IBM WebSphere eXtreme Scale Version 7.1.1
Version 7.1

Guide de programmation

21 novembre 2011

Remarque
Certaines illustrations de ce manuel ne sont pas disponibles en français à la date d'édition.

décembre 2011

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

Avis aux lecteurs canadiens

Le présent document a été traduit en France. Voici les principales différences et particularités dont vous devez tenir compte.

Illustrations

Les illustrations sont fournies à titre d'exemple. Certaines peuvent contenir des données propres à la France.

Terminologie

La terminologie des titres IBM peut différer d'un pays à l'autre. Reportez-vous au tableau ci-dessous, au besoin.

IBM France	IBM Canada
ingénieur commercial	représentant
agence commerciale	succursale
ingénieur technico-commercial	informaticien
inspecteur	technicien du matériel

Claviers

Les lettres sont disposées différemment : le clavier français est de type AZERTY, et le clavier français-canadien de type QWERTY.

OS/2 et Windows - Paramètres canadiens

Au Canada, on utilise :

- les pages de codes 850 (multilingue) et 863 (français-canadien),
- le code pays 002,
- le code clavier CF.

Nomenclature

Les touches présentées dans le tableau d'équivalence suivant sont libellées différemment selon qu'il s'agit du clavier de la France, du clavier du Canada ou du clavier des États-Unis. Reportez-vous à ce tableau pour faire correspondre les touches françaises figurant dans le présent document aux touches de votre clavier.

France	Canada	Etats-Unis
(Pos1)		Home
Fin	Fin	End
(PgAr)		PgUp
(PgAv)		PgDn
Inser	Inser	Ins
Suppr	Suppr	Del
Echap	Echap	Esc
Attn	Intrp	Break
Impr écran	ImpEc	PrtSc
Verr num	Num	Num Lock
Arrêt défil	Défil	Scroll Lock
(Verr maj)	FixMaj	Caps Lock
AltGr	AltCar	Alt (à droite)

Brevets

Il est possible qu'IBM détienne des brevets ou qu'elle ait déposé des demandes de brevets portant sur certains sujets abordés dans ce document. Le fait qu'IBM vous fournisse le présent document ne signifie pas qu'elle vous accorde un permis d'utilisation de ces brevets. Vous pouvez envoyer, par écrit, vos demandes de renseignements relatives aux permis d'utilisation au directeur général des relations commerciales d'IBM, 3600 Steeles Avenue East, Markham, Ontario, L3R 9Z7.

Assistance téléphonique

Si vous avez besoin d'assistance ou si vous voulez commander du matériel, des logiciels et des publications IBM, contactez IBM direct au 1 800 465-1234.

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 :

- La *Présentation du produit* contient une vue de haut niveau des concepts de WebSphere eXtreme Scale, avec des scénarios d'utilisation et des tutoriels.
- Le *Guide d'installation* explique comment installer les topologies communes de WebSphere eXtreme Scale.
- 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.

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

Vous pouvez également accéder aux mêmes informations dans cette bibliothèque dans le centre de documentation de la version 7.1.1 WebSphere eXtreme Scale.

Utilisation hors ligne des manuels

Tous les manuels de la bibliothèque WebSphere eXtreme Scale contiennent des liens vers le centre de documentation, avec l'URL racine `http://publib.boulder.ibm.com/infocenter/wxsinfo/v7r1m1`. Ces liens vous permettent d'accéder directement aux informations associées. Toutefois, si vous travaillez hors ligne et rencontrez l'une de ces liens, vous pouvez rechercher le titre du lien dans les autres manuels dans la bibliothèque. La documentation d'API, le glossaire et les références des messages ne sont pas disponibles dans les manuels PDF.

A qui s'adresse ce document

Ce document est principalement destiné aux développeurs d'applications.

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. Tutoriels

Vous pouvez utiliser les tutoriels pour mieux comprendre les scénarios d'utilisation du produit, y compris le gestionnaire d'entités, les requêtes et la sécurité.

Tutoriel : interrogation d'une grille de données en mémoire locale

Vous pouvez développer un ObjectGrid interne local qui peut stocker des informations de commande pour un site Web et montrer comment utiliser l'API ObjectQuery pour interroger la grille de données.

Avant de commencer

Vérifiez que le fichier `objectgrid.jar` se trouve bien dans le chemin d'accès aux classes.

Pourquoi et quand exécuter cette tâche

Chaque étape du tutoriel repose sur l'étape précédente. Suivez chacune des étapes pour générer une application Java Platform, Standard Edition Version 5 (ou ultérieure) simple qui utilise une grille de données locale interne.

Tutoriel ObjectQuery - Etape 1

A l'aide de la procédure ci-après, vous pouvez continuer à développer un ObjectGrid local en mémoire qui stocke les informations sur les commandes d'un magasin en ligne à l'aide des API ObjectMap. Vous définissez un schéma pour la mappe et exécutez une requête sur cette dernière.

Procédure

1. Créez un ObjectGrid avec un schéma de mappe.

   Créez un ObjectGrid avec un schéma de mappe pour la mappe, puis insérez un objet dans le cache et extrayez-le ensuite à l'aide d'une simple requête.

   OrderBean.java

   ```
   public class OrderBean implements Serializable {
       String orderNumber;
       java.util.Date date;
       String customerName;
       String itemName;
       int quantity;
       double price;
   }
   ```

2. Définissez la clé primaire.

   Le code précédent affiche un objet OrderBean. Cet objet implémente l'interface java.io.Serializable car tous les objets du cache doivent (par défaut) être sérialisables.

   L'attribut orderNumber est la clé primaire de l'objet. L'exemple de programme ci-après peut être exécuté en mode autonome. Vous devez suivre ce tutoriel dans un projet Java Eclipse dont le fichier `objectgrid.jar` est ajouté au chemin d'accès aux classes.

Application.java

```
package querytutorial.basic.step1;

import java.util.Iterator;

import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.config.QueryConfig;
import com.ibm.websphere.objectgrid.config.QueryMapping;
import com.ibm.websphere.objectgrid.query.ObjectQuery;

public class Application
{

    static public void main(String [] args) throws Exception
    {
        ObjectGrid og = ObjectGridManagerFactory.getObjectGridManager().createObjectGrid();
        og.defineMap("Order");

        // Définissez le schéma
        QueryConfig queryCfg = new QueryConfig();
        queryCfg.addQueryMapping(new QueryMapping("Order", OrderBean.class.getName(),
     "orderNumber", QueryMapping.FIELD_ACCESS));
        og.setQueryConfig(queryCfg);

        Session s = og.getSession();
        ObjectMap orderMap = s.getMap("Order");

        s.begin();
        OrderBean o = new OrderBean();
        o.customerName = "John Smith";
        o.date = new java.util.Date(System.currentTimeMillis());
        o.itemName = "Widget";
        o.orderNumber = "1";
        o.price = 99.99;
        o.quantity = 1;
        orderMap.put(o.orderNumber, o);
        s.commit();

        s.begin();
        ObjectQuery query = s.createObjectQuery("SELECT o FROM Order o WHERE o.itemName='Widget'");
        Iterator result = query.getResultIterator();
        o = (OrderBean) result.next();
        System.out.println("Found order for customer: " + o.customerName);
        s.commit();
    }
}
```

Cette application eXtreme Scale commence par initialiser une instance ObjectGrid locale avec un nom généré automatiquement. Ensuite, l'application crée une mappe de sauvegarde et une configuration de requête qui définit le type Java associé à la mappe, le nom du champ qui sert de clé primaire pour la mappe et la manière d'accéder aux données dans l'objet. Vous obtenez ensuite une session pour extraire l'instance ObjectMap et insérez un objet OrderBean dans la mappe, dans une transaction.

Une fois que les données ont été validées dans le cache, vous pouvez utiliser ObjectQuery pour rechercher l'objet OrderBean à l'aide de l'un des champs persistants de la classe. Les champs persistants sont ceux qui ne possèdent pas le modificateur transitoire. Comme vous n'avez pas défini d'index sur la mappe de sauvegarde, ObjectQuery doit analyser chaque objet de la mappe à l'aide de la réflexion Java.

Que faire ensuite

«Tutoriel ObjectQuery - Etape 2» montre comment un index peut être utilisé pour optimiser la requête.

Tutoriel ObjectQuery - Etape 2

A l'aide de la procédure ci-après, vous pouvez continuer à créer une instance ObjectGrid avec une mappe et un index, ainsi qu'un schéma pour la mappe. Vous pouvez ensuite insérer un objet dans le cache et l'extraire ultérieurement à l'aide d'une simple requête.

Avant de commencer

Vous devez avoir effectué l'étape «Tutoriel ObjectQuery - Etape 1», à la page 1 avant de passer à cette étape du tutoriel.

Procédure

Schéma et index

Application.java

```
// Crêez un index
    HashIndex idx= new HashIndex();
    idx.setName("theItemName");
    idx.setAttributeName("itemName");
    idx.setRangeIndex(true);
    idx.setFieldAccessAttribute(true);
    orderBMap.addMapIndexPlugin(idx);
}
```

L'index doit être une instance de com.ibm.websphere.objectgrid.plugins.index.HashIndex avec les paramètres suivants :

- Le nom est arbitraire, mais il doit être unique pour une mappe de sauvegarde donnée.
- Le nom d'attribut correspond au nom du champ ou à la propriété de bean que le moteur d'indexation utilise pour introspecter la classe. En l'occurrence, il s'agit du nom du champ pour lequel vous créerez l'index.
- RangeIndex doit toujours avoir la valeur true.
- La valeur de FieldAccessAttribute doit correspondre à celle définie dans l'objet QueryMapping lors de la création du schéma de requête. Dans ce cas, l'objet Java est accessible directement par les champs.

Lorsqu'une requête exécute ces filtres sur le champ itemName, le moteur de requête utilise automatiquement l'index défini. Le recours à l'index permet à la requête de s'exécuter beaucoup plus vite sans qu'une analyse de la mappe soit nécessaire. L'étape suivante montre comment un index peut être utilisé pour optimiser la requête.
Etape suivante

Tutoriel ObjectQuery - Etape 3

L'étape ci-après permet de créer un ObjectGrid avec deux mappes et un schéma pour les mappes possédant une relation, puis d'insérer des objets dans le cache et de les extraire ultérieurement à l'aide d'une simple requête.

Avant de commencer

Assurez-vous d'avoir bien effectué l'étape «Tutoriel ObjectQuery - Etape 2», à la page 2 avant de passer à cette étape.

Pourquoi et quand exécuter cette tâche

Cet exemple contient deux mappes, chacune mappée à un seul type Java. La mappe Order contient des objets OrderBean et la mappe Customer, des objets CustomerBean.

Procédure

Définissez les mappes avec une relation.

OrderBean.java

```
public class OrderBean implements Serializable {
    String orderNumber;
    java.util.Date date;
    String customerId;
    String itemName;
    int quantity;
    double price;
}
```

OrderBean ne contient plus customerName. A la place, il contient customerId, qui correspond à la clé primaire de l'objet CustomerBean et de la mappe Customer.

