-in n'est pas utile.

WebSphere eXtreme Scale fournit une implémentation par défaut de SubjectValidation: com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectValidationImpl. Vous pouvez utiliser cette implémentation pour valider le sujet WebSphere Application authentifié par le serveur. Vous pouvez configurer cette classe comme la classe d'implémentation SubjectValidation lors de l'utilisation d'eXtreme Scale dans WebSphere Application Server. L'implémentation WSSubjectValidationImpl considère qu'un objet Subject est valide uniquement si le jeton d'informations d'identification associé à ce sujet n'a pas été falsifié. Vous pouvez modifier d'autres parties de l'objet Subject. L'implémentation WSSubjectValidationImpl demande à WebSphere Application Server le sujet d'origine correspondant au jeton d'informations d'identification et renvoie l'objet Subject d'origine comme objet Subject validé. Par conséquent, les modifications apportées au contenu Subject autre que le jeton d'informations d'identification ne prennent pas effet. Le fragment de code suivant illustre le flux de base de la méthode WSSubjectValidationImpl.validateSubject(Subject).

```
// Créez un LoginContext avec un schéma WSLogin et
// transmettez un gestionnaire d'appel.
LoginContext lc = new LoginContext("WSLogin",
```

```
new WSCredTokenCallbackHandlerImpl(subject));

// Lorsque cette méthode est appelée, les méthodes du gestionnaire d'appel
// sont appelées pour établir la connexion de l'utilisateur.
lc.login();

// Obtenez le sujet du LoginContext
return lc.getSubject();
```

Dans le fragment de code précédent, un objet de gestionnaire d'appel de jeton d'informations d'identification, WSCredTokenCallbackHandlerImpl, est créé avec l'objet Subject à valider. Un objet LoginContext est ensuite créé avec le schéma de connexion WSLogin. Lorsque la méthode lc.login est appelée, la sécuritéWebSphere Application Server extrait le jeton d'informations d'identification de l'objet Subject, puis renvoie l'objet Subject correspondant en tant qu'objet Subject validé.

Pour plus de détails, reportez-vous aux API Java de l'implémentation SubjectValidation et WSSubjectValidationImpl.

Configuration du plug-in

Vous pouvez configurer le plug-in SubjectValidation et le plug-in SubjectSource de deux façons :

- **Configuration XML**Vous pouvez utiliser le fichier XML ObjectGrid pour définir une ObjectGrid et définir ces deux plug-in. Voici un exemple dans lequel la classe WSSubjectSourceImpl est configurée comme plug-in SubjectSource et la classe WSSubjectValidation est configurée comme plug-in SubjectValidation.

  ```
  <objectGrids>
   <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
    authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS">
          <bean id="SubjectSource"
  className="com.ibm.websphere.objectgrid.security.plugins.builtins.
   WSSubjectSourceImpl" />
      <bean id="SubjectValidation"
   className="com.ibm.websphere.objectgrid.security.plugins.builtins.
   WSSubjectValidationImpl" />
      <bean id="TransactionCallback"
  className="com.ibm.websphere.samples.objectgrid.
   HeapTransactionCallback" />
  ...
  </objectGrids>
  ```

- **Programmation** Si vous souhaitez créer une ObjectGrid à l'aide d'une API, vous pouvez appeler la méthode suivante pour définir le plug-in SubjectSource ou SubjectValidation.

  ```
  **
  * Définissez le plug-in SubjectValidation pour cette instance ObjectGrid. Un
  * plug-in SubjectValidation peut être utilisé pour valider l'objet Subject
  * transmis en tant que sujet valide. Reportez-vous à {@link SubjectValidation}
  * pour plus d'informations.
  * @param subjectValidation the SubjectValidation plug-in
  */
  void setSubjectValidation(SubjectValidation subjectValidation);


  /**
  * Définissez le plug-in SubjectSource. Un plug-in SubjectSource peut être utilisé
  * pour obtenir un objet Subject de l'environnement pour représenter le
  * client ObjectGrid.
  *
  ```

```
 * @param source the SubjectSource plug-in
 */
 void setSubjectSource(SubjectSource source);
```

Ecriture du code d'authentification JAAS

Vous pouvez écrire votre propre code d'authentification JASS (Java Authentication and Authorization Service) pour gérer l'authentification. Vous devez écrire vos modules de connexion et les configurer pour le module d'authentification.

Le module de connexion reçoit des informations sur un utilisateur et authentifie ce dernier. Il peut s'agir de n'importe quelles informations identifiant l'utilisateur. Par exemple, ces informations peuvent être un ID utilisateur et un mot de passe, un certificat client, etc. Après avoir reçu les informations, le module de connexion vérifie que les informations représentent un sujet valide et crée ensuite un objet Subject. Plusieurs implémentations de modules de connexion sont disponible au public actuellement.

Après avoir écrit un module de connexion, configurez-le pour l'exécution à utiliser. Vous devez configurer un module de connexion JAAS. Ce module de connexion contient le module de connexion et son schéma d'authentification. For exemple :

```
FileLogin
{
    com.acme.auth.FileLoginModule required
};
```

Le schéma d'authentification est FileLogin et le module de connexion est com.acme.auth.FileLoginModule. Le jeton requis indique que le module FileLoginModule doit valider cette connexion faute de quoi le schéma échouera.

Le fichier de configuration du module de connexion JAAS peut être défini de l'une des façons suivantes :

- Définissez le fichier de configuration du module de connexion JAAS dans la propriété login.config.url du fichier java.security, par exemple :

  ```
  login.config.url.1=file:${java.home}/lib/security/file.login
  ```

- Définissez le fichier de configuration du module de connexion JAAS à partir de la ligne de commande à l'aide des arguments **-Djava.security.auth.login.config** de la machine virtuelle Java, par exemple, `-Djava.security.auth.login.config ==$JAVA_HOME/lib/security/file.login`

Si votre code est exécuté dans WebSphere Application Server, vous devez configurer la connexion JAAS dans la console d'administration et stocker cette configuration de connexion dans la configuration du serveur d'applications. Pour plus d'informations, reportez-vous à la rubrique relative à la configuration de connexion pour JAAS (Java Authentication and Authorization Service).

Chapitre 10. Points à prendre en considération par les développeurs d'applications concernant les performances

Pour améliorer les performances de votre grille de données en mémoire ou de votre espace de traitement de base de données, vous pouvez examiner un certain nombre de points à prendre en considération comme l'optimisation du paramétrage des machines virtuelles Java et l'application des pratiques recommandées pour les fonctionnalités du produit telles que le verrouillage, la sérialisation et les performances des requêtes.

Optimisation des machines virtuelles Java

Vous devez prendre en compte plusieurs aspects spécifiques de l'optimisation des machines virtuelles Java (JVM) pour de meilleures performances de WebSphere eXtreme Scale.

La configuration recommandée est d'avoir des segments de 1 à 2 Go avec une machine virtuelle Java pour quatre coeurs. La taille des segments dépend de la nature des objets qui sont stockées sur les serveurs, point que nous aborderons un peu plus loin.

Taille de segment et récupération de place : préconisations

La taille optimale des segments dépend de trois facteurs :

1. le nombre d'objets actifs présents dans le segment
2. le degré de complexité des objets actifs présents dans le segment
3. le nombre de coeurs utilisables par la machine virtuelle Java

Par exemple, une application stockant des tableaux d'octets de 10 Ko pourra exécuter un segment bien plus important qu'une application utilisant des graphes complexes POJO.

Toutes les machines virtuelles Java modernes utilisent aujourd'hui des algorithmes de récupération de place en parallèle ; en d'autres termes, utiliser davantage de coeurs peut réduire les pauses dans la récupération de place. Et la place sera récupérée plus rapidement sur des systèmes 8 coeurs que sur des systèmes quadricoeurs.

Utilisation de la mémoire réelle et taille spécifiée pour les segments

Une machine virtuelle Java avec un segment de 1 Go utilise en fait approximativement 1,3 Go de mémoire réelle. Dans notre lab, nous sommes arrivés à faire tourner des machines virtuelles Java de 1 Go sur un système ayant 16 Go de RAM. Le système a commencé à paginer une fois que les segments ont été pleins jusqu'à 800 Mo ou plus.

Récupération de place

Dans le cas de machines virtuelles Java IBM, utilisez le collecteur avgoptpause pour des scénarios d'actualisations à haut débit (100% des transactions modifient les entrées). Le collecteur gencon fonctionne encore mieux qu'avgoptpause dans le

cas de scénarios où les données sont actualisées de manière relativement peu fréquentes (10 % du temps, voire moins). A vous de tester les deux collecteurs pour voir celui qui fonctionne le mieux dans votre cas de figure. Si vous constatez un problème de performances, activez le mode prolixe pour la récupération de place afin de vérifier le pourcentage de temps passé à cette récupération. Il a pu être constaté des scénarios où 80 % du temps était consacré à la récupération de place, jusqu'à ce que l'optimisation règle le problème.

Performances des machines virtuelles Java

WebSphere eXtreme Scale peut s'exécuter sur des versions différentes de Java 2 Platform, Standard Edition (J2SE). ObjectGrid version 6.1 prend en charge J2SE version 1.4.2 et plus récentes. Pour une meilleure productivité de développement et de meilleures performances, utilisez J2SE 5 ou plus récent afin de profiter des annotations et d'une récupération de place améliorée. ObjectGrid tourne sur des machines virtuelles Java 32 ou 64 bits.

Les clients ObjectGrid version 6.0.2 peuvent se connecter à une grille ObjectGrid version 6.1. Utilisez des clients ObjectGrid version 6.1 pour des clients J2SE version 1.4.2 ou mieux. La seule raison d'utiliser un client ObjectGrid version 6.0.2 est sa prise en charge de J2SE version 1.3.

WebSphere eXtreme Scale est testé avec un sous-ensemble des machines virtuelles disponibles, mais la liste des machines prises en charge n'est pas exhaustive. Vous pouvez faire tourner WebSphere eXtreme Scale sur n'importe quelle version 1.4.2 ou supérieure, mais si un problème est rencontré sur la machine virtuelle Java, vous devrez contacter le fournisseur de celle-ci. Dans la mesure du possible, sur les plateformes compatibles WebSphere Application Server, utilisez la machine virtuelle Java provenant de l'environnement d'exécution WebSphere.

La meilleure machine virtuelle Java est la machine Java Platform, Standard Edition 6. Java 2 Platform, Standard Edition, v 1.4 offre de médiocres performances, particulièrement dans les scénarios où le collecteur gencon fait la différence. Les performances de Java Platform Standard Edition 5 sont satisfaisantes mais Java Platform, Standard Edition 6 fonctionne encore mieux.

Optimisation d'orb.properties

Nous recommandons d'utiliser en production le fichier orb.properties qui suit. Dans notre lab, nous avons utilisé ce fichier sur des grilles contenant jusqu'à 1 500 machines virtuelles Java. Le fichier orb.properties se trouve dans le dossier lib du JRE utilisé.

```
# IBM JDK properties for ORB
org.omg.CORBA.ORBClass=com.ibm.CORBA.iiop.ORB
org.omg.CORBA.ORBSingletonClass=com.ibm.rmi.corba.ORBSingleton

# WS Interceptors
org.omg.PortableInterceptor.ORBInitializerClass.com.ibm.ws.objectgrid.corba.ObjectGridInitializer

# WS ORB & Plugins properties
com.ibm.CORBA.ForceTunnel=never
com.ibm.CORBA.RequestTimeout=10
com.ibm.CORBA.ConnectTimeout=10

# Needed when lots of JVMs connect to the catalog at the same time
com.ibm.CORBA.ServerSocketQueueDepth=2048

# Clients and the catalog server can have sockets open to all JVMs
com.ibm.CORBA.MaxOpenConnections=1016

# Thread Pool for handling incoming requests, 200 threads here
com.ibm.CORBA.ThreadPool.IsGrowable=false
com.ibm.CORBA.ThreadPool.MaximumSize=200
com.ibm.CORBA.ThreadPool.MinimumSize=200
```

```
com.ibm.CORBA.ThreadPool.InactivityTimeout=180000

# No splitting up large requests/responses in to smaller chunks
com.ibm.CORBA.FragmentSize=0
```

Nombre des unités d'exécution

Le nombre d'unités d'exécution dépend de quelques facteurs. Il existe une limite au nombre d'unités d'exécution que peut gérer un seul fragment. Un nombre plus élevé de fragments pour chaque machine virtuelle Java a pour conséquence un nombre plus élevé d'unités d'exécution et donc d'accès simultanés. Chaque fragment supplémentaire apporte davantage de chemins simultanés vers les données. Chaque fragment est aussi simultané que possible, mais, même ici, il existe une limite.

Pratiques CopyMode recommandées

WebSphere eXtreme Scale effectue une copie de la valeur en appliquant l'un des six paramétrages possibles pour CopyMode. C'est à vous de déterminer lequel de ces paramétrages fonctionne le mieux pour les besoins de votre déploiement.

La méthode setCopyMode(CopyMode, valueInterfaceClass) de l'API BackingMap permet de définir le mode de copie d'après l'un des champs statiques finaux suivants qui sont définis dans la classe com.ibm.websphere.objectgrid.CopyMode.

Lorsqu'une application utilise l'interface ObjectMap pour obtenir une référence à une entrée de mappe, n'utilisez cette référence qu'au sein de la transaction WebSphere eXtreme Scale qui s'est procuré cette référence. Son utilisation dans un autre transaction peut provoquer des erreurs. Si, par exemple, vous utilisez la stratégie de verrouillage pessimiste pour la mappe de sauvegarde, un appel à la méthode get ou à la méthode getForUpdate acquerra un verrou S (shared) ou U (update), selon la transaction. La méthode get retournera la référence à la valeur et le verrou obtenu sera libéré au terme de la transaction. La transaction doit appeler les méthodes get ou getForUpdate pour verrouiller l'entrée de mappe dans une autre transaction. Chaque transaction doit obtenir sa propre référence à la valeur en appelant les méthodes get ou getForUpdate au lieu de réutiliser la même référence dans plusieurs transactions.

CopyMode pour les mappes d'entités

Lorsqu'on utilise une mappe qui est associée à une entité d'API EntityManager, la mappe retourne toujours directement les objets tuple d'entités sans procéder à une copie, sauf si l'on utilise COPY_TO_BYTES comme mode de copie. Il est donc important de modifier CopyMode, faute de quoi le tuple ne serait pas copié de manière appropriée en cas de changement.

COPY_ON_READ_AND_COMMIT

Le mode COPY_ON_READ_AND_COMMIT est le mode par défaut. L'argument valueInterfaceClass est ignoré lorsque ce mode est utilisé. Ce mode s'assure qu'une application ne contient pas de référence à l'objet value qui se trouve dans la mappe de sauvegarde. A la place, l'application utilise toujours une copie de la valeur qui se trouve dans l'instance BackingMap. Le mode COPY_ON_READ_AND_COMMIT garantit que l'application ne pourra jamais endommager par inadvertance les données qui sont mises en cache dans la mappe de sauvegarde. Lorsqu'une transaction d'application appelle une méthode ObjectMap.get pour une clé donnée et qu'il s'agit du premier accès à l'entrée ObjectMap de cette clé, c'est une copie de

la valeur qui est retournée. Lorsque la transaction est validée, toutes les modifications validées par l'application sont alors copiées vers la mappe de sauvegarde pour garantir que l'application ne dispose d'aucune référence à la valeur validée dans la mappe.

COPY_ON_READ

Par rapport au mode COPY_ON_READ_AND_COMMIT, le mode COPY_ON_READ donne de meilleures performances car il élimine la copie qui est effectuée lors de la validation des transactions. L'argument valueInterfaceClass est ignoré lorsque ce mode est utilisé. Pour conserver l'intégrité des données de la mappe de sauvegarde, l'application s'assure que toutes les références à une entrée sont supprimées une fois la transaction validée. Dans ce mode, la méthode ObjectMap.get retourne une copie de la valeur au lieu d'une référence à celle-ci afin de garantir que les modifications de la valeur opérées par l'application n'affecteront pas la valeur dans la mappe tant que la transaction n'est pas validée. Mais la différence est que, lors de la validation, il n'est effectué aucune copie des modifications. Ce qui est stocké dans la mappe de sauvegarde, c'est la référence à la copie, celle qui a été retournée par la méthode ObjectMap.get. Une fois la transaction validée, l'application détruit toutes les références aux entrées de la mappe. Si l'application ne détruit pas les références, les données mises en cache dans la mappe de sauvegarde risquent d'être endommagées. Si une application utilise ce mode et rencontre des problèmes, passez en mode COPY_ON_READ_AND_COMMIT pour voir si le problème persiste. S'il disparaît, c'est que l'application ne parvient pas à détruire toutes ses références après que la transaction a été validée.

COPY_ON_WRITE

Par rapport au mode COPY_ON_READ_AND_COMMIT, le mode COPY_ON_WRITE donne lui aussi de meilleures performances car il élimine la copie qui est effectuée lorsque la méthode ObjectMap.get est appelée pour la première fois par une transaction pour une clé donnée. La méthode ObjectMap.get retourne un proxy de la valeur au lieu d'une référence directe à l'objet value. Le proxy garantit qu'aucune copie de la valeur n'est effectuée tant que l'application n'appelle pas de méthode set sur l'interface Value qui est spécifiée comme argument valueInterfaceClass. Le proxy fournit une copie lors de l'écriture. Lors de la validation d'une transaction, la mappe de sauvegarde examine le proxy pour déterminer si une copie a été effectuée en tant que résultat de l'appel à une méthode set. Si c'est le cas, la référence à cette copie est stockée dans la mappe de sauvegarde. Ce mode présente donc un énorme avantage : une valeur n'est jamais copiée lors d'une lecture ou lors d'une validation lorsque la transaction ne fait jamais appel à une méthode set pour modifier la valeur.