CustomerBean.java

```
public class CustomerBean implements Serializable{
    private static final long serialVersionUID = 1L;
    String id;
    String firstName;
    String surname;
    String address;
    String phoneNumber;
}
```

La relation entre les deux types ou mappes est la suivante :

Application.java

```
public class Application
{

    static public void main(String [] args)
        throws Exception
    {
        ObjectGrid og = ObjectGridManagerFactory.getObjectGridManager().createObjectGrid();
        og.defineMap("Order");
        og.defineMap("Customer");

        // Définissez le schéma
        QueryConfig queryCfg = new QueryConfig();
        queryCfg.addQueryMapping(new QueryMapping(
            "Order", OrderBean.class.getName(), "orderNumber", QueryMapping.FIELD_ACCESS));
        queryCfg.addQueryMapping(new QueryMapping(
            "Customer", CustomerBean.class.getName(), "id", QueryMapping.FIELD_ACCESS));
        queryCfg.addQueryRelationship(new QueryRelationship(
             OrderBean.class.getName(), CustomerBean.class.getName(), "customerId", null));
        og.setQueryConfig(queryCfg);

        Session s = og.getSession();
        ObjectMap orderMap = s.getMap("Order");
        ObjectMap custMap = s.getMap("Customer");

        s.begin();
        CustomerBean cust = new CustomerBean();
        cust.address = "Main Street";
        cust.firstName = "John";
        cust.surname = "Smith";
        cust.id = "C001";
        cust.phoneNumber = "5555551212";
        custMap.insert(cust.id, cust);

        OrderBean o = new OrderBean();
        o.customerId = cust.id;
        o.date = new java.util.Date();
        o.itemName = "Widget";
        o.orderNumber = "1";
        o.price = 99.99;
        o.quantity = 1;
        orderMap.insert(o.orderNumber, o);
        s.commit();

        s.begin();
        ObjectQuery query = s.createObjectQuery(
```

```
            "SELECT c FROM Order o JOIN o.customerId as c WHERE o.itemName='Widget'");
        Iterator result = query.getResultIterator();
        cust = (CustomerBean) result.next();
        System.out.println("Found order for customer: " + cust.firstName + " " + cust.surname);
        s.commit();
    }
}
```

Le XML équivalent XML dans le descripteur de déploiement ObjectGrid est le suivant :

```
<?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">
      <backingMap name="Order"/>
      <backingMap name="Customer"/>

      <querySchema>
         <mapSchemas>
           <mapSchema
             mapName="Order"
             valueClass="com.mycompany.OrderBean"
             primaryKeyField="orderNumber"
             accessType="FIELD"/>
           <mapSchema
             mapName="Customer"
             valueClass="com.mycompany.CustomerBean"
             primaryKeyField="id"
             accessType="FIELD"/>
         </mapSchemas>
         <relationships>
           <relationship
             source="com.mycompany.OrderBean"
             target="com.mycompany.CustomerBean"
             relationField="customerId"/>
         </relationships>
      </querySchema>
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

Que faire ensuite

L'étape «Tutoriel ObjectQuery - Etape 4», développe l'étape actuelle en incluant des objets d'accès par champ et par propriété ainsi que des relations supplémentaires.

Tutoriel ObjectQuery - Etape 4

L'étape ci-après montre comment créer une instance ObjectGrid avec quatre mappes et un schéma pour les mappes possédant plusieurs relations unidirectionnelles et bidirectionnelles. Vous pouvez ensuite insérer des objets dans le cache et les extraire ultérieurement à l'aide de plusieurs requêtes.

Avant de commencer

Vérifiez que vous avez bien effectué l'étape «Tutoriel ObjectQuery - Etape 3», à la page 3 avant de passer à l'étape en cours.

Procédure

Relations à plusieurs mappes

OrderBean.java

```
public class OrderBean implements Serializable {
    String orderNumber;
    java.util.Date date;
    String customerId;
    String itemName;
    int quantity;
    double price;
}
```

Comme dans l'étape précédente, OrderBean ne contient plus customerName. A la place, il contient customerId, qui correspond à la clé primaire de l'objet CustomerBean et de la mappe Customer.

CustomerBean.java

```
public class CustomerBean implements Serializable{
    private static final long serialVersionUID = 1L;
    String id;
    String firstName;
    String surname;
    String address;
    String phoneNumber;
}
```

Une fois que vous avez créé les classes spécifiées ci-dessus, vous pouvez exécuter l'application ci-après.

Application.java

```
public class Application
{

    static public void main(String [] args)
        throws Exception
    {
        ObjectGrid og = ObjectGridManagerFactory.getObjectGridManager().createObjectGrid();
        og.defineMap("Order");
        og.defineMap("Customer");

        // Define the schema
        QueryConfig queryCfg = new QueryConfig();
        queryCfg.addQueryMapping(new QueryMapping(
            "Order", OrderBean.class.getName(), "orderNumber", QueryMapping.FIELD_ACCESS));
        queryCfg.addQueryMapping(new QueryMapping(
            "Customer", CustomerBean.class.getName(), "id", QueryMapping.FIELD_ACCESS));
        queryCfg.addQueryRelationship(new QueryRelationship(
             OrderBean.class.getName(), CustomerBean.class.getName(), "customerId", null));
        og.setQueryConfig(queryCfg);

        Session s = og.getSession();
        ObjectMap orderMap = s.getMap("Order");
        ObjectMap custMap = s.getMap("Customer");

        s.begin();
        CustomerBean cust = new CustomerBean();
        cust.address = "Main Street";
        cust.firstName = "John";
        cust.surname = "Smith";
        cust.id = "C001";
        cust.phoneNumber = "5555551212";
        custMap.insert(cust.id, cust);

        OrderBean o = new OrderBean();
        o.customerId = cust.id;
        o.date = new java.util.Date();
        o.itemName = "Widget";
        o.orderNumber = "1";
        o.price = 99.99;
        o.quantity = 1;
        orderMap.insert(o.orderNumber, o);
        s.commit();

        s.begin();
        ObjectQuery query = s.createObjectQuery(
            "SELECT c FROM Order o JOIN o.customerId as c WHERE o.itemName='Widget'");
        Iterator result = query.getResultIterator();
```

```
        cust = (CustomerBean) result.next();
        System.out.println("Found order for customer: " + cust.firstName + " " + cust.surname);
        s.commit();
    }
}
```

L'utilisation de la configuration XML ci-après (dans le descripteur de déploiement ObjectGrid) permet d'obtenir les mêmes résultats que l'approche par programme ci-dessus.

```
<?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="og1">
      <backingMap name="Order"/>
      <backingMap name="Customer"/>

      <querySchema>
         <mapSchemas>
           <mapSchema
             mapName="Order"
             valueClass="com.mycompany.OrderBean"
             primaryKeyField="orderNumber"
             accessType="FIELD"/>
           <mapSchema
             mapName="Customer"
             valueClass="com.mycompany.CustomerBean"
             primaryKeyField="id"
             accessType="FIELD"/>
         </mapSchemas>
         <relationships>
           <relationship
             source="com.mycompany.OrderBean"
             target="com.mycompany.CustomerBean"
             relationField="customerId"/>
         </relationships>
      </querySchema>
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

Tutoriel : Stockage des informations de commande dans des entités

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.

Concepts associés:

«Mise en cache d'objets sans aucune relation impliquée (API ObjectMap)», à la page 155
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.

«Optimisation des performances de l'interface EntityManager», à la page 462
L'interface EntityManager sépare les applications de l'état conservé dans son magasin de données de grille de données de serveurs.

«Mise en cache d'objets et de leurs relations (API EntityManager)», à la page 166
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 dans WebSphere Application Server, entre autres, appliquent cette approche. Les API fondées sur les mappes connaissent toutefois des limitations. L'API EntityManager simplifie l'interaction avec la grille de données en facilitant la déclaration et l'interaction avec un graphique complexe d'objets associés.

«Gestionnaire d'entités dans un environnement distribué», à la page 179
Vous pouvez utiliser l'API EntityManager avec un ObjectGrid local ou dans un environnement distribué eXtreme Scale. 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.

«Interactions avec EntityManager», à la page 183
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.

«Stratégie FetchPlan de l'EntityManager», à la page 192
Un plan d'extraction (FetchPlan) est la stratégie utilisée par EntityManager pour extraire des objets associés si l'application doit accéder aux relations.

«Files d'attente des requêtes d'entité», à la page 196
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.

Référence associée:

«Agent d'instrumentation des performances d'entité», à la page 463
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.

«Définition d'un schéma d'entité», à la page 170
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.

«Programmes d'écoute d'entité et méthodes de rappel», à la page 186
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.

«Exemple de programme d'écoute d'entité», à la page 190
Vous pouvez écrire des programmes d'écoute d'entité en fonction de vos exigences. Vous trouverez ci-après plusieurs exemples de script.

«Interface EntityTransaction», à la page 201
Vous pouvez utiliser l'interface EntityTransaction pour démarquer les transactions.

Information associée:

«Leçon 2 du tutoriel d'initiation : Création d'une application client», à la page 63
Pour pouvoir insérer, supprimer, mettre à jour et extraire des données dans votre grille de données, vous devez écrire une application client. L'exemple d'initiation inclut une application client que vous pouvez utiliser pour en savoir plus sur la création de votre propre application client.

Tutoriel du gestionnaire d'entités : création d'une classe entité

Créez un ObjectGrid local avec une entité en créant une classe entité, en enregistrant le type d'entité avec et en stockant une instance d'entité dans le cache.

Procédure

1. Créez l'objet Order. Pour identifier l'objet en tant qu'entité ObjectGrid, ajoutez l'annotation @Entity. Lorsque vous ajoutez cette annotation, tous les attributs sérialisables de l'objet sont automatiquement conservés dans eXtreme Scale, à moins que vous n'utilisiez des annotations permettant de substituer ces attributs. L'attribut **orderNumber** est annoté avec `@Id` pour indiquer que l'attribut est une clé primaire. Ci-après, un exemple d'objet Order :

   Order.java

   ```
   @Entity
   public class Order {
       @Id String orderNumber;
       Date date;
       String customerName;
       String itemName;
       int quantity;
       double price;
   }
   ```

2. Exécutez l'application Hello world d'eXtreme Scale pour démontrer les opérations d'entités. L'exemple de programme suivant peut être lancé en mode autonome pour démontrer les opérations d'entités. Utilisez ce programme dans un projet Java Eclipse auquel le fichier `objectgrid.jar` a été ajouté dans le chemin d'accès aux classes. Ci-après, un exemple d'une application Hello world simple qui utilise eXtreme Scale :

   Application.java

   ```
   package emtutorial.basic.step1;

   import com.ibm.websphere.objectgrid.ObjectGrid;
   import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
   import com.ibm.websphere.objectgrid.Session;
   import com.ibm.websphere.objectgrid.em.EntityManager;

   public class Application
   ```

```
{

    static public void main(String [] args)
        throws Exception
    {
        ObjectGrid og =
    ObjectGridManagerFactory.getObjectGridManager().createObjectGrid();
        og.registerEntities(new Class[] {Order.class});

        Session s = og.getSession();
        EntityManager em = s.getEntityManager();

        em.getTransaction().begin();

        Order o = new Order();
        o.customerName = "John Smith";
        o.date = new java.util.Date(System.currentTimeMillis());
        o.itemName = "Widget";
        o.orderNumber = "1";
        o.price = 99.99;
        o.quantity = 1;

        em.persist(o);
        em.getTransaction().commit();

        em.getTransaction().begin();
        o = (Order)em.find(Order.class, "1");
        System.out.println("Found order for customer: " + o.customerName);
        em.getTransaction().commit();
    }
}
```

Cet exemple d'application effectue les opérations suivantes :

a. Initialise une version d'eXtreme Scale locale avec un nom généré automatiquement.
b. Enregistre les classes entité avec l'application à l'aide de l'API registerEntities, même si l'API registerEntities n'est pas toujours nécessaire.
c. Restaure une session et une référence dans le gestionnaire d'entités pour la session.
d. Associe chaque session eXtreme Scale à un seul EntityManager et un seul EntityTransaction. L'EntityManager est à présent utilisé.
e. La méthode registerEntities crée un objet BackingMap appelé Order et associe les métadonnées de cet objet à l'objet BackingMap. Ces métadonnées incluent les attributs clés et non clés, ainsi que les noms et les types d'attribut.
f. Une transaction démarre et crée une instance Order. La transaction est remplie avec des valeurs. La transaction est ensuite conservée à l'aide de la méthode EntityManager.persist, qui identifie l'entité comme étant en attente d'inclusion dans la mappe associée.
g. La transaction est ensuite validée et l'entité est incluse dans ObjectMap.
h. Une autre transaction est créée et l'objet Order est restauré à l'aide de la clé 1. Le transtypage est nécessaire dans la méthode EntityManager.find. La fonction Java SE 5 n'est pas utilisée pour vérifier que le fichier `objectgrid.jar` est compatible avec les machines virtuelles Java.

Tutoriel du gestionnaire d'entités : mise en forme de relations d'entités

Création d'une relation simple entre des entités en créant deux classes entité avec une relation, en enregistrant les entités avec l'ObjectGrid et en stockant les instances d'entités dans le cache.

Procédure

1. Créez l'entité customer qui sert à pour stocker les informations sur les clients indépendamment de l'objet Order. Exemple d'entité customer :

Customer.java

```
@Entity
public class Customer
{
    @Id String id;
    String firstName;
    String surname;
    String address;
    String phoneNumber;
}
```

Cette classe comprend des informations sur le client, telles que le nom, l'adresse et le numéro de téléphone.

2. Créez l'objet Order, qui est similaire à l'objet Order de la rubrique «Tutoriel du gestionnaire d'entités : création d'une classe entité», à la page 9. Ci-après, un exemple d'objet Order :

Order.java

```
@Entity
public class Order {
    @Id String orderNumber;
    Date date;
    @ManyToOne(cascade=CascadeType.PERSIST) Customer customer;
    String itemName;
    int quantity;
    double price;
}
```

Dans cet exemple, une référence à l'objet Customer remplace l'attribut customerName. La référence possède une annotation qui indique une relation plusieurs à un. Ce type de relation indique que chaque commande a un client, mais que plusieurs commandes peuvent référencer le même client. Le modificateur d'annotations en cascade indique que si EntityManager conserve l'objet Order, il doit également conserver l'objet Customer. Si vous décidez de ne pas définir l'option de conservation de la cascade (option par défaut), vous devez conserver manuellement l'objet Customer avec l'objet Order.

3. A l'aide des entités, définissez les mappes pour l'instance ObjectGrid. Chaque mappe est définie pour une entité spécifique : l'une est nommée Order, l'autre Customer. L'exemple d'application suivant illustre le stockage et la récupération d'une commande client :

Application.java

```
public class Application
{
    static public void main(String [] args)
        throws Exception
    {
        ObjectGrid og =
     ObjectGridManagerFactory.getObjectGridManager().createObjectGrid();
        og.registerEntities(new Class[] {Order.class});

        Session s = og.getSession();
        EntityManager em = s.getEntityManager();

        em.getTransaction().begin();

        Customer cust = new Customer();
        cust.address = "Main Street";
        cust.firstName = "John";
        cust.surname = "Smith";
        cust.id = "C001";
        cust.phoneNumber = "5555551212";

        Order o = new Order();
        o.customer = cust;
        o.date = new java.util.Date();
        o.itemName = "Widget";
```

```
        o.orderNumber = "1";
        o.price = 99.99;
        o.quantity = 1;

        em.persist(o);
        em.getTransaction().commit();

        em.getTransaction().begin();
        o = (Order)em.find(Order.class, "1");
        System.out.println("Found order for customer: "
    + o.customer.firstName + " " + o.customer.surname);
        em.getTransaction().commit();
    }
}
```

Cette application est similaire à l'exemple d'application de l'étape précédente. Dans l'exemple précédent, seule une classe Order a été enregistrée. WebSphere eXtreme Scale détecte et inclut automatiquement la référence dans l'entité Customer et une instance Customer pour `John Smith` est créée et référencée depuis le nouvel objet Order. Par conséquent, le nouveau client est automatiquement conservé, car la relation entre deux commandes inclut le modificateur de cascade, qui requiert la conservation de chaque objet. Lorsque l'objet Order est détecté, le gestionnaire d'entités recherche l'objet Customer associé et insère une référence dans l'objet.

Tutoriel du gestionnaire d'entités : schéma d'entité de commande

Création de quatre classes entité à l'aide de relations uniques et bidirectionnelles, de listes ordonnées et de relations de clés externes. Les API Entity sont utilisées pour conserver et rechercher les entités. Construite sur les entités Order et Customer des sections précédentes du tutoriel, cette étape ajoute deux entités supplémentaires : Item et OrderLine.

Pourquoi et quand exécuter cette tâche

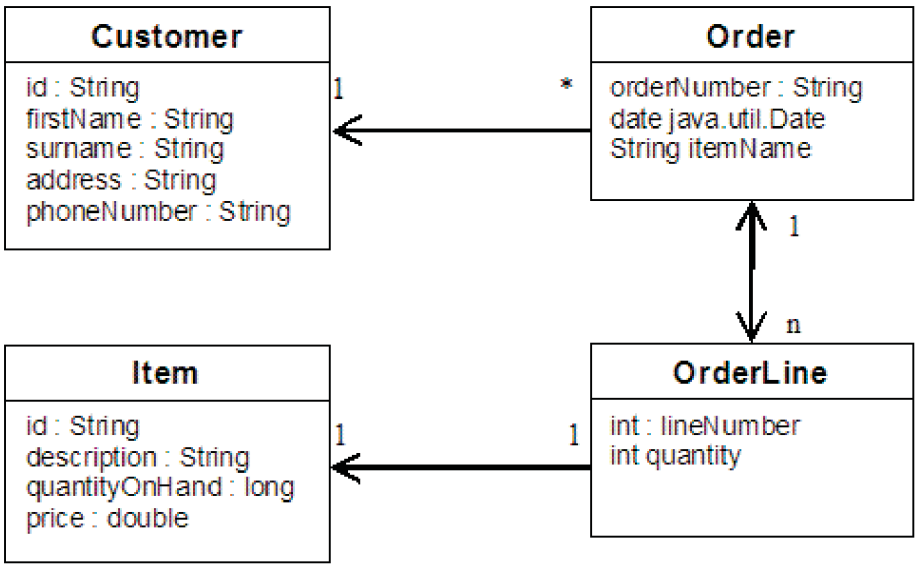


Figure 1. Schéma d'entité de commande. Une entité Order (de commande) a une référence à un client et à zéro ou plus lignes de commande (OrderLines). Chaque entité OrderLine a une référence à un seul article et inclut la quantité commandée.

Procédure

1. Créez l'entité client, similaire aux exemples précédents.

   Customer.java
   ```
   @Entity
   public class Customer
   {
       @Id String id;
       String firstName;
       String surname;
       String address;
       String phoneNumber;
   }
   ```

2. Créez l'entité Item, qui contient les informations sur un produit inclus dans le stock du magasin, telles que la description, la quantité et le prix du produit.

   Item.java
   ```
   @Entity
   public class Item
   {
       @Id String id;
       String description;
       long quantityOnHand;
       double price;
   }
   ```

3. Créez l'entité OrderLine. Chaque entité Order possède zéro ou plus entités OrderLines, qui identifient la quantité de chaque article de la commande. La clé pour OrderLine est une clé composée qui comprend l'entité Order possédée par l'entité OrderLine et un nombre entier qui affecte un numéro à la ligne de commande. Ajoutez le modificateur de conservation de cascade à chaque relation de vos entités.

   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;
   }
   ```

4. Créez l'objet Order final, doté d'une référence au client de la commande et à une collection d'objets OrderLine.

   Order.java
   ```
   @Entity
   public class Order {
       @Id String orderNumber;
       java.util.Date date;
       @ManyToOne(cascade=CascadeType.PERSIST) Customer customer;
       @OneToMany(cascade=CascadeType.ALL, mappedBy="order")
      @OrderBy("lineNumber") List<OrderLine> lines;
   }
   ```

   La cascade ALL est utilisée comme modificateur des lignes. Ce modificateur signale EntityManager pour cascader les opérations PERSIST et REMOVE. Par exemple, si l'entité Order est conservée ou supprimée, toutes les entités OrderLine sont également conservées ou supprimées.

   Si une entité OrderLine est supprimée de la liste des lignes dans l'objet Order, la référence est rompue. Toutefois, l'entité OrderLine n'est pas supprimée du cache. Vous devez utiliser l'API de suppression d'EntityManager pour supprimer les entités du cache. L'opération REMOVE n'est pas utilisée sur

l'entité Customer ou Item de la ligne de commande. Ainsi, l'entité Customer est conservée même si la commande ou l'article est supprimé(e) lors de la suppression de la ligne de commande.

Le modificateur mappedBy indique une relation inverse avec l'entité cible. Le modificateur identifie l'attribut de l'entité cible qui référence l'entité source, ainsi que le côté propriétaire de la relation un à un ou plusieurs à plusieurs. En règle générale, vous pouvez omettre le modificateur. Toutefois, une erreur s'affiche indiquant qu'il doit être spécifié si WebSphere eXtreme Scale ne parvient pas à le détecter automatiquement. Une entité OrderLine qui contient deux des attributs Order dans une relation plusieurs à un provoque généralement cette erreur.

L'annotation @OrderBy spécifie l'ordre dans lequel les entités OrderLine doivent apparaître dans la liste des lignes. Si l'annotation n'est pas spécifiée, les lignes s'affichent dans un ordre arbitraire. Bien que les lignes soient ajoutées à l'entité Order par la soumission d'une liste ArrayList, qui conserve l'ordre, EntityManager ne reconnaît pas nécessairement cet ordre. Lorsque vous émettez la méthode de recherche pour récupérer l'objet Order depuis le cache, l'objet List n'est pas un objet ArrayList.

5. Créez l'application. L'exemple suivant illustre l'objet Order final, qui est doté d'une référence au client de la commande et à une collection d'objets OrderLine.
   a. Recherchez les articles à commander, qui deviennent ensuite des entités gérées.
   b. Créez l'entité OrderLine et liez-la à chaque article.
   c. Créez l'entité Order et associez-la à chaque ligne de commande et au client.
   d. Conservez la commande, qui conserve automatiquement chaque ligne de commande.
   e. Validez la transaction, qui détache chaque entité et synchronise l'état des entités avec le cache.
   f. Imprimez les informations de la commande. Les entités OrderLine sont automatiquement triées par l'ID OrderLine.