Les modes COPY_ON_READ_AND_COMMIT et COPY_ON_READ procèdent tous les deux à une copie complète lorsqu'une valeur est extraite de la mappe d'objet. Ces modes ne sont pas optimaux si une application ne modifie que certaines des valeurs qui sont extraites dans une transaction. A cet égard, le mode COPY_ON_WRITE est beaucoup plus efficace, mais il requiert que l'application utilise un pattern simple. Les objets value sont tenus de prendre en charge une interface. L'application doit utiliser les méthodes de cette interface lorsqu'elle interagit avec la valeur dans une session eXtreme Scale. Si c'est le cas, eXtreme Scale crée des proxys pour les valeurs qui sont retournées à l'application. Le proxy a une référence qui est une valeur réelle. Si l'application n'effectue que des opérations de lecture, ces dernière s'exécutent toujours sur la copie réelle. Si l'application modifie un attribut dans l'objet, le proxy commence par créer une

copie de l'objet réel, puis il effectue la modification dans cette copie. A partir de ce stade, le proxy n'intervient que sur cette copie. L'utilisation de la copie permet à l'opération de copie d'être totalement évitée pour les objets qui ne sont que lus par l'application. Toutes les opérations de modification doivent commencer par le préfixe set. Les JavaBeans Enterprise sont normalement programmés pour nommer de la sorte les méthodes qui modifient les attributs des objets. Il convient donc de respecter cette convention. Tous les objets qui sont modifiés sont copiés au moment même où ils sont modifiés par l'application. Ce scénario de lecture-écriture est le scénario le plus efficace pris en charge par eXtreme Scale. Pour configurer une mappe afin qu'elle utilise le mode COPY_ON_WRITE, utilisez l'exemple qui suit. Dans cet exemple, l'application stocke des objets Person qui sont indexés à l'aide du nom (name) présent dans la mappe. L'objet Person est représenté dans le fragment de code qui suit.

```
class Person {
    String name;
    int age;
    public Person() {
    }
    public void setName(String n)  {
        name = n;
    }
    public String getName() {
        return name;
    }
    public void setAge(int a) {
        age = a;
    }
    public int getAge() {
        return age;
    }
}
```

L'application n'utilise l'interface IPerson que lorsqu'elle interagit avec des valeurs qui sont extraites d'un ObjectMap. Modifiez l'objet pour qu'il utilise une interface comme dans l'exemple qui suit.

```
interface IPerson
{
    void setName(String n);
    String getName();
    void setAge(int a);
    int getAge();
}
// L'on modifie Person pour implémenter l'interface IPerson
class Person implements IPerson {
    ...
}
```

L'application a alors besoin de configurer la mappe de sauvegarde pour que celle-ci utilise le mode COPY_ON_WRITE, comme dans l'exemple qui suit :

```
ObjectGrid dg = ...;
BackingMap bm = dg.defineMap("PERSON");
// L'on utilise COPY_ON_WRITE pour cette mappe
// avec IPerson comme classe valueProxyInfo
bm.setCopyMode(CopyMode.COPY_ON_WRITE,IPerson.class);
// L'application doit alors utiliser
// le pattern suivant lorsqu'on utilise la mappe PERSON.
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
...
sess.begin();
// l'application transtype en IPerson et non en Person la valeur retournée
IPerson p = (IPerson)person.get("Billy");
```

```
p.setAge(p.getAge()+1);
...
// l'on fabrique un nouveau Person et on l'ajoute à Map
Person p1 = new Person();
p1.setName("Bobby");
p1.setAge(12);
person.insert(p1.getName(), p1);
sess.commit();
// le fragment suivant NE FONCTIONNERA PAS. Il se traduira par
une ClassCastException sess.begin();
// l'erreur ici est que c'est Person qui est utilisé
// et non IPerson
Person a = (Person)person.get("Bobby");
sess.commit();
```

La première section montre l'application extrayant une valeur qui a été nommée Billy dans la mappe. L'application transtype la valeur retournée en objet IPerson et non en objet Person car le proxy qui est retourné implémente deux interfaces :

- l'interface spécifiée dans l'appel à la méthode BackingMap.setCopyMode
- l'interface com.ibm.websphere.objectgrid.ValueProxyInfo

Vous pouvez transtyper le proxy en deux types. La dernière partie du fragment de code montre ce qui n'est pas autorisé en mode COPY_ON_WRITE. L'application extrait l'enregistrement Bobby et essaie de le transtyper en objet Person. Cette action échoue avec une exception de transtypage de classe car le proxy qui est retourné n'est pas un objet Person. Le proxy retourné implémente en effet l'objet IPerson et l'interface ValueProxyInfo.

L'interface ValueProxyInfo et prise en charge des actualisations partielles : cette interface autorise une application à extraire soit la valeur en lecture seule validée qui est référencée par le proxy, soit l'ensemble des attributs qui ont été modifiés au cours de cette transaction.

```
public interface ValueProxyInfo {
    List /**/ ibmGetDirtyAttributes();
    Object ibmGetRealValue();
}
```

La méthode ibmGetRealValue retourne une copie en lecture seule de l'objet. L'application ne doit pas modifier cette valeur. La méthode ibmGetDirtyAttributes retourne une liste de chaînes représentant les attributs qui ont été modifiés par l'application au cours de cette transaction. ibmGetDirtyAttributes est essentiellement utilisé dans un loader JDBC (Java Database Connectivity) ou CMP. Les attributs mentionnés dans la liste sont les seuls qui ont besoin d'être actualisés, que ce soit dans l'instruction SQL ou dans l'objet mappé à la table, ce qui donne une bien plus grande efficacité au SQL généré par le loader. Lorsqu'une transaction copy on write est validée et si un loader est connecté, le loader peut transtyper vers l'interface ValueProxyInfo les valeurs des objets modifiés pour obtenir ces informations.

Gérer la méthode equals lors de l'utilisation de COPY_ON_WRITE ou de proxys : le code suivant construit un objet Person qu'il insère ensuite dans un ObjectMap. Ensuite, il extrait ce même objet à l'aide de la méthode ObjectMap.get. La valeur est transtypée vers l'interface. Si la valeur est transtypée vers l'interface Person, une exception ClassCastException est générée car la valeur retournée est un proxy qui implémente l'interface IPerson et ce n'est pas un objet Person. La vérification d'égalité échoue car l'on utilise l'opération == alors qu'il ne s'agit pas du même objet.

```
session.begin();
// new sur l'objet Person
Person p = new Person(...);
personMap.insert(p.getName, p);
// On l'extrait à nouveau (pensez à utililser l'interface pour le transtypage)
IPerson p2 = personMap.get(p.getName());
if(p2 == p) {
    // ils sont identiques
} else {
    // ils ne le sont pas
}
```

Autre point à prendre en considération : lorsqu'on doit remplacer la méthode equals. Comme le montre le fragment de code qui suit, la méthode equals doit vérifier que l'argument est un objet qui implémente l'interface IPerson et elle doit transtyper l'argument en IPerson. Comme l'argument peut très bien être un proxy qui implémente l'interface IPerson, vous devez utiliser les méthodes getAge et getName lorsque vous comparez l'égalité des variables d'instance.

```
{
    if ( obj == null ) return false;
    if ( obj instanceof IPerson ) {
        IPerson x = (IPerson) obj;
        return ( age.equals( x.getAge() ) && name.equals( x.getName() ) )
    }
    return false;
}
```

Configurations requises pour ObjectQuery et HashIndex : lorsqu'on utilise COPY_ON_WRITE avec le moteur de requête ObjectQuery ou avec un plug-in HashIndex, il est important de configurer le schéma ObjectQuery et le plug-in HashIndex pour que ces derniers puissent accéder aux objets à l'aide des méthodes de propriétés, ce qui est l'accès par défaut. S'ils sont configurés pour utiliser un accès aux champs, le moteur de requête et l'index tenteront d'accéder aux champs de l'objet proxy, ce qui retournera toujours un null ou un 0 puisque l'instance d'objet est un proxy.

NO_COPY

Le mode NO_COPY autorise une application à s'assurer qu'elle ne modifie jamais d'objet value obtenu à l'aide d'une méthode ObjectMap.get en échange d'améliorations des performances. L'argument valueInterfaceClass est ignoré lorsque ce mode est utilisé. Si ce mode est utilisé, il n'est jamais effectué de copie de la valeur. Si l'application modifie des valeurs, les données de la mappe de sauvegarde sont endommagées. Le mode NO_COPY est essentiellement utile pour les mappes en lecture seule où les données ne sont jamais modifiées par l'application. Si l'application utilise ce mode et rencontre des problèmes, passez en mode COPY_ON_READ_AND_COMMIT pour voir si le problème persiste. S'il disparaît, c'est que l'application modifie la valeur retournée par la méthode ObjectMap.get, soit pendant la transaction, soit après que celle-ci a été validée. Toutes les mappes associées aux entités de 'API EntityManager utilisent automatiquement ce mode sans tenir compte de ce qui est spécifié dans la configuration d'eXtreme Scale.

Toutes les mappes associées aux entités de 'API EntityManager utilisent automatiquement ce mode sans tenir compte de ce qui est spécifié dans la configuration d'eXtreme Scale.

COPY_TO_BYTES

Il est possible de stocker des objets dans un format sérialisé au lieu du format POJO. COPY_TO_BYTES permet de réduire l'empreinte mémoire consommée par un graphe d'objets de taille importante. Pour des informations complémentaires, voir «Mappes de tableaux d'octets», à la page 40.