```
Application.java

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

        // Ajoutez des articles à notre stock.
        em.getTransaction().begin();
        createItems(em);
        em.getTransaction().commit();

        // Créez un client ayant des articles dans son panier.
        em.getTransaction().begin();
        Customer cust = createCustomer();
        em.persist(cust);

        // Créez une commande et ajoutez une ligne de commande
        // pour chaque article.
        // Chaque article d'une ligne est automatiquement conservé,
    // car l'option Cascade=ALL est définie.
        Order order = createOrderFromItems(em, cust, "ORDER_1",
     new String[]{"1", "2"}, new int[]{1,3});
        em.persist(order);
        em.getTransaction().commit();

        // Imprimez le récapitulatif de la commande.
```

```
        em.getTransaction().begin();
        order = (Order)em.find(Order.class, "ORDER_1");
        System.out.println(printOrderSummary(order));
        em.getTransaction().commit();
    }

    public static Customer createCustomer() {
        Customer cust = new Customer();
        cust.address = "Main Street";
        cust.firstName = "John";
        cust.surname = "Smith";
        cust.id = "C001";
        cust.phoneNumber = "5555551212";
        return cust;
    }

    public static void createItems(EntityManager em) {
        Item item1 = new Item();
        item1.id = "1";
        item1.price = 9.99;
        item1.description = "Widget 1";
        item1.quantityOnHand = 4000;
        em.persist(item1);

        Item item2 = new Item();
        item2.id = "2";
        item2.price = 15.99;
        item2.description = "Widget 2";
        item2.quantityOnHand = 225;
        em.persist(item2);

    }

    public static Order createOrderFromItems(EntityManager em,
   Customer cust, String orderId, String[] itemIds, int[] qty) {

        Item[] items = getItems(em, itemIds);

        Order order = new Order();
        order.customer = cust;
        order.date = new java.util.Date();
        order.orderNumber = orderId;
        order.lines = new ArrayList<OrderLine>(items.length);
    for(int i=0;i<items.length;i++){
      OrderLine line = new OrderLine();
            line.lineNumber = i+1;
            line.item = items[i];
            line.price = line.item.price;
            line.quantity = qty[i];
            line.order = order;
            order.lines.add(line);
        }
        return order;
    }

    public static Item[] getItems(EntityManager em, String[] itemIds) {
        Item[] items = new Item[itemIds.length];
        for(int i=0;i<items.length;i++){
    items[i] = (Item) em.find(Item.class, itemIds[i]);
        }
        return items;
    }
```

L'étape suivante consiste à supprimer une entité. L'interface d'EntityManager est dotée d'une méthode de suppression qui désigne un objet comme étant supprimé. L'application doit supprimer l'entité de toutes les collections de

relations avant d'appeler la méthode de suppression. Pour la dernière étape, modifiez les références et émettez la méthode de suppression, em.remove(object).

Tutoriel du gestionnaire d'entités : mise à jour d'entrées

Si vous souhaitez modifier une entité, vous pouvez rechercher l'instance, mettre à jour cette instance ainsi que toute entité référencée et valider la transaction.

Procédure

Mettez à jour des entrées. L'exemple suivant explique comment rechercher une instance Order, comment modifier cette instance ainsi que toute entité référencée et comment valider la transaction.

```
public static void updateCustomerOrder(EntityManager em) {
    em.getTransaction().begin();
    Order order = (Order) em.find(Order.class, "ORDER_1");
    processDiscount(order, 10);
    Customer cust = order.customer;
    cust.phoneNumber = "5075551234";
    em.getTransaction().commit();
}

public static void processDiscount(Order order, double discountPct) {
    for(OrderLine line : order.lines) {
        line.price = line.price * ((100-discountPct)/100);
    }
}
```

Le vidage de la transaction synchronise toutes les entités gérées avec le cache. Lorsqu'une transaction est validée, un vidage se produit automatiquement. Dans ce cas, l'instance Order devient une entité gérée. Toutes les entités référencées depuis les instances Order, Customer et OrderLine deviennent également des entités gérées. Lorsque la transaction est vidée, chaque entité est vérifiée afin de déterminer si elle a été modifiée. Les entités modifiées sont mises à jour dans le cache. Une fois la transaction terminée, après sa validation ou son annulation, les entités sont détachées et les modifications qui y sont apportées ne sont pas reflétées dans le cache.

Tutoriel du gestionnaire d'entités : mise à jour et suppression d'entrées à l'aide d'un index

Vous pouvez utiliser un index pour rechercher, mettre à jour et supprimer des entités.

Procédure

Mettez à jour et supprimez des entités à l'aide d'un index. Utilisez un index pour rechercher, mettre à jour et supprimer des entités. Dans les exemples suivants, la classe de l'entité Order est mise à jour afin d'utiliser l'annotation @Index. L'annotation @Index signale à WebSphere eXtreme Scale de créer un index d'intervalles pour un attribut. Le nom de l'index est identique au nom de l'attribut et l'index est toujours de type MapRangeIndex.

Order.java

```
@Entity
public class Order {
    @Id String orderNumber;
    @Index java.util.Date date;
    @OneToOne(cascade=CascadeType.PERSIST) Customer customer;
    @OneToMany(cascade=CascadeType.ALL, mappedBy="order")
    @OrderBy("lineNumber") List<OrderLine> lines;  }
```

L'exemple suivant montre l'annulation de toutes les commandes soumises au cours de la dernière minute. Recherchez la commande à l'aide d'un index, replacez les articles de la commande en stock et supprimez la commande ainsi que les articles de la ligne associée du système.

```
public static void cancelOrdersUsingIndex(Session s)
  throws ObjectGridException {
    // Annulez toutes les commandes soumises il y a une minute.
    java.util.Date cancelTime = new
   java.util.Date(System.currentTimeMillis() - 60000);
    EntityManager em = s.getEntityManager();
    em.getTransaction().begin();
    MapRangeIndex dateIndex = (MapRangeIndex)
   s.getMap("Order").getIndex("date");
    Iterator<Tuple> orderKeys = dateIndex.findGreaterEqual(cancelTime);
  while(orderKeys.hasNext()) {
   Tuple orderKey = orderKeys.next();
   // Recherchez la commande à supprimer.
   Order curOrder = (Order) em.find(Order.class, orderKey);
   // Vérifiez que la commande n'a pas été mise à jour par une autre personne.
   if(curOrder != null && curOrder.date.getTime() >= cancelTime.getTime()) {
    for(OrderLine line : curOrder.lines) {
     // Replacez l'article en stock.
     line.item.quantityOnHand += line.quantity;
     line.quantity = 0;
    }
   em.remove(curOrder);
   }
  }
 em.getTransaction().commit();
}
```

Tutoriel du gestionnaire d'entités : mise à jour et suppression d'entrées à l'aide d'une requête

Vous pouvez mettre à jour et supprimer des entités à l'aide d'une requête.

Procédure

Mettez à jour et supprimez des entités à l'aide d'une requête.

Order.java

```
@Entity
public class Order {
    @Id String orderNumber;
    @Index java.util.Date date;
    @OneToOne(cascade=CascadeType.PERSIST) Customer customer;
    é@OneToMany(cascade=CascadeType.ALL, mappedBy="order")
   @OrderBy("lineNumber") List<OrderLine> lines;
}
```

La classe de l'entité Order est identique à celle de l'exemple précédent. La classe fournit toujours l'annotation @Index, car la chaîne de requête utilise la date pour rechercher l'entité. Le moteur de requête utilise des index chaque fois que possible.

```
public static void cancelOrdersUsingQuery(Session s) {
        // Annulez toutes les commandes soumises il y a une minute.
        java.util.Date cancelTime =
     new java.util.Date(System.currentTimeMillis() - 60000);
        EntityManager em = s.getEntityManager();
        em.getTransaction().begin();

        // Créez une requête qui recherche la commande par rapport à la date.  Etant donné
        // que nous avons un index défini sur la date de commande, la requête
    // l'utilisera automatiquement.
        Query query = em.createQuery("SELECT order FROM Order order
     WHERE order.date >= ?1");
        query.setParameter(1, cancelTime);
        Iterator<Order> orderIterator = query.getResultIterator();
    while(orderIterator.hasNext()) {
     Order order = orderIterator.next();
     // Vérifiez que la commande n'a pas été mise à jour par une autre personne.
     // Etant donné que la requête a utilisé un index, il n'y avait pas de verrou sur la ligne.
```