Utilisation incorrecte de CopyMode

Des erreurs se produisent lorsqu'une application tente d'améliorer les performances en utilisant les modes COPY_ON_READ, COPY_ON_WRITE ou NO_COPY décrits plus haut. Les erreurs intermittentes ne se produisent pas lorsqu'on passe au mode COPY_ON_READ_AND_COMMIT.

Problème

Le problème peut être dû à des données endommagées dans la mappe ObjectGrid, ce qui est la conséquence de la violation par l'application du contrat de programmation propre au mode de copie utilisé. L'altération des données peut provoquer des erreurs imprévisibles par intermittence ou d'une manière inexpliquée ou inattendue.

Solution

L'application doit respecter le contrat de programmation qui est énoncé pour le mode de copie utilisé. Dans les modes COPY_ON_READ et COPY_ON_WRITE, l'application utilise une référence à un objet valeur extérieur à la portée de la transaction à partir de laquelle la référence a été obtenue. Pour pouvoir utiliser ces modes, l'application doit supprimer la référence à l'objet value après la fin de la transaction et obtenir une nouvelle référence à l'objet value à chaque transaction qui accède à cet objet. En mode NO_COPY, l'application ne doit jamais modifier l'objet value. Dans ce cas, soit programmez l'application pour qu'elle ne touche pas à l'objet value, soit faites-lui utiliser un autre mode de copie.

Mappes de tableaux d'octets

Vous pouvez stocker les paires clé-valeur de vos mappes dans un tableau d'octets plutôt que sous la forme d'un objet Java simple, ce qui réduit l'encombrement mémoire nécessaire à un graphe d'objets de grande taille.

Avantages

La quantité de mémoire nécessaire augmente avec le nombre d'objets d'un graphe. Lorsque vous réduisez un graphe complexe à un tableau d'octets, un seul objet est conservé dans le segment de mémoire, et non plusieurs. L'environnement d'exécution Java exécute ses recherches dans un nombre réduit d'objets lors de la récupération de place.

Le mécanisme de copie par défaut utilisé par WebSphere eXtreme Scale est la sérialisation, qui est un mécanisme coûteux. Si, par exemple, vous utilisez le mode de copie par défaut `COPY_ON_READ_AND_COMMIT`, une copie est faite au moment de la lecture et au moment de l'extraction. Avec un tableau d'octets, la valeur augmente en raison du nombre d'octets, mais aucune copie n'est exécutée au moment de la lecture, et la valeur est sérialisée en plusieurs octets, mais aucune copie n'est exécutée au moment de l'extraction. L'utilisation des tableaux d'octets résulte en

une cohérence des données équivalente à celle qui serait obtenue avec le paramètre par défaut, mais en réduisant la mémoire utilisée.

Notez qu'un mécanisme de sérialisation optimisé est indispensable pour pouvoir constater une réduction de la mémoire consommée dans le cadre de l'utilisation des tableaux d'octets. Pour plus d'informations, voir «Performances de sérialisation», à la page 231.

Configuration des mappes de tableaux d'octets

Vous pouvez activer les mappes de tableaux d'octets à l'aide du fichier XML ObjectGrid et en associant l'attribut CopyMode utilisé par une mappe au paramètre COPY_TO_BYTES, comme dans l'exemple suivant :

```
<backingMap name="byteMap" copyMode="COPY_TO_BYTES" />
```

Pour plus d'informations, consultez la rubrique relative au fichier XML descripteur d'ObjectGrid du *Guide d'administration*.

Considérations

Vous devez réfléchir à l'éventuelle utilisation des mappes de tableaux d'octets dans un scénario donné. Même si celle-ci doit vous permettre de réduire la consommation de mémoire, l'utilisation du processeur risque d'augmenter.

La liste suivante recense les différents facteurs à prendre en compte avant d'opter pour l'utilisation de la fonction de mappes de tableaux d'octets.

Type de l'objet

Avec certains types d'objet, les mappes de tableaux d'octets ne permettent pas de réduire la consommation de mémoire. Il existe donc certains types d'objets pour lesquels l'utilisation de ces mappes n'est pas recommandée. Si vous utilisez comme valeur des encapsuleurs primitifs Java ou un objet Java simple ne contenant aucune référence à d'autres objets (se contentant de stocker des champs primitifs), le nombre d'objets Java est déjà aussi peu élevé que possible : il se limite à un. La quantité de mémoire utilisée par cet objet étant déjà optimisée, l'utilisation d'une mappe de tableaux d'octets n'est pas recommandée. Ce type de mappes convient mieux aux types d'objets contenant d'autres objets ou collections d'objets dans lesquels le nombre total d'objets simples Java est supérieur à un.

Par exemple, dans le cas d'un objet Customer associé à une adresse professionnelle, à une adresse personnelle et à une collection de commandes, vous pouvez réduire le nombre d'objets du segment de mémoire et le nombre d'octets utilisés par ces objets à l'aide des mappes de tableaux d'octets.

Adresse locale

Lorsque vous utilisez d'autres modes de copie, vous pouvez optimiser les applications lors des copies si les objets sont clonables avec l'ObjectTransformer par défaut ou lorsqu'un ObjectTransformer personnalisé est fourni avec une méthode copyValue optimisée. Par rapport aux autres modes de copie, le coût des copies des opérations de lecture, d'écriture ou de validation est supérieur lorsque l'accès aux objets se fait localement. Si, par exemple, un cache local existe dans une topologie répartie ou que vous accédiez directement à une instance ObjectGrid locale ou de serveur, la durée d'accès et de validation augmentent en cas d'utilisation des mappes de tableaux d'octets du fait du coût de la sérialisation.

Dans le même type de topologie, ce coût augmente également si vous utilisez de agents de grille de données ou si vous accédez au serveur principalement lorsque vous utilisez le plug-in ObjectGridEventGroup.ShardEvents.

Interactions des plug-in

Avec les mappes de tableaux d'octets, les objets n'augmentent pas lors de la communication d'un client à un serveur, à moins que le serveur n'ait besoin du formulaire d'objet Java simple. Les plug-in qui interagissent avec la valeur de la mappe vont connaître une réduction de leurs performances en raison de l'augmentation nécessaire de la valeur.

Les plug-in utilisant LogElement.getCacheEntry ou LogElement.getCurrentValue devront subir ce coût supplémentaire. Si vous souhaitez obtenir la clé, vous pouvez utiliser LogElement.getKey, qui permet d'éviter les frais supplémentaires associés à la méthode LogElement.getCacheEntry().getKey. Les sections suivantes décrivent les plug-in dans le contexte de l'utilisation des tableaux d'octets.

Index et requêtes

Lorsque des objets sont stockés au format objet Java simple, le coût de l'indexation et de l'interrogation est minime car l'objet n'a pas besoin d'être augmenté. Lorsque vous utilisez une mappe de tableau d'objets, vous devez subir un coût supplémentaire lié à l'augmentation de l'objet. En général, si votre application utilise des index ou des requêtes, nous ne vous recommandons pas d'utiliser les mappes de tableaux d'octets, à moins que vous exécutiez uniquement des requêtes sur les attributs clés.

Verrouillage optimiste

Lorsque vous utilisez la stratégie de verrouillage optimiste, vous devrez subir un coût supplémentaire lors des opérations de mise à jour et d'invalidation. Vous devez en effet augmenter la valeur du serveur pour obtenir la valeur de la version permettant une vérification de la collision optimiste. Si vous utilisez le verrouillage optimiste uniquement pour garantir les opérations d'extraction, mais que vous n'ayez pas besoin de la vérification de collision optimiste, vous pouvez utiliser com.ibm.websphere.objectgrid.plugins.builtins.NoVersioningOptimisticCallback pour désactiver la vérification de la version.

Chargeur

Avec un Loader, vous devez également subir le coût lié à l'augmentation et à la resérialisation de la valeur. Vous pouvez cependant utiliser les mappes de tableaux d'octets avec un Loader, mais vous devez prendre en compte le coût des modifications à apporter dans ce genre de scénario. Vous pouvez par exemple utiliser la fonction de tableau d'octets dans le contexte d'un cache qui est principalement lu. Dans ce cas, l'avantage lié à un nombre réduit d'objets dans le segment de mémoire et à une utilisation moindre de la mémoire compense la perte provoquée par l'utilisation des tableaux d'octets pour les opérations d'insertion et de mise à jour.

ObjectGridEventListener

Lorsque vous utilisez la méthode transactionEnd dans le plug-in ObjectGridEventListener, vous devez supporter un coût supplémentaire côté serveur pour les demandes distantes lors de l'accès à un CacheEntry de

LogElement ou à la valeur en cours. Si l'implémentation de la méthode n'accède pas à ces champs, ce coût supplémentaire n'est pas généré.

Meilleures pratiques pour les performances de l'expulseur de plug-in

Si vous utilisez les expulseurs de plug-in, ces derniers ne sont actifs que si vous les créez et les associez à une mappe de sauvegarde. Les meilleures pratiques suivantes contribuent à renforcer les performances pour les expulseurs les moins fréquemment utilisés (LFU) et les expulseurs les moins récemment utilisés (LRU).