```
    if(order != null && order.date.getTime() >= cancelTime.getTime()) {
      for(OrderLine line : order.lines) {
       // Replacez l'article en stock.
       line.item.quantityOnHand += line.quantity;
       line.quantity = 0;
      }
      em.remove(order);
    }
  }
  em.getTransaction().commit();
}
```

Comme dans l'exemple précédent, la méthode cancelOrdersUsingQuery tente d'annuler toutes les commandes soumises au cours de la dernière minute. Pour annuler la commande, recherchez la commande à l'aide d'une requête, replacez les articles de la commande en stock et supprimez la commande ainsi que les articles de la ligne associée du système.

Tutoriel : Exécution des ensembles eXtreme Scale dans la structure OSGi

L'exemple OSGi repose sur les exemples de sérialiseurs Google Protocol Buffers. A la fin de ce groupe de leçons, vous aurez exécuter les exemples de plug-ins du sérialiseur dans l'infrastructure OSGi.

Objectifs d'apprentissage

Cet exemple montre les ensembles OSGi. Le plug-in de sérialiseur est secondaire et il n'est pas requis. L'exemple OSGi est disponible dans la galerie des exemples WebSphere eXtreme Scale. Vous devez télécharger l'exemple et l'extraire dans le répertoire *wxs_home*/samples. Le répertoire racine de l'exemple OSGi est wxs_home/samples/OSGiProto.

L'exemple de sérialiseur Google Buffers Protocole se trouve dans le répertoire *wxs_home*/samples/SerializerProto.

L'exemple de sérialiseur Binary JSON (BSON) se trouve dans le répertoire *wxs_home*/samples/SerializerBSON.

Les exemples de commande dans ce tutoriel suppose que vous exécutez le système d'exploitation UNIX. Vous devez ajuster l'exemple de commande pour l'exécuter sur un système d'exploitation Windows.

A la fin des leçons de ce tutoriel, vous comprendrez les concepts des exemples OSGi et saurez :

- installer l'ensemble de serveur WebSphere eXtreme Scale dans le conteneur OSGi pour démarrer le serveur eXtreme Scale ;
- configurer votre environnement de développement eXtreme Scale pour exécuter l'exemple de client ;
- utiliser la commande xscmd pour interroger le classement de l'exemple d'ensemble, le mettre à niveau vers un nouveau classement de services et vérifier le nouveau classement de services.

Durée

Ce module prend 60 minutes environ.

Prérequis

Dans ce tutoriel, vous devez télécharger et extraire les exemples de sérialiseurs et :

- Installer et extraire le produit eXtreme Scale
- Configurer l'environnement Eclipse Equinox

Introduction : Démarrage et configuration du serveur eXtreme Scale et du conteneur pour exécuter les plug-ins dans la structure OSGi

Dans ce tutoriel, vous allez démarrer un serveur eXtreme Scale dans l'infrastructure OSGi, démarrer un conteneur eXtreme Scale et connecter les exemples de plug-ins avec l'environnement d'exécution eXtreme Scale.

Objectifs d'apprentissage

A la fin des leçons de ce tutoriel, vous comprendrez les concepts des exemples OSGi et saurez :

- installer l'ensemble de serveur WebSphere eXtreme Scale dans le conteneur OSGi pour démarrer le serveur eXtreme Scale ;
- configurer l'environnement de développement eXtreme Scale pour exécuter l'exemple de client ;
- utiliser la commande xscmd pour interroger le classement de l'exemple d'ensemble, le mettre à niveau vers un nouveau classement de services et vérifier le nouveau classement de services.

Durée

Ce tutorial prend 60 minutes environ. Si vous explorez d'autres concepts liés à ce tutoriel, il peut prendre plus de temps.

Niveau de compétence

Intermédiaire

Audience

Les développeurs et les administrateurs qui veulent créer, installer et exécuter des ensembles eXtreme Scale dans l'infrastructure OSGi.

Configuration requise

- Client de ligne de commande Luminis OSGi Configuration Admin, version 0.2.5
- Apache Felix File Install, version 3.0.2
- Lorsque vous utilisez Eclipse Gemini en tant que fournisseur de conteneur Blueprint, les éléments suivants sont requis :
  - Eclipse Gemini Blueprint, version 1.0.0
  - Spring Framework, version 3.0.5
  - SpringSource AOP Alliance API, version 1.0.0
  - SpringSource Apache Commons Logging, version 1.1.1
- Lorsque vous utilisez Apache Aries en tant que fournisseur du conteneur Blueprint, vous devez disposer de la configuration suivante :
  - Dernière image instantanée Aries

– Bibliothèque ASM
– Consignation PAX

Prerequis

Pour pouvoir exécuter ce tutoriel, vous devez télécharger l'exemple et l'extraire dans le répertoire `wxs_home/samples`. Le répertoire racine de l'exemple OSGi est `wxs_home/samples/OSGiProto`.

Résultats attendus

A la fin de ce tutoriel, vous aurez installé les exemples d'ensembles et exécuté un client eXtreme Scale pour insérer des données dans la grille. Vous serez également amené à interroger et mettre à jour ces exemples d'ensembles en utilisant les fonctions dynamiques que fournit le conteneur OSGi.

Concepts associés:

«Présentation de l'infrastructure OSGi», à la page 37
OSGi définit un système de module dynamique pour Java. La plateforme de service OSGi présente une architecture à couches et elle est conçue pour s'exécuter dans plusieurs profils standard Java. Vous pouvez démarrer les serveurs et les clients WebSphere eXtreme Scale dans un conteneur OSGi.

Tâches associées:

«Installation de l'infrastructure OSGi Eclipse Equinox avec Eclipse Gemini pour les clients et les serveurs», à la page 39
Si vous souhaitez déployer WebSphere eXtreme Scale dans la structure OSGi, vous devez configurer l'environnement Eclipse Equinox.

Référence associée:

Fichier de propriétés du serveur
Le fichier de propriétés du serveur contient plusieurs propriétés qui définissent différents paramètres pour votre serveur, tels que les paramètres de trace, la consignation et la configuration de la sécurité. Le fichier de propriétés du serveur est utilisé par les services de catalogue et les serveurs de conteneur dans les serveurs autonomes et les serveurs hébergés dans WebSphere Application Server.

Module 1 : Préparation de l'installation et de la configuration des ensembles de serveur eXtreme Scale

Effectuez ce module pour explorer les exemples d'ensembles OSGi et examiner les fichiers de configuration que vous utilisez pour configurer le serveur eXtreme Scale.

Objectifs d'apprentissage

A la fin des leçons de ce module, vous comprendrez les concepts et saurez :

- localiser et explorer les ensembles qui sont inclus dans le modèle OSG ;
- Examinez les fichiers de configuration utilisés pour configurer la grille et le serveur eXtreme Scale.

Leçon 1.1 : Explication des exemples d'ensembles OSGi

Suivez cette leçon pour localiser et explorer les ensembles fournis dans l'exemple OSGi.

Exemples d'ensembles OSGi :

Hormis les ensembles qui sont configurés dans le fichier `config.ini`, qui est indiqué dans la rubrique sur la configuration de l'environnement Eclipse Equinox, les ensembles supplémentaires suivants sont utilisés dans le modèle OSGi :

objectgrid.jar
Ensemble d'exécution de serveur WebSphere eXtreme Scale. Cet ensemble se trouve dans le répertoire *wxs_home*`/lib`.

com.google.protobuf_2.4.0a.jar
Ensemble Google Protocol Buffers, version 2.4.0a. Cet ensemble se trouve dans le répertoire *wxs_sample_osgi_root*`/lib`.

ProtoBufSamplePlugins-1.0.0.jar
Version 1.0.0 de l'ensemble de plug-in utilisateur avec l'exemple ObjectGridEventListener et les implémentations de plug-in MapSerializerPlugin. Cet ensemble se trouve dans le répertoire *wxs_sample_osgi_root*`/lib`. Les services sont configurés avec le classement de service 1.

Cette version utilise le XML Blueprint standard pour configurer les services de plug-in eXtreme Scale. La classe de service est une classe implémentée par l'utilisateur pour l'interface WebSphere eXtreme Scale, com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactory. La classe implémentée par l'utilisateur crée un bean pour chaque demande et fonctionne de la même manière qu'un bean de portée prototype.

ProtoBufSamplePlugins-2.0.0.jar
Version 2.0.0 de l'ensemble de plug-in utilisateur avec l'exemple ObjectGridEventListener et les implémentations de plug-in MapSerializerPlugin. Cet ensemble se trouve dans le répertoire *wxs_sample_osgi_root*`/lib`. Les services sont configurés avec le classement de service 2.

Cette version utilise le XML Blueprint standard pour configurer les services de plug-in eXtreme Scale. La classe de service utilise une classe intégrée WebSphere eXtreme Scale, com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactoryImpl, qui utilise le service BlueprintContainer. En utilisant la configuration XML Blueprint standard, les beans peuvent être configurés en tant que portée singleton ou portée prototype. Le bean n'est pas configuré en tant que portée de fragment.

ProtoBufSamplePlugins-Gemini-3.0.0.jar
Version 3.0.0 de l'ensemble de plug-in utilisateur avec l'exemple ObjectGridEventListener et les implémentations de plug-in MapSerializerPlugin. Cet ensemble se trouve dans le répertoire *wxs_sample_osgi_root*`/lib`. Les services sont configurés avec le classement de service 3.

Cette version utilise le XML Blueprint XML d'Eclipse Gemini pour configurer les services de plug-in eXtreme Scale. La classe de service utilise une classe de service intégrée, WebSphere eXtreme Scale, com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactoryImpl, qui utilise le service BlueprintContainer. Pour configurer un bean de portée de fragment, utilisez une approche Gemini. Cette version configure le bean myShardListener comme bean de portée de fragment en fournissant `{http://www.ibm.com/schema/objectgrid}shard` comme valeur de portée et en configurant un attribut factice pour que Gemini reconnaisse la portée personnalisée. Le problème Eclipse est généré par : https://bugs.eclipse.org/bugs/show_bug.cgi?id=348776

ProtoBufSamplePlugins-Aries-4.0.0.jar
Version 4.0.0 de l'ensemble de plug-in utilisateur avec l'exemple ObjectGridEventListener et les implémentations de plug-in MapSerializerPlugin. Cet ensemble se trouve dans le répertoire *wxs_sample_osgi_root*/lib. Les services sont configurés avec le classement de service 4.

Cette version utilise le XML Blueprint standard pur configurer les services de plug-in eXtreme Scale. La classe de service utilise une classe intégrée WebSphere eXtreme Scale, com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactoryImpl, qui utilise le service BlueprintContainer. En utilisant la configuration XML Blueprint standard, les beans peuvent être configurés en utilisant une porté personnalisée. Cette version configure myShardListenerbean comme bean à portée de fragment en fournissant {http://www.ibm.com/schema/objectgrid}shard comme valeur de portée.

ProtoBufSamplePlugins-Activator-5.0.0.jar
Version 5.0.0 de l'ensemble de plug-in utilisateur avec l'exemple ObjectGridEventListener et les implémentations de plug-in MapSerializerPlugin. Cet ensemble se trouve dans le répertoire *wxs_sample_osgi_root*/lib. Les services sont configurés avec le classement de service 5.

Cette version n'utilise pas du tout le conteneur Blueprint. Dans cette version, les services sont enregistrés à l'aide de l'enregistrement de service OSGi. La classe de service est une classe implémentée par l'utilisateur pour l'interface WebSphere eXtreme Scale, com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactory. La classe implémentée par l'utilisateur crée un bean pour chaque demande. Elle fonctionne d'une manière similaire à un bean à portée prototype.

Point de contrôle de la leçon :

En explorant les ensembles qui sont fournis avec le modèle OSGi, vous pouvez mieux comprendre la procédure de développement de vos propres implémentations qui s'exécutent dans le conteneur OSGi.

Vous avez appris :
- les ensembles inclus avec l'exemple OSGi ;
- l'emplacement de ces ensembles ;
- l'élément utilisé pour configurer le classement de service de chaque ensemble.

Leçon 1.2 : Description des fichiers de configuration OSGi

L'exemple OSGi contient trois fichiers de configuration. Vous utilisez ces fichiers pour démarrer et configurer la grille WebSphere eXtreme Scale et le serveur.

Fichiers de configuration OSGi :

Dans cette leçon, vous allez explorer les fichiers de configuration suivants :
- collocated.server.properties
- protoBufObjectGrid.xml
- protoBufDeployment.xml

collocated.server.properties

Une configuration de serveur est nécessaire pour démarrer un serveur. Lorsque l'ensemble de serveur eXtreme Scale est démarré, il ne démarre pas un serveur. Il attend la création du PID de configuration `com.ibm.websphere.xs.server` avec un fichier de propriétés de serveur. Ce fichier de propriétés du serveur indique le nom du serveur, le numéro de port et d'autres propriétés du serveur.

Dans la plupart des cas, vous créerez une configuration pour définir le fichier des propriétés du serveur. Dans de rares cas, vous pouvez vouloir uniquement démarrer un serveur avec chaque propriété affectée d'une valeur par défaut. Dans ce cas, vous pouvez créer une configuration appelée `com.ibm.websphere.xs.server` avec la valeur `default`.

Pour plus d'informations sur le fichier des propriétés du serveur, voir la rubrique Fichier de propriétés du serveur.

L'exemple OSGi inclut le fichier de propriétés de serveur *wxs_sample_osgi_root*`/server/properties/collocated.server.properties`. Ce fichier de propriétés démarre un service de catalogue unique et un serveur de conteneur dans le processus d'infrastructure OSGi. Les clients eXtreme Scale se connectent sur le port 2809 et les clients JMX, sur le port 1099. Contenu du fichier de propriétés de serveur :

```
serverName=collocatedServer
isCatalog=true
catalogClusterEndPoints=collocatedServer:localhost:6601:6602
traceSpec=ObjectGridOSGi=all=enabled
traceFile=logs/trace.log
listenerPort=2809
JMXServicePort=1099
```

protoBufObjectGrid.xml

L'exemple de fichier XML descripteur `protoBufObjectGrid.xml` ObjectGrid contient les éléments suivants avec les commentaires supprimés.