Expulseur le moins fréquemment utilisé (LFU)

Le concept d'un expulseur LFU consiste à supprimer les entrées d'une mappe qui ne sont pas utilisées fréquemment. Les entrées de la mappe sont réparties sur une quantité définie de segments de mémoire binaires. Lorsqu'une entrée de cache est de plus en plus sollicitée, elle est placée plus haut dans le segment de mémoire. Lorsque l'expulseur tente d'effectuer un ensemble d'expulsions, il supprime uniquement les entrées de cache situées en dessous d'un point spécifique du segment de mémoire binaire. Par conséquent, les entrées les moins fréquemment utilisées sont expulsées.

Expulseur le moins récemment utilisé (LRU)

L'expulseur LRU suit les mêmes concepts que l'expulseur LFU à quelques différentes près. Il diffère principalement en ce qu'il utilise une file d'attente premier entré, premier sorti et non un ensemble de segments de mémoire binaires. A chaque accès à une entrée de cache, il remonte en tête de la file d'attente. Par conséquent, le début de la file d'attente contient les entrées de mappe les plus récemment utilisées et la fin de la file d'attente contient les entrées de mappe les moins récemment utilisés. Par exemple, l'entrée de cache A est utilisée 50 fois et l'entrée de cache B est utilisée une seule fois juste après l'entrée de cache A. Dans ce cas, l'entrée de cache B est en tête de la file d'attente car elle a été utilisée en dernier alors que l'entrée de cache A se trouve à la fin de la file d'attente. L'expulseur LRU expulse les entrées de cache situées à la fin de la file d'attente car ce sont les entrées les moins récemment utilisées.

Propriétés LFU et LRU et meilleures pratiques pour améliorer les performances

Nombre de segments de mémoire

Lors de l'utilisation de l'expulseur LFU, toutes les entrées de cache d'une mappe donnée sont organisées en fonction du nombre de segments de mémoire spécifié, ce qui contribue à améliorer considérablement les performances et à réduire toutes les expulsions résultant de la synchronisation sur un segment de mémoire binaire qui contient toutes les organisations de la mappe. Une quantité supérieure de segments de mémoire accélère le temps requis pour réorganiser les segments de mémoire car chaque segment de mémoire dispose de moins d'entrées. Le nombre de segments de mémoire doit représenter 10 % du nombre total d'entrées de votre mappe de base.

Nombre de files d'attente

Lors de l'utilisation de l'expulseur LRU, toutes les entrées de cache d'une mappe donnée sont organisées en fonction du nombre de files d'attente LRU, ce qui contribue à améliorer considérablement les performances et à réduire toutes les

expulsions résultant de la synchronisation sur une file d'attente qui contient toutes les organisations de la mappe. Le nombre de files d'attente doit représenter 10 % du nombre total d'entrées de votre mappe de base.

Propriété MaxSize

Lorsqu'un expulseur LFU ou LRU commence à expulser des entrées, il utilise la propriété MaxSize pour déterminer le nombre de segments de mémoire binaires ou de files d'attente LRU à expulser. Par exemple, supposons que les segments de mémoire ou files d'attente comportent 10 entrées de mappe dans chaque file d'attente de mappe. Si la propriété MaxSize est définie sur 7, l'expulseur expulse 3 entrées de chaque segment de mémoire ou de file d'attente pour ramener le nombre de segments de mémoire ou de files d'attente à 7. L'expulseur expulse uniquement les entrées de mappe d'un segment de mémoire ou d'une file d'attente lorsque ce dernier ou cette dernière contient un nombre d'éléments supérieur à la valeur de la propriété MaxSize. La valeur de la propriété MaxSize doit représenter 70 % de la taille de votre segment de mémoire ou de votre file d'attente. Pour cet exemple, la valeur est définie sur 7. Vous pouvez calculer la taille approximative de chaque segment de mémoire ou file d'attente en divisant le nombre d'entrées de la mappe de base par le nombre de segments de mémoire ou files d'attente utilisés.

Propriété SleepTime

Un expulseur ne supprime pas constamment les entrées de la mappe. En revanche, il est en veille pendant un intervalle de temps déterminé et ne vérifie la mappe que toutes les n secondes où n se réfère à la propriété SleepTime. Cette propriété influe également sur les performances de manière positive : une analyse d'expulsion trop fréquente réduit les performances en raison de l'utilisation des ressources nécessaires à cette opération. Toutefois, une utilisation trop rare de l'expulseur peut entraîner la présence d'entrées superflues dans la mappe. Une mappe saturée d'entrées inutiles peut nuire à la configuration requise pour la mémoire et aux ressources de traitement nécessaires pour la mappe. Il est recommandé de définir un intervalle d'analyse d'expulsion de quinze secondes pour la plupart des mappes. Si l'écriture de la mappe est trop fréquente et si le taux de transactions est trop élevé, pensez à réduire cette valeur. En revanche, si l'accès à la mappe n'est pas fréquent, vous pouvez augmenter la valeur.

Exemple

L'exemple suivant définit une mappe, crée un expulseur LFU, définit les propriétés de l'expulseur et définit la mappe pour utiliser l'expulseur :

```
//Utilisez ObjectGridManager pour créer/obtenir la grille d'objets. Voir
// la section ObjectGridManager
ObjectGrid objGrid = ObjectGridManager.create.....
BackingMap bMap = objGrid.defineMap("SomeMap");

//Définissez les propriétés sur la base de 50 000 entrées de mappe
LFUEvictor someEvictor = new LFUEvictor();
someEvictor.setNumberOfHeaps(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

L'utilisation de l'expulseur LRU s'apparente à celle d'un expulseur LFU. Voici un exemple :

```
ObjectGrid objGrid = new ObjectGrid;
BackingMap bMap = objGrid.defineMap("SomeMap");
```

```
//Définissez les propriétés sur la base de 50 000 entrées de mappe
LRUEvictor someEvictor = new LRUEvictor();
someEvictor.setNumberOfLRUQueues(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

Notez que seulement deux lignes sont différentes de l'exemple LFUEvictor.

Meilleures pratiques pour les performances de verrouillage

Les stratégies de verrouillage et les paramètres d'isolement de transactions affectent les performances de vos applications.

Extraction d'une instance mise en cache

Pour plus d'informations sur le verrouillage des entrées de mappe, consultez le manuel *Guide d'administration*.

Stratégie de verrouillage pessimiste

Recourez à la stratégie de verrouillage pessimiste pour les opérations de mappe en lecture et en écriture pour lesquelles les clés entrent généralement en conflit. La stratégie de verrouillage pessimiste a une incidence considérable sur les performances.

Isolement de transactions lecture validée et lecture non validée

Lorsque vous utilisez la stratégie de verrouillage pessimiste, définissez le niveau d'isolement de transaction à l'aide de la méthode Session.setTransactionIsolation. Pour l'isolement de transactions lecture validée et lecture non validée, utilisez l'argument Session.TRANSACTION_READ_COMMITTED ou Session.TRANSACTION_READ_UNCOMMITTED en fonction de l'isolement. Pour rétablir le comportement de verrouillage pessimiste par défaut pour le niveau d'isolement de transaction, faites appel à la méthode Session.setTransactionIsolation avec l'argument Session.REPEATABLE_READ.

L'isolement lecture validée réduit la durée des verrous partagés, ce qui peut améliorer l'accès simultané et limiter le risque d'interblocage. Ce niveau d'isolement doit être utilisé lorsqu'une transaction ne doit pas garantir l'invariabilité des valeurs de lecture pendant la durée de la transaction.

Utilisez la valeur lecture non validée lorsque la transaction ne doit pas tenir compte les données validées.

Stratégie de verrouillage optimiste

Il s'agit de la configuration par défaut. Cette stratégie améliore les performances et l'évolutivité par rapport à la stratégie pessimiste. Utilisez cette stratégie lorsque vos applications peuvent tolérer certains échecs de la mise à jour optimiste tout en offrant cependant de meilleures performances que la stratégie pessimiste. Cette stratégie s'adapte particulièrement aux opérations de lecture et aux applications de mise à jour exceptionnelles.

Plug-in OptimisticCallback

La stratégie de verrouillage optimiste effectue une copie des entrées de cache et les compare si nécessaire. Cette opération peut être coûteuse car la copie des entrées risque d'entraîner des tâches de clonage ou de sérialisation. Pour augmenter les performances le plus rapidement possible, implémentez le plug-in personnalisé pour les mappes de non-entité.

Pour plus d'informations, voir. Pour plus d'informations sur le plug-in OptimisticCallback, consultez le manuel *Présentation du produit*.

Utilisation de champs version pour les entités

Lorsque vous utilisez le verrouillage optimiste avec les entités, recourez à l'annotation @Version ou l'attribut équivalent dans le fichier descripteur de métadonnées Entité . L'annotation de version permet à l'ObjectGrid de suivre de façon efficace la version d'un objet. Si l'entité ne comporte pas de champ version et si le verrouillage optimiste est utilisé pour l'entité, l'entité entière doit être copiée et comparée.