```
<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" txTimeout="15">

            <bean id="ObjectGridEventListener"
                osgiService="myShardListener"/>

            <backingMap name="Map" readOnly="false"
                lockStrategy="PESSIMISTIC" lockTimeout="5"
                copyMode="COPY_TO_BYTES"
                pluginCollectionRef="serializer"/>

        </objectGrid>
    </objectGrids>

    <backingMapPluginCollections>
        <backingMapPluginCollection id="serializer">
            <bean id="MapSerializerPlugin"
                osgiService="myProtoBufSerializer"/>"/>
        </backingMapPluginCollection>
    </backingMapPluginCollections>
 </objectGridConfig>
```

Il existe deux plug-ins configurés dans ce fichier descripteur XML ObjectGrid :

ObjectGridEventListener

Plug-in au niveau du fragment. Pour chaque instance ObjectGrid, il existe une instance de ObjectGridEventListener. Elle est configurée pour utiliser le service OSGi myShardListener. Cela signifie que lorsque la grille est créée, le plug-in ObjectGridEventListener utilise le service OSGi myShardListener avec le classement de service le plus élevé disponible.

MapSerializerPlugin

Plug-in au niveau de la mappe. Pour la mappe de sauvegarde nommée `Map`, il existe un plug-in MapSerializerPlugin configuré. Il est configuré pour utiliser le service OSGI myProtoBufSerializer. Cela signifie que lorsque la grille est créée, le plug-in MapSerializerPlugin utilise le service, myProtoBufSerializer, avec le classement de service le plus élevé disponible.

protoBufDeployment.xml

Le fichier descripteur XML de déploiement décrit la stratégie de déploiement pour la grille `Grid` qui utilise cinq partitions. Reportez-vous à l'exemple de code suivant de ce fichier XML :

```
<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="5">
      <map ref="Map"/>
    </mapSet>
  </objectgridDeployment>
</deploymentPolicy>
```

blueprint.xml

Comme alternative à l'utilisation du fichier `collocated.server.properties` en association avec le PID de configuration, `com.ibm.websphere.xs.server`, vous pouvez inclure le fichier XML ObjectGrid et des fichiers XML de déploiement dans un ensemble OSGi, avec un fichier XML Blueprint, comme dans l'exemple suivant :

```
<blueprint>
   xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
       xmlns:objectgrid="http://www.ibm.com/schema/objectgrid"
       default-activation="lazy">

    <objectgrid:server id="server" isCatalog="true"
                name="server"
                tracespec="ObjectGridOSGi=all=enabled"
                tracefile="C:/Temp/logs/trace.log"
                workingDirectory="C:/Temp/working"
                jmxport="1099">
        <objectgrid:catalog host="localhost" port="2809"/>
    </objectgrid:server>

    <objectgrid:container id="container"
  objectgridxml="/META-INF/objectgrid.xml"
       deploymentxml="/META-INF/deployment.xml"
  server="server"/>
</blueprint>
```

Point de contrôle de la leçon :

Dans cette leçon, vous avez découvert les fichiers de configuration qui sont utilisées dans l'exemple OSGi. Maintenant, lorsque vous démarrez et configurez la

grille eXtreme Scale et le serveur, vous savez quels fichiers sont utilisés dans ces processus et comment ces fichiers interagissent avec vos plug-ins dans l'infrastructure OSGi.

Module 2 : Installation et démarrage des ensembles eXtreme Scale dans l'infrastructure OSGi

Utilisez les modules de cette leçon pour installer l'ensemble de serveur eXtreme Scale dans le conteneur OSGi et démarrer le serveur WebSphere eXtreme Scale.

Le démarrage du serveur dans l'infrastructure OSGi n'implique pas que les ensembles OSGi sont prêts à être exécutés. Vous devez configurer les propriétés du serveur et les conteneurs de sorte que les ensembles OSGi que vous installez soient reconnus et puissent s'exécuter correctement.

Objectifs d'apprentissage

A la fin des leçons de ce module, vous comprendrez les concepts et saurez :

- installer les ensembles eXtreme Scale en utilisant la console OSGi Equinox ;
- configurer le serveur eXtreme Scale ;
- configurer le conteneur eXtreme Scale ;
- démarrer les exemples d'ensembles eXtreme Scale.

Prérequis

Pour pouvoir exécuter ce module, vous devez effectuer préalablement les tâches suivantes :

- Installer et extraire le produit eXtreme Scale
- Définir l'environnement Eclipse Equinox

Vous devez également préparer l'accès aux fichiers suivants pour suivre les leçons de ce module :

- Ensemble `objectgrid.jar`. Vous installez cet ensemble eXtreme Scale.
- `Fichier collocated.server.properties`. Vous ajoutez les propriétés du serveur à ce fichier de configuration.
- Vous pouvez envisager d'installer et de démarrer les ensembles suivants :
- `protobuf-java-2.4.0a-bundle.jar`
- `ProtoBufSamplePlugins-1.0.0.jar`
- `ProtoBufSamplePlugins-2.0.0.jar`

Leçon 2.1 : Démarrage de la console et installation de l'ensemble de serveur eXtreme Scale

Dans cette leçon, vous utilisez la console Equinox OSGi pour lancer et installer un WebSphere eXtreme Scale

1. Utilisez la commande suivante pour démarrer la console OSGi Equinox :

   ```
   cd equinox_root

   java -jar
   plugins\org.eclipse.osgi_3.6.1.R36x_v20100806.jar
   -console
   ```

2. Une fois la console OSGi démarrée, exécutez la commande `ss` dans la console ; les ensembles suivants sont démarrés :

Sortie Eclipse Gemini :

```
osgi> ss
Framework is launched.
id State Bundle
0 ACTIVE org.eclipse.osgi_3.6.1.R36x_v20100806
1 ACTIVE org.eclipse.osgi.services_3.2.100.v20100503
2 ACTIVE org.eclipse.osgi.util_3.2.100.v20100503
3 ACTIVE org.eclipse.equinox.cm_1.0.200.v20100520
4 ACTIVE com.springsource.org.apache.commons.logging_1.1.1
5 ACTIVE com.springsource.org.aopalliance_1.0.0
6 ACTIVE org.springframework.aop_3.0.5.RELEASE
7 ACTIVE org.springframework.asm_3.0.5.RELEASE
8 ACTIVE org.springframework.beans_3.0.5.RELEASE
9 ACTIVE org.springframework.context_3.0.5.RELEASE
10 ACTIVE org.springframework.core_3.0.5.RELEASE
11 ACTIVE org.springframework.expression_3.0.5.RELEASE
12 ACTIVE org.apache.felix.fileinstall_3.0.2
13 ACTIVE net.luminis.cmc_0.2.5
14 ACTIVE org.eclipse.gemini.blueprint.core_1.0.0.RELEASE
15 ACTIVE org.eclipse.gemini.blueprint.extender_1.0.0.RELEASE
16 ACTIVE org.eclipse.gemini.blueprint.io_1.0.0.RELEASE
```

Sortie Apache Aries :

```
osgi> ss
Framework is launched.
id State Bundle
0 ACTIVE org.eclipse.osgi_3.6.1.R36x_v20100806
1 ACTIVE org.eclipse.osgi.services_3.2.100.v20100503
2 ACTIVE org.eclipse.osgi.util_3.2.100.v20100503
3 ACTIVE org.eclipse.equinox.cm_1.0.200.v20100520
4 ACTIVE org.ops4j.pax.logging.pax-logging-api_1.6.3
5 ACTIVE org.ops4j.pax.logging.pax-logging-service_1.6.3
6 ACTIVE org.objectweb.asm.all_3.3.0
7 ACTIVE org.apache.aries.blueprint_0.3.2.SNAPSHOT
8 ACTIVE org.apache.aries.util_0.4.0.SNAPSHOT
9 ACTIVE org.apache.aries.proxy_0.4.0.SNAPSHOT
10 ACTIVE org.apache.felix.fileinstall_3.0.2
11 ACTIVE net.luminis.cmc_0.2.5
```

3. Installez l'ensemble `objectgrid.jar`. Pour démarrer un serveur dans la machine JVM (Java virtual machine), vous devez installer un ensemble de serveur eXtreme Scale. Cet ensemble de serveur eXtreme Scale peut démarrer un serveur et créer des conteneurs. Utilisez la commande suivante pour installer le fichier `objectgrid.jar` :

   ```
   osgi> install file:///wxs_home/lib/objectgrid.jar
   ```

   Reportez-vous à l'exemple suivant :

   ```
   osgi> install
   file:///opt/wxs/ObjectGrid/lib/objectgrid.jar
   ```

   Equinox affiche son ID d'ensemble, par exemple :

   ```
   Bundle id is 19
   ```

   A faire : Votre ID d'ensemble peut être différent. Le chemin de ficher doit être une adresse URL absolue dans le chemin de l'ensemble. Les chemins relatifs ne sont pas pris en charge.

Point de contrôle de la leçon :

Dans cette leçon, vous avez utilisé la console OSGi Equinox pour installer l'ensemble `objectgrid.jar` que vous allez utiliser pour démarrer et créer ensuite un conteneur dans ce tutoriel.

Leçon 2.2 : Personnalisation et configuration du serveur eXtreme Scale

Suivez cette leçon pour personnaliser et ajouter les propriétés au serveur WebSphere eXtreme Scale.

1. Editez le fichier `wxs_sample_osgi_root/server/properties/collocated.server.properties`.
   a. Remplacez la propriété workingDirectory par `equinox_root`.
   b. Remplacez la propriété traceFile par `equinox_root/logs/trace.log`.
2. Enregistrez le fichier.
3. Entrez les lignes de code suivantes dans la console OSGi pour créer la configuration du serveur à partir du fichier :

```
osgi> cm create com.ibm.websphere.xs.server

osgi> cm put com.ibm.websphere.xs.server
objectgrid.server.props
wxs_sample_osgi_root/server/properties/collocated.serve
r.properties
```

4. Pour afficher la configuration, exécutez la commande suivante :

```
osgi> cm get com.ibm.websphere.xs.server
Configuration for service (pid) "com.ibm.websphere.xs.server"
(bundle location = null)
key value
------ ------
objectgrid.server.props objectgrid.server.props
```

Point de contrôle de la leçon :

Au cours de cette leçon, vous avez édité le fichier `wxs_sample_osgi_root/server/properties/collocated.server.properties` pour spécifier les paramètres du serveur, telles que le répertoire de travail et l'emplacement des fichiers journaux de trace.

Leçon 2.3 : Configuration du conteneur eXtreme Scale

Suivez cette leçon pour configurer un conteneur qui inclut le fichier descripteur XML d'ObjectGrid et le fichier XML de déploiement d'ObjectGrid WebSphere eXtreme Scale. Ces fichiers incluent la configuration de la grille et sa topologie.

Pour créer un conteneur, commencez par créer un service de configuration à l'aide du PID (process identification number) de la fabrique de services gérés, com.ibm.websphere.xs.container. La configuration de service est une fabrique de services gérés qui permet de créer plusieurs PID de service depuis le PID de la fabrique. Pour démarrer le service de conteneur, affectez aux PID objectgridFile et deploymentPolicyFile chaque PID de service.

Procédez comme suit pour personnaliser et ajouter les propriétés du serveur à l'infrastructure OSGi :

1. Dans la console OSGI, entrez la commande suivante pour créer le conteneur depuis le fichier :

```
osgi> cm createf com.ibm.websphere.xs.container
PID: com.ibm.websphere.xs.container-1291179621421-0
```

2. Entrez la commande suivante pour lier le PID nouvellement créé aux fichiers XML ObjectGrid.

   A faire : Le numéro PID sera différent de celui de cet exemple.

```
osgi> cm put com.ibm.websphere.xs.container-1291179621421-0
objectgridFile wxs_sample_osgi_root/server/META-INF/protoBufObjectgrid.xml

osgi> cm put com.ibm.websphere.xs.container-1291179621421-0
deploymentPolicyFile wxs_sample_osgi_root/server/META-INF/protoBufDeployment.xml
```

3. Utilisez la commande suivante pour afficher la configuration :

```
osgi> cm get com.ibm.websphere.xs.container-1291760127968-0
Configuration for service (pid) "com.ibm.websphere.xs.container-1291760127968-0"
(bundle location = null)

key value
------                  ------
deploymentPolicyFile    /opt/wxs/ObjectGrid/samples/OSGiProto/server/META-INF/protoBufDeployment.xml
objectgridFile          /opt/wxs/ObjectGrid/samples/OSGiProto/server/META-INF/protoBufObjectgrid.xml
service.factoryPid    com.ibm.websphere.xs.container
service.pid        com.ibm.websphere.xs.container-1291760127968-0
```

Point de contrôle de la leçon :

Dans cette leçon, vous avez créé un service de configuration qui vous a permis de créer un conteneur eXtreme Scale. Comme les fichiers XML ObjectGrid contiennent la configuration de la grille et sa topologie, vous devez lier le conteneur que vous avez créé pour ces fichiers XML ObjectGrid. Avec cette configuration, le conteneur eXtreme Scale peut reconnaître les ensembles OSGi que vous allez exécuter ultérieurement dans ce tutoriel.

Leçon 2.4 : Installation de Google Protocol Buffers et des ensembles de plug-ins

Suivez ce tutoriel pour installer l'ensemble `protobuf-java-2.4.0a-bundle.jar` et l'ensemble de plug-ins `ProtoBufSamplePlugins-1.0.0.jar` en utilisant la console OSGi Equinox OSGi.

Procédez comme suit pour installer l'ensemble Google Protocol Buffers.

Dans la console OSGI, entrez la commande suivante pour installer l'ensemble :

```
osgi> install file:///wxs_sample_osgi_root/common/lib/com.google.protobuf_2.4.0a.jar
```

La sortie suivante s'affiche :

```
Bundle ID is 21
```

Présentation des exemples d'ensembles de plug-ins :

L'exemple OSGi inclut cinq exemples d'ensembles qui contiennent les plug-ins eXtreme Scale, notamment un plug-in personnalisé ObjectGridEventListener et un plug-in MapSerializerPlugin. Le plug-in MapSerializerPlugin utilise l'exemple Google Protocol Buffers et les messages fournis par l'exemple MapSerializerPlugin.

Les ensembles suivants se trouvent dans le répertoire *wxs_sample_osgi_root*`/lib` : `ProtoBufSamplePlugins-1.0.0.jar` et `ProtoBufSamplePlugins-2.0.0.jar`.

Le fichier `blueprint.xml` contient ce qui suit sans les commentaires :

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
 <bean id="myShardListener" class="com.ibm.websphere.samples.xs.proto.osgi.MyShardListenerFactory"/>
 <service ref="myShardListener" interface="com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactory" ranking="1">
 </service>

 <bean id="myProtoBufSerializer" class="com.ibm.websphere.samples.xs.proto.osgi.ProtoMapSerializerFactory">
  <property name="keyType" value="com.ibm.websphere.samples.xs.serializer.app.proto.DataObjects1$OrderKey" />
  <property name="valueType" value="com.ibm.websphere.samples.xs.serializer.app.proto.DataObjects1$Order" />
 </bean>
```

```
 <service ref="myProtoBufSerializer" interface="com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactory"
  ranking="1">
 </service>
</blueprint>
```

Le fichier XML Blueprint exporte les deux services myShardListener et myProtoBufSerializer. Ces deux services sont référencés dans le fichier `protoBufObjectgrid.xml`

Installation de l'ensemble de plug-ins :

Procédez comme suit pour installer l'ensemble `ProtoBufSamplePlugins-1.0.0.jar`.

Exécutez la commande suivante dans la console OSGi Equinox pour installer l'ensemble de plug-ins `ProtoBufSamplePlugins-1.0.0.jar` :

```
osgi> install file:///wxs_sample_osgi_root/common/lib/ProtoBufSamplePlugins-1.0.0.jar
```

La sortie suivante s'affiche :

```
Bundle ID is 22
```

Point de contrôle de la leçon :

Dans cette leçon, vous avez installé l'ensemble `protobuf-java-2.4.0a-bundle.jar` et l'ensemble de plug-ins `ProtoBufSamplePlugins-1.0.0.jar`.

Leçon 2.5 : Démarrage des ensembles OSGi

Le serveur WebSphere eXtreme Scale est modularisé comme ensemble de serveur OSGi. Suivez cette leçon pour installer l'ensemble de serveur eXtreme Scale et d'autres ensembles OSGi que vous avez installés.

1. Démarrer l'exemple d'ensemble de plug-in. Exécutez la commande suivante dans la console OSGi Equinox pour démarrer l'ensemble. Dans cet exemple, l'ID d'ensemble de l'exemple de plug-in est 22.

   ```
   osgi> start 22
   ```

2. Démarrer l'ensemble Google Protocol Buffers. Exécutez la commande suivante dans la console OSGi Equinox pour démarrer l'ensemble. Dans cet exemple, l'ID d'ensemble du plug-in Google Protocol Buffers est 21.

   ```
   osgi> start 21
   ```

3. Démarrer l'ensemble de serveur. Exécutez la commande suivante dans la console OSGi Equinox pour démarrer le serveur. Dans cet exemple, l'ID de l'ensemble de serveur eXtreme Scale est 19.

   ```
   osgi> start 19
   ```

Après avoir démarré le serveur, le programme d'écoute d'événement MyShardListener démarre et il est prêt à insérer ou mettre à jour les enregistrements. Vous pouvez visualiser la sortie suivante sur la console OSGi pour vérifier que l'ensemble de plug-in a démarré correctement :

```
SystemOut O MyShardListener@1253853884(version=1.0.0) order
com.ibm.websphere.samples.xs.serializer.proto.DataObjects1$Order$Builder
@1aba1aba(22) inserted
```

Point de contrôle de la leçon :

Dans cette leçon, vous avez démarré deux ensembles de plug-in et l'ensemble de serveur dans le conteneur eXtreme Scale que vous avez configuré pour l'infrastructure OSGi.

Module 3 : Exécution de l'exemple de client eXtreme Scale

Le serveur WebSphere eXtreme Scale fonctionne maintenant dans un environnement OSGi. Suivez les étapes de ce module pour exécuter un client WebSphere eXtreme Scale qui insère des données dans la grille.

Objectifs d'apprentissage

A la fin des leçons de ce module, vous saurez :

- exécuter une application client qui se connecte à la grille et y insère et en extrait des données ;
- démarrer une commande en utilisant l'application client non-OSGi.

Prerequis

Exécutez le Module 2 : Installation et démarrage des ensembles eXtreme Scale dans l'infrastructure OSGi.

Leçon 3.1 : Configuration d'Eclipse pour exécuter le client et créer les exemples

Effectuez cette leçon pour importer le projet Eclipse que vous utiliserez pour exécuter le client et générer les exemples de plug-ins.

L'exemple inclut un programme client Java SE qui se connecte à la grille, insère des données et en extrait. Il contient également des projets que vous pouvez utiliser pour générer et redéployer les ensembles OSGi.

Le projet fourni a été testé avec Eclipse 3.x et les versions suivantes et ne nécessite que la perspective du projet de développement Java. Procédez comme suit pour configurer votre environnement de développement WebSphere eXtreme Scale.

1. Ouvrez Eclipse dans un espace de travail nouveau ou existant.
2. Dans le menu Fichier, sélectionnez **Importer**.
3. Développez le dossier Général. Sélectionnez **Projets existants dans l'espace de travail** et cliquez sur **Suivant**.
4. Dans la zone **Sélectionner le répertoire racine**, tapez le répertoire *wxs_sample_osgi_root* ou accédez-y. Cliquez sur **Terminer**. Plusieurs nouveaux projets sont affichés dans votre espace de travail. Vous devez corriger plusieurs erreurs de génération en définissant la bibliothèque utilisateur eXtreme Scale. Procédez comme suit pour définir la bibliothèque utilisateur.
5. Dans le menu Fenêtre, sélectionnez **Préférences**.
6. Développez la branche **Java** > **Chemin de génération** et sélectionnez **Bibliothèques utilisateur**.
7. Cliquez sur **Nouveau**.
8. Tapez `eXtremeScale` dans la zone du **nom de bibliothèque utilisateur** et cliquez sur **OK**.
9. Sélectionnez la nouvelle bibliothèque utilisateur et cliquez **Ajouter des fichiers JAR**.
   a. Recherchez et sélectionnez le fichier `objectgrid.jar` dans le répertoire *wxs_install_root*`/lib`. Cliquez sur **OK**.
   b. Pour inclure la documentation d'API ObjectGrid, sélectionnez l'emplacement de la documentation d'API pour le fichier `objectgrid.jar` que vous avez ajouté à l'étape précédente. Cliquez sur **Editer**.

c. Dans la zone Chemin d'emplacement de la documentation d'API, sélectionnez le fichier `Javadoc.zip` qui se trouve dans le répertoire *wxs_install_root*`/docs/javadoc.zip`.