Stratégie sans verrouillage

Utilisez la stratégie sans verrouillage pour les applications en lecture seule. Cette stratégie ne déclenche aucun verrouillage et n'utilise aucun gestionnaire de verrous. Par conséquent, elle offre plus d'accès simultanés, de performances et d'évolutivité.

Performances de sérialisation

WebSphere eXtreme Scale utilise plusieurs processus Java pour héberger les données. Ces processus sérialisent les données, c'est-à-dire les convertissent (sous la forme d'instances d'objets Java) en octets, puis si nécessaire de nouveau en objets pour permettre le déplacement des données entre les processus client et serveur. La conversion des paramètres de données est l'opération la plus coûteuse et doit être effectuée par le développeur d'applications lors de la conception du schéma, la configuration de la grille et l'interaction avec les API d'accès aux données.

La sérialisation Java par défaut et les routines de copie sont relativement lentes et peuvent consommer 60 à 70 % de la capacité du processeur dans une configuration classique. Les sections ci-dessous présentent des options d'amélioration des performances de la sérialisation.

Ecriture d'un ObjectTransformer pour chaque mappe de sauvegarde

Un ObjectTransformer peut être associé à une mappe de sauvegarde. Votre application peut comporter une classe qui implémente l'interface ObjectTransformer et offre des implémentations pour les opérations suivantes :

- Copie des valeurs
- Sérialisation et inflation des clés vers et à partir des flux
- Sérialisation et inflation des valeurs vers et à partir des flux

L'application ne doit pas copier des clés car celles-ci sont considérées comme étant non modifiables.

Pour plus d'informations, voir Plug-in de sérialisation et de copie d'objets mis en cache et Pratiques recommandées pour l'interface ObjectTransformer.

Remarque : L'interface ObjectTransformer est uniquement appelée lorsque la grille d'objets connaît les données en cours de transformation. Par exemple, les agents de l'API DataGrid sont utilisés, les agents et les données d'instance des agents ou les données renvoyées par l'agent doivent être optimisées à l'aide de techniques de sérialisation personnalisées. L'interface ObjectTransformer n'est pas appelée pour les agent de l'API DataGrid.

Utilisation d'entités

Lors de l'utilisation de l'API EntityManager avec les entités, la grille ne stocke pas les objets d'entité directement dans les mappes de sauvegarde. L'API EntityManager convertit l'objet d'entité en objets de bloc de données. Pour plus d'informations, voir Pour plus d'informations, consultez la rubrique relative à l'utilisation d'un chargeur avec les mappes d'entité et les blocs de données dans le *Guide de programmation*. Les mappes d'entité sont automatiquement associées à un ObjectTransformer hautement optimisé. Dès que l'API ObjectMap ou EntityManager est utilisée pour interagir avec les mappes d'entité, l'entité ObjectTransformer est appelée.

Sérialisation personnalisée

Dans certains cas, les objets doivent être modifiés de façon à utiliser la sérialisation personnalisée, telle que l'implémentation de l'interface java.io.Externalizable ou l'implémentation des méthodes writeObject et readObject pour les classes qui mettent en oeuvre l'interface java.io.Serializable. Les techniques de sérialisation personnalisée doivent être employées lorsque les objets sont sérialisés à l'aide de mécanismes autres que les méthodes de l'API ObjectGrid ou EntityManager.

Par exemple, lorsque les objets ou entités sont stockés en tant que données d'instance dans un agent d'API DataGrid ou lorsque l'agent renvoie des objets ou des entités, ces objets ne sont transformés à l'aide d'un ObjectTransformer. Toutefois, l'agent fait automatiquement appel à l'ObjectTransformer lors de l'utilisation de l'interface `EntityMixin`. Pour plus de détails, voir Agents DataGrid et mappes basées sur les entités.

Tableaux d'octets

Lors de l'utilisation des API ObjectMap ou DataGrid, les objets de clé et de valeur sont sérialisés dès que le client interagit avec la grille et lorsque les objets sont répliqués. Pour limiter les frais liés à la sérialisation, utilisez des tableaux d'octets et non les objets Java. Le stockage en mémoire des tableaux d'octets revient nettement moins cher car le kit JDK doit rechercher moins d'objets lors de la récupération de place et les tableaux d'octets peuvent être agrandis à la demande. Les tableaux d'octets doivent être uniquement utilisés si vous ne devez pas accéder aux objets utilisant des requêtes ou des index. Les données étant stockées sous la forme d'octets, leur accès n'est autorisé qu'avec la clé correspondante.

WebSphere eXtreme Scale peut automatiquement stocker les données sous la forme de tableaux d'octets à l'aide l'option de configuration de mappes CopyMode.COPY_TO_BYTES ou ce mode de stockage peut être traité manuellement par le client. Cette option permet de stocker les données en mémoire de façon efficace et peut, à la demande, agrandir automatiquement les objets au sein du tableau d'octets pour une utilisation des requêtes et des index.

Meilleures pratiques concernant l'interface ObjectTransformer

L'interface ObjectTransformer envoie les rappels à l'application pour permettre l'implémentation personnalisée d'opérations courantes et coûteuses telles de la sérialisation ou la copie complète sur des objets.

Présentation

Pour des informations détaillées sur l'interface ObjectTransformer, voir «Plug-in ObjectTransformer», à la page 227. Du point de vue des performances et à partir des informations sur la méthode CopyMode (sous la rubrique des meilleures pratiques concernant la méthode CopyMode), eXtreme Scale copie clairement les valeurs dans tous les cas, sauf lorsque le mode NO_COPY est utilisé. Le mécanisme de copie par défaut utilisé dans eXtreme Scale est la sérialisation, qui est connue pour être une opération coûteuse. L'interface ObjectTransformer est utilisée dans cette situation. L'interface ObjectTransformer envoie des rappels à l'application pour permettre l'implémentation personnalisée d'opérations courantes et coûteuses, telles que la sérialisation d'objets et la copie complète sur des objets.

Une application permet l'implémentation de l'interface ObjectTransformer sur une mappe. eXtreme Scale délègue ensuite aux méthodes de cet objet et dépend de l'application pour fournir une version optimisée de chaque méthode de l'interface. Voici l'interface ObjectTransformer :

```
public interface ObjectTransformer {
    void serializeKey(Object key, ObjectOutputStream stream) throws IOException;
    void serializeValue(Object value, ObjectOutputStream stream) throws IOException;
    Object inflateKey(ObjectInputStream stream) throws IOException, ClassNotFoundException;
    Object inflateValue(ObjectInputStream stream) throws IOException, ClassNotFoundException;
    Object copyValue(Object value);
    Object copyKey(Object key);
}
```

Vous pouvez associer l'interface ObjectTransformer à une BackingMap en utilisant le code de l'exemple suivant :

```
ObjectGrid g = ...;
BackingMap bm = g.defineMap("PERSON");
MyObjectTransformer ot = new MyObjectTransformer();
bm.setObjectTransformer(ot);
```

Ajustement de la sérialisation d'objets et inflation

La sérialisation d'objets est généralement l'opération la plus coûteuse en termes de performances, de même qu'eXtreme Scale, qui utilise le mécanisme de sérialisation par défaut si un plug-in ObjectTransformer n'est pas fourni par l'application. Soit l'application fournit des implémentations des objets sérialisables readObject et writeObject, soit les objets implémentent l'interface externalisable, ce qui est environ dix fois plus rapide. Si les objets dans la mappe ne peuvent pas être vérifiés, une application peut associer une interface ObjectTransformer avec ObjectMap. Les méthodes de sérialisation et d'inflation sont proposées pour permettre à l'application de fournir un code personnalisé permettant d'optimiser ces opérations, compte tenu de leur impact important sur les performances du système. La méthode de sérialisation permet de sérialiser l'objet sur le flux fourni. La méthode d'inflation fournit le flux d'entrée et attend que l'application crée l'objet, augmente la taille de ce dernier en utilisant les données du flux et le renvoie. Les implémentations des méthodes de sérialisation et d'inflation doivent être symétriques.

Ajustement des opérations de copie complète

Après qu'une application a reçu un objet d'une ObjectMap, eXtreme Scale effectue une copie complète de la valeur de l'objet afin de s'assurer que la copie de la mappe BaseMap maintient l'intégrité des données. L'application peut ensuite modifier la valeur de l'objet en toute sécurité. Lorsque la transaction est validée, la copie de la valeur de l'objet dans la mappe BaseMap est mise à jour avec la nouvelle valeur modifiée et l'application arrête d'utiliser la valeur. Vous pouvez copier de nouveau l'objet au moment de la validation afin de disposer d'une copie privée. Cependant, dans ce cas, le coût des performances de cette action est compensé par le fait que le programmeur de l'application ne peut pas utiliser cette valeur une fois la transaction validée. L'ObjectTransformer par défaut tente d'utiliser soit un clone soit une paire sérialisation-inflation pour générer une copie. La paire sérialisation-inflation est le pire scénario imaginable en termes de performances. Si le profilage révèle que la sérialisation et l'inflation présentent un problème pour votre application, écrivez une méthode de clonage appropriée pour créer une copie complète. Si vous ne pouvez pas modifier la classe, créez un plug-in ObjectTransformer personnalisé et implémentez des méthodes copyValue et copyKey plus efficaces.

Chapitre 11. Résolution des incidents

En plus des journaux et de la trace ainsi que des des messages et des notes sur l'édition, mentionnés dans la présente section, vous pouvez utiliser les outils de surveillance pour identifier et résoudre les incidents liés à l'emplacement des données dans l'environnement, la disponibilité des serveurs dans la grille, etc. Si vous utilisez un environnement WebSphere Application Server, vous pouvez utiliser PMI (Performance Monitoring Infrastructure). Si vous utilisez un environnement autonome, vous pouvez utiliser l'outil de surveillance d'un fournisseur, tel que CA Wily Introscope ou Hyperic HQ. Vous pouvez également utiliser et personnaliser l'exemple d'utilitaire xsAdmin pour afficher des informations textuelles sur votre environnement.

Journaux et trace

Vous pouvez utiliser des journaux et une trace pour surveiller votre environnement et en identifier et résoudre les problèmes. Les journaux se trouvent dans des emplacements différents suivant votre configuration. Il se peut que vous deviez fournir une trace pour un serveur lorsque vous travaillez avec le service d'assistance d'IBM.

Journaux avec WebSphere Application Server

Pour plus d'informations, voir le centre de documentation de WebSphere Application Server.

Journaux avec WebSphere eXtreme Scale dans un environnement autonome

Avec les serveurs de catalogues et de conteneurs autonomes, vous définissez l'emplacement des journaux et des spécifications de trace. Les journaux des serveurs de catalogues se trouvent dans l'emplacement où vous avez exécuté la commande de démarrage des serveurs.

Définition de l'emplacement des journaux des serveurs conteneurs

Par défaut, les journaux d'un conteneur se trouvent dans le répertoire où la commande serveur a été exécutée. Si vous démarrez les serveurs dans le répertoire *<rép_base_eXtremeScale>*/bin, les journaux et les fichiers de trace se trouvent dans les répertoires logs/*<nom_serveur>* du répertoire bin. Pour spécifier un autre emplacement pour les journaux des serveurs conteneurs, créez un fichier de propriétés, tel que le fichier server.properties, avec le contenu suivant :

```
workingDirectory=<directory>
traceSpec=
systemStreamToFileEnabled=true
```

La propriété workingDirectory correspond au répertoire racine des journaux et du fichier de trace facultatif. WebSphere eXtreme Scale crée un répertoire du nom du serveur conteneur avec un fichier SystemOut.log, un fichier SystemErr.log et un fichier de trace si la trace a été activée avec l'option traceSpec. Pour utiliser un fichier de propriétés au démarrage des conteneurs, utilisez l'option **-serverProps** et spécifiez l'emplacement du fichier de propriétés du serveur.

Les messages d'information courants à rechercher dans le fichier SystemOut.log sont les messages de confirmation du démarrage. Pour plus d'informations sur un message spécifique, voir «Messages», à la page 390.

Trace avec WebSphere Application Server

Pour plus d'informations, voir le centre de documentation de WebSphere Application Server.

Trace sur le service de catalogue

Vous pouvez définir la trace sur un service de catalogue à l'aide des paramètres **-traceSpec** et **-traceFile** au démarrage du service de catalogue. Par exemple :

```
startOgServer.sh catalogServer -traceSpec
ObjectGridPlacement=all=enabled -traceFile
/home/user1/logs/trace.log
```

Si vous démarrez le service de catalogue dans le répertoire *<rép_base_eXtremeScale>*/bin, les journaux et les fichiers de trace se trouveront dans le répertoire logs/*<nom_service_catalogue>* du répertoire bin. Consultez les informations sur le démarrage du processus du service de catalogue dans un environnement autonome, dans le *Guide d'administration*.

Trace sur un serveur de conteneur autonome

Vous pouvez activer la trace sur un serveur de conteneur de deux manières. Vous pouvez créer un fichier de propriétés de serveur comme expliqué dans la section des journaux ou activer la trace à l'aide de la ligne de commande, au démarrage. Pour activer la trace du conteneur avec un fichier de propriétés de serveur, mettez à jour la propriété **traceSpec** avec la spécification de trace requise. Pour activer la trace du conteneur à l'aide de paramètres de démarrage, utilisez les paramètres **-traceSpec** et **-traceFile**. Par exemple :

```
startOgServer.sh c0 -objectGridFile ../xml/myObjectGrid.xml
-deploymentPolicyFile ../xml/myDepPolicy.xml -catalogServiceEndpoints
server1.rchland.ibm.com:2809 -traceSpec
ObjectGridPlacement=all=enabled -traceFile /home/user1/logs/trace.log
```

Si vous démarrez le serveur dans le répertoire *<rép_base_eXtremeScale>*/bin, les journaux et les fichiers de trace se trouvent dans les répertoires logs/*<nom_serveur>* du répertoire bin

Pour plus d'informations, voir

Trace avec l'interface ObjectGridManager

Une autre option consiste à définir la trace lors de la phase d'exécution sur une interface ObjectGridManager. La définition de la sur une interface ObjectGridManager permet d'extraire la trace sur un client eXtreme Scale lorsqu'il se connecte à eXtreme Scale et valide des transactions. Pour définir la trace sur une interface ObjectGridManager, fournissez une spécification de trace et un journal de trace.

```
ObjectGridManager manager = ObjectGridManagerFactory.getObjectGridManager();
...
manager.setTraceEnabled(true);
manager.setTraceFileName("logs/myClient.log");
manager.setTraceSpecification("ObjectGridReplication=all=enabled");
```

Activation de la trace à l'aide de l'utilitaire xsadmin

Pour activer la trace à l'aide de l'utilitaire xsadmin, utilisez l'option **setTraceSpec**. Utilisez l'utilitaire xsadmin pour activer la trace sur un environnement autonome lors de la phase d'exécution et non au démarrage. Vous pouvez activer la trace sur tous les serveurs et services de catalogue ou filtrer les serveurs en fonction du nom ObjectGrid, etc. Par exemple, pour activer la trace ObjectGridReplication avec accès au serveur du service de catalogue, exécutez :

```
<base_eXtremeScale>/bin>xsadmin.bat -setTraceSpec "ObjectGridReplication=all=enabled"
```

Vous pouvez également désactiver la trace en affectant à la spécification de trace la valeur `*=all=disabled.`

Pour plus d'informations, voir les informations sur l'utilitaire xsAdmin dans le *Guide d'administration*.

Fichiers et répertoires ffdc

Les fichiers FFDC sont destinés au support technique d'IBM, pour le débogage. Ces fichiers peuvent être demandés par le support technique d'IBM, en cas de problème.

Ces fichiers se trouvent dans un répertoire libellé `ffdc` et contiennent des fichiers similaires au suivant :

```
server2_exception.log
server2_20802080_07.03.05_10.52.18_0.txt
```

Options de trace

Vous pouvez activer la trace pour fournir des informations sur votre environnement au service d'assistance IBM.

A propos de la trace

La trace de WebSphere eXtreme Scale est divisée en plusieurs composants. Comme pour la trace WebSphere Application Server, vous pouvez spécifier le niveau de trace à utiliser. Les niveaux de trace courants sont les suivants : all, debug, entryExit et event.

Voici un exemple de chaîne de trace :

```
ObjectGridComponent=level=enabled
```

Vous pouvez concaténer les chaînes de trace. Utilisez le symbole `*` (astérisque) pour spécifier une valeur générique, telle que `ObjectGrid*=all=enabled`. Si vous devez fournir une trace au service d'assistance IBM, une chaîne de trace spécifique est demandée. Par exemple, en cas de problème de réplication, la trace `ObjectGridReplication=debug=enabled` peut être demandée.

Spécification de la trace

ObjectGrid
Moteur général du cache central.

ObjectGridCatalogServer
Service de catalogue général.

ObjectGridChannel
Communications statiques de la topologie de déploiement.

7.1+ **ObjectGridClientInfo**
Informations sur le client DB2.

7.1+ **ObjectGridClientInfoUser**
Informations sur l'utilisateur DB2.

ObjectgridCORBA
Communications dynamiques de la topologie de déploiement.

ObjectGridDataGrid
API AgentManager.

ObjectGridDynaCache
Fournisseur de cache dynamique de WebSphere eXtreme Scale.

ObjectGridEntityManager
API EntityManager. A utiliser avec l'option Projector.

ObjectGridEvictors
Expulseurs pré-intégrés d'ObjectGrid.

ObjectGridJPA
Chargeurs JPA (Java Persistence API).

ObjectGridJPACache
Plug-in de cache JPA.

ObjectGridLocking
Gestionnaire de verrouillage des entrées de cache d'ObjectGrid.

ObjectGridMBean
Beans de gestion.

7.1+ **ObjectGridMonitor**
Infrastructure de la surveillance de l'historique.

ObjectGridPlacement
Service de positionnement des fragments de serveur de catalogues.

ObjectGridQuery
Requête ObjectGrid.

ObjectGridReplication
Service de réplication.

ObjectGridRouting
Détails du routage client/serveur.

ObjectGridSecurity
Trace de la sécurité.

ObjectGridStats
Statistiques d'ObjectGrid.

ObjectGridStreamQuery
API de la requête de flux.

ObjectGridWriteBehind
Ecriture différée d'ObjectGrid.

Projector
Moteur dans l'API EntityManager.

QueryEngine
Moteur de requête des API de requête Object et EntityManager.

QueryEnginePlan
Diagnostiques du plan de requête.

IBM Support Assistant for WebSphere eXtreme Scale

IBM Support Assistant permet de collecter des données, d'analyser des symptômes et d'accéder à des informations sur les produits.

IBM Support Assistant Lite

IBM Support Assistant Lite for WebSphere eXtreme Scale assure une collecte automatique des données et l'analyse des symptômes pour l'identification des problèmes et de leurs causes.

IBM Support Assistant Lite réduit le temps consacré à la reproduction des problèmes en adaptant son ensemble de niveaux de traçabilité (niveaux de fiabilité, de disponibilité et de facilité de maintenance, qui sont définis automatiquement par l'outil) afin de simplifier l'identification des problèmes. Mais si cette assistance ne suffit pas et que vous ayez besoin de l'aide d'un technicien, IBM Support Assistant Lite réduit également le temps consacré à l'envoi des informations appropriées au support technique d'IBM.

IBM Support Assistant Lite est inclus dans chaque installation de WebSphere eXtreme Scale version 7.1.0

IBM Support Assistant

IBM® Support Assistant (ISA) permet d'accéder rapidement à des ressources de produits, de formation et de support qui pourront vous aider à répondre de vous-mêmes à vos questions et à résoudre les problèmes rencontrés avec des logiciels IBM sans avoir besoin de contacter le support IBM. Différents plug-in spécifiques à différents produits vous permettent de personnaliser IBM Support Assistant en fonction des produits particuliers que vous avez installés. IBM Support Assistant peut également collecter des données système, des fichiers journaux et d'autres informations qui aideront le support technique d'IBM à déterminer la cause des problèmes.

IBM Support Assistant est un utilitaire qui s'installe sur le poste de travail et non sur le serveur WebSphere eXtreme Scale lui-même. En effet, sa mémoire et ses besoins en ressources risqueraient d'affecter de manière négative les performances du serveur WebSphere eXtreme Scale. Les composants portables de diagnostics qui sont inclus dans l'Assistant sont conçus pour avoir un impact minimal sur le fonctionnement normal d'un serveur.

IBM Support Assistant est utilisable des manières suivantes :

- pour effectuer des recherches dans des sources IBM et non IBM de connaissances et d'information sur plusieurs produits IBM afin de répondre à une question ou de résoudre un problème
- pour trouver des informations complémentaires dans des ressources Web dédiées à un produit donné (pages d'accueil du produit et de son support, groupes de discussions et forums d'utilisateurs, ressources d'acquisition de compétences et de formations, informations de résolution des problèmes et FAQ)
- pour renforcer vos capacités à diagnostiquer les problèmes d'un produit donné grâce aux outils de diagnostics ciblés proposés par l'Assistant

- pour simplifier la collecte des données de diagnostic afin de vous aider, IBM et vous, à résoudre vos problèmes (collecte de données générales ou liées à un symptôme particulier)
- pour vous aider à signaler des problèmes au support IBM via une interface personnalisée en ligne avec possibilité d'attacher aux incidents signalés les données de diagnostic mentionnées plus haut ou toute autre information

Enfin, la fonctionnalité Updater intégrée permet de mettre à jour le support pour d'autres produits logiciels et d'autres fonctionnalités au fur et à mesure de leur disponibilité. Pour configurer IBM Support Assistant afin de l'utiliser avec WebSphere eXtreme Scale, commencez par l'installer à l'aide des fichiers fournis dans l'image téléchargée à partir de la page Web IBM Support Overview (http://www-947.ibm.com/support/entry/portal/Overview/Software/Other_Software/IBM_Support_Assistant. Ensuite, utilisez IBM Support Assistant pour repérer et installer les mises à jour de produits qui vous intéressent. Vous pouvez également choisir d'installer des plug-in pour d'autres logiciels IBM de votre environnement. Vous trouverez des informations complémentaires et la dernière version d'IBM Support Assistant à la page Web IBM Support Assistant (http://www.ibm.com/software/support/isa/).

Messages

Lorsqu'un message apparaît dans le journal ou d'autres parties de l'interface du produit, vous pouvez le rechercher en fonction de son préfixe de composant pour obtenir de plus amples informations.

Recherche de messages

Lorsque vous rencontrez un message dans un journal, copiez le numéro du message avec sa lettre préfixe et son numéro et effectuez une recherche dans le Centre de documentation (par exemple, `CWOBJ1526I`). Lorsque vous recherchez le message, vous pouvez trouver une explication supplémentaire du message et les éventuelles actions à effectuer pour résoudre le problème.

Pour l'index des messages du produit, reportez-vous au Centre de documentation.

Notes sur l'édition

Des liens permettent d'accéder au site Web de support technique, à la documentation, aux dernières mises à jour, aux restrictions et aux incidents recensés du produit.

- «Dernières mises à jour, limitations et problèmes connus»
- «Accès à la configuration système et logicielle requise», à la page 391
- «Accès à la documentation du produit», à la page 391
- «Accès au site de support technique du produit», à la page 391
- «Contacter le service de support logiciel IBM», à la page 391

Dernières mises à jour, limitations et problèmes connus

Les notes sur l'édition sont disponibles sur le site de support technique du produit sous forme de notes techniques. Pour afficher une liste de toutes les notes techniques de WebSphere eXtreme Scale, accédez à la page Web du support

technique. Cliquer sur les liens indiqués ici générera une recherche dans la page Web du support des notes sur l'édition correspondantes, lesquelles seront retournées sous forme de liste.

- 7.1+ Pour voir la liste des notes sur l'édition de la version 7.1, allez à la page Web du support.
- Pour voir la liste des notes sur l'édition de la version 7.0, allez à la page Web du support.
- Pour voir la liste des notes sur l'édition de la version 6.1, allez au wiki des notes sur l'édition.

Accès à la configuration système et logicielle requise

La configuration matérielle et logicielle requise est détaillée dans les pages suivantes :

- Configuration requise détaillée

Accès à la documentation du produit

Pour accéder à l'ensemble des informations, accédez à la page Bibliothèque.

Accès au site de support technique du produit

Pour rechercher les informations de support technique et notamment les dernières notes techniques, les fichiers à télécharger et les correctifs, accédez à la page du support technique.

Contacter le service de support logiciel IBM

Si un incident survient lors de l'utilisation du produit, essayez tout d'abord d'effectuer les opérations suivantes :

- Suivez les étapes décrites dans la documentation du produit
- Recherchez la documentation connexe dans l'aide en ligne
- Recherchez les messages d'erreur dans le document de référence des messages

Si vous ne parvenez pas à résoudre l'erreur à l'aide d'une des méthodes précédentes, prenez contact avec le support technique IBM.

Remarques

Les références aux produits, logiciels et services d'IBM n'impliquent pas qu'ils soient distribués dans tous les pays dans lesquels IBM exerce son activité. Toute référence à un produit, logiciel ou service IBM n'implique pas que seul ce produit, logiciel ou service puisse être utilisé. Tout autre élément fonctionnellement équivalent peut être utilisé, s'il n'enfreint aucun droit d'IBM. L'évaluation et la vérification de son fonctionnement en conjonction avec d'autres produits, hormis ceux expressément désignés par IBM, relèvent de la responsabilité de l'utilisateur.

IBM peut détenir des brevets ou des demandes de brevet couvrant les produits mentionnés dans le présent document. La remise de ce document ne vous donne aucun droit de licence sur ces brevets ou demandes de brevet. Si vous désirez recevoir des informations concernant l'acquisition de licences, veuillez en faire la demande par écrit à l'adresse suivante :

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, New York 10594 USA

Les licenciés souhaitant obtenir des informations permettant : (i) l'échange des données entre des logiciels créés de façon indépendante et d'autres logiciels (dont celui-ci), et (ii) l'utilisation mutuelle des données ainsi échangées, doivent adresser leur demande à :

IBM Corporation
Mail Station P300
522 South Road
Poughkeepsie, NY 12601-5400
USA
Attention: Information Requests

Ces informations peuvent être soumises à des conditions particulières, prévoyant notamment le paiement d'une redevance.

Marques

Les termes suivant sont des marques d'IBM Corporation aux Etats-Unis et/ou dans certains autres pays :

- AIX
- CICS
- Cloudscape
- DB2
- Domino
- IBM
- Lotus
- RACF
- Redbooks
- Tivoli
- WebSphere
- z/OS

Java ainsi que tous les logos et toutes les marques incluant Java sont des marques de Sun Microsystems, Inc. aux Etats-Unis et/ou dans certains autres pays.

LINUX est une marque de Linus Torvalds aux Etats-Unis et/ou dans certains autres pays.

Microsoft, Windows®, Windows NT® et le logo Windows sont des marques de Microsoft Corporation aux Etats-Unis et/ou dans certains autres pays.

UNIX® est une marque enregistrée de l'Open Group aux Etats-Unis et dans d'autres pays.

Les autres noms de sociétés, de produits et de services peuvent appartenir à des tiers.

Index

P

R

S

T

V

W