Point de contrôle de la leçon :

Dans cette leçon, vous avez importé l'exemple de projet Eclipse, défini par la bibliothèque utilisateur eXtreme Scale, et inclus la documentation d'API correspondante pour l'exemple de projet. Maintenant, vous pouvez démarrer l'exemple d'application client.

Leçon 3.2 : Démarrage d'un client et insertion de données dans la grille

Etudiez cette leçon pour démarrer un client et exécuter une application client.

L'application client Java est `com.ibm.websphere.samples.xs.proto.client.Client`.

Ce client utilise une substitution de client, le fichier descripteur XML ObjectGrid pour remplacer la configuration OSGi, afin que le client puisse s'exécuter dans un environnement non-OSGi. Voir le contenu suivant du fichier avec les commentaires et les en-têtes supprimés. Certaines lignes de code sont réparties sur plusieurs lignes à des fins de formatage.

```
<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" txTimeout="15">
            <bean id="ObjectGridEventListener" className="" osgiService=""/>
            <backingMap name="Map" readOnly="false"
                lockStrategy="PESSIMISTIC" lockTimeout="5"
                copyMode="COPY_TO_BYTES" pluginCollectionRef="serializer"/>

        </objectGrid>
    </objectGrids>

    <backingMapPluginCollections>
        <backingMapPluginCollection id="serializer">

   <bean id="MapSerializer"
   className="com.ibm.websphere.samples.xs.serializer.proto.ProtoMapSerializer"
       osgiService="">
    <property name="keyType" type="java.lang.String"
        value="com.ibm.websphere.samples.xs.serializer.proto.DataObjects2$OrderKey" />
          <property name="valueType" type="java.lang.String"
              value="com.ibm.websphere.samples.xs.serializer.proto.DataObjects2$Order" />
   </bean>
  </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

Procédez comme suit pour démarrer l'application client.

1. Utilisez l'exemple de code suivant pour modifier les attributs de la classe Client pour refléter votre environnement.

   ```
   private String catHost = "localhost";
   private int catListenerPort = 2809;
   private String clientOGXML = "wxs_sample_osgi_root/client/META-INF/
   clientProtoBufObjectgrid.xml";
   private String gridName = "Grid";
   private String mapName = "Map";
   ```

2. Exécutez l'application client.

Lorsque vous exécutez l'application, le message suivant est affiché. Le message indique qu'une commande a été insérée :

```
order
com.ibm.websphere.samples.xs.serializer.proto.DataObjects1$Or
der$Builder@5d165d16(5000000) inserted
```

Point de contrôle de la leçon :

Dans cette leçon, vous avez démarré l'application `com.ibm.websphere.samples.xs.proto.client.Client` qui a généré une commande.

Module 4: Interrogation et mise à niveau de l'exemple d'ensemble

Suivez les leçons de ce module pour utiliser la commande **xscmd** pour interroger le classement de services de l'exemple d'ensemble, mettez-le à niveau vers un nouveau classement de services et vérifiez ce dernier.

Un projet Eclipse est fourni pour faciliter l'exécution des exemples d'applications.

Objectifs d'apprentissage

Après avoir suivi les leçons de ce module, vous saurez :

- interroger le classement de services en cours d'un service ;
- interroger le classement en cours de tous les services ;
- interroger tous les classements disponibles d'un service ;
- interroger tous les classements de services disponibles ;
- utiliser l'outil xscmd pour déterminer les classements de services disponibles ;
- mettre à jour les classements des exemples de services OSGi.

Prérequis

Exécutez le Module 3 : Exécution de l'exemple de client eXtreme Scale.

Leçon 4.1 : recherche des classements de services

Etudiez cette leçon pour identifier les classements de services en cours ainsi que les classements de services qui sont disponibles pour la mise à niveau.

- Identifiez le classement de service en cours d'un service. Entrez la commande suivante pour interroger le classement de service en cours utilisé pour le service myShardListener utilisée par la grille ObjectGrid `Grid` et le groupe de mappes `MapSet`.
  1. Accédez au répertoire suivant :
     ```
     cd wxs_home/bin
     ```
  2. Entrez la commande suivante pour interroger le classement en cours du service, myShardListener.
     ```
     ./xscmd.sh -c osgiCurrent -g Grid -ms MapSet -sn myShardListener
     ```

     La sortie suivante s'affiche :
     ```
     OSGi Service Name: myShardListener
     ObjectGrid Name MapSet Name Server Name        Current Ranking
     --------------- ----------- -----------        ---------------
     Grid            MapSet      collocatedServer   1

     CWXSI0040I: The command osgiCurrent has completed successfully.
     ```
- Identifiez le classement en cours de tous les services. Entrez la commande suivante pour interroger le classement de service en cours utilisé pour tous les services utilisé par la grille ObjectGrid `Grid` et le groupe de mappes `MapSet`.

1. Accédez au répertoire suivant :

   ```
   cd wxs_home/bin
   ```

2. Entrez la commande suivante pour interroger les classements de tous les services.

   ```
   ./xscmd.sh -c osgiCurrent -g Grid -ms MapSet
   ```

   La sortie suivante s'affiche :

```
OSGi Service Name    Current Ranking ObjectGrid Name MapSet Name Server Name
-----------------    --------------- --------------- ----------- -----------
myProtoBufSerializer 1               Grid            MapSet      collocatedServer
myShardListener      1               Grid            MapSet      collocatedServer

CWXSI0040I: The command osgiCurrent has completed successfully.
```

- Identifiez tous les classements disponibles d'un service. Entrez la commande suivante pour interroger les classements du service myShardListener.
  1. Accédez au répertoire suivant :

     ```
     cd wxs_home/bin
     ```

  2. Entrez la commande suivante pour interroger tous les classements d'un service.

     ```
     ./xscmd.sh -c osgiAll -sn myShardListener
     ```

     La sortie suivante s'affiche :

     ```
     Server: collocatedServer
        OSGi Service Name Available Rankings
        ----------------- ------------------
        myShardListener 1

        Summary - All servers have the same service rankings.

     CWXSI0040I: The command osgiAll has completed successfully.
     ```

     La sortie est regroupée en fonction du serveur. Dans cet exemple, seul le serveur collocatedServer existe.
- Identifiez tous les classements de services disponibles. Entrez la commande suivante pour rechercher tous les classements de tous les services.
  1. Accédez au répertoire suivant :

     ```
     cd wxs_home/bin
     ```

  2. Entrez la commande suivante pour rechercher tous les classements de services disponibles.

     ```
     ./xscmd.sh -c osgiAll
     ```

     La sortie suivante s'affiche :

     ```
     Server: collocatedServer
        OSGi Service Name    Available Rankings
        -----------------    ------------------
        myProtoBufSerializer 1
        myShardListener 1

        Summary - All servers have the same service rankings.
     ```
- Installez et démarrez la version 2 de l'ensemble de plug-in. Dans la console OSGi du serveur, installez un nouvel ensemble contenant une nouvelle version de la classe Order et le plug-in MapSerializerPlugin. Voir Leçon 2.4 : Installation de Google Protocol et exemples d'ensembles de plug-ins pour plus d'informations sur l'installation de l'ensemble `ProtoBufSamplePlugins-2.0.0.jar`.
  1. Après l'installation, démarrez le nouvel ensemble. Les services du nouvel ensemble sont disponibles, mais ils ne sont pas encore utilisés par le serveur

eXtreme Scale. Vous devez exécuter une demande de mise à jour de service pour pouvoir utiliser un service avec une version donnée.

- Maintenant, lorsque vous interrogez de nouveau tous les classements de services disponibles, le classement de service 2 est ajouté dans la sortie.

1. Accédez au répertoire suivant :

```
cd wxs_home/bin
```

2. Entrez la commande suivante pour rechercher tous les classements de services disponibles.

```
./xscmd.sh -c osgiAll
```

La sortie suivante s'affiche :

```
Server: collocatedServer
   OSGi Service Name    Available Rankings
   -----------------    ------------------
   myProtoBufSerializer 1, 2
   myShardListener      1, 2

Summary - All servers have the same service rankings.
```

Point de contrôle de la leçon :

Dans ce tutoriel, vous avez interrogé des classements de services spécifiques et les classements de services disponibles. Vous avez également affiché le classement de service d'un nouvel ensemble que vous avez installé et démarré.

Leçon 4.2 : Déterminer si des classements de services spécifiques sont disponibles

Suivez cette leçon pour déterminer si des classements de services spécifiques sont disponibles pour les noms de service que vous spécifiez.

1. Entrez la commande suivante pour déterminer si le service myShardListener, avec le classement de service 2 et le service myProtoBufSerializer avec le classement de service 2 sont disponibles. La liste des classements de service est transmise à l'aide de l'option `-sr`.

a. Accédez au répertoire suivant :

```
cd wxs_home/bin
```

b. Entrez la commande suivante pour déterminer si les services sont disponibles :

```
./xscmd.sh -c osgiCheck -g Grid -ms MapSet -sr
"myShardListener;2,myProtoBufSerializer;2"
```

La sortie suivante s'affiche :

```
CWXSI0040I: The command osgiCheck has completed successfully.
```

2. Entrez la commande suivante pour déterminer si le service myShardListener, avec le classement de service 2 et le service myProtoBufSerializer avec le classement de service 3 sont disponibles.

a. Accédez au répertoire suivant :

```
cd wxs_home/bin
```

b. Entrez la commande suivante pour déterminer si les services sont disponibles :

```
./xsadmin.sh -c osgiCheck -g Grid -ms MapSet -sr
"myShardListener;2,myProtoBufSerializer;3"
```

La sortie suivante s'affiche :

```
Server OSGi Service Unavailable Rankings
------ ------------ --------------------
collocatedServer myProtoBufSerializer 3
```

Point de contrôle de la leçon :

Dans cette leçon, vous avez spécifié les services myShardListener et myProtoBufSerializer, ainsi que des classements de services spécifiques pour déterminer si ces classements étaient disponibles.

Leçon 4.3 : Mise à jour des classements de services

Suivez cette leçon pour mettre à jour les classements de services en cours que vous avez interrogés.

1. La commande suivante met à jour les classements de services myShardListener et myProtoBufSerializer vers le classement de services 2. La liste des classements de services est envoyée en utilisant l'option `-sr`.
   a. Accédez au répertoire suivant :
      ```
      cd wxs_home/bin
      ```
   b. Entrez la commande suivante pour mettre à jour les classements de services :
      ```
      ./xscmd.sh -c osgiUpdate -g Grid -ms MapSet
      -sr "myShardListener;2,myProtoBufSerializer;2"
      ```

      La sortie suivante s'affiche :
      ```
      Update succeeded for the following service rankings:
      Service Ranking
      ------- -------
      myProtoBufSerializer 2
      myShardListener 2

      CWXSI0040I: The command osgiUpdate has completed successfully.
      ```

      La sortie suivante s'affiche sur la console OSGi :
      ```
      SystemOut O MyShardListener@326505334(version=2.0.0) order
      com.ibm.websphere.samples.xs.serializer.proto.DataObjects2$Order$Builder@
      22342234(34) updated
      ```

      Notez que le service MyShardListener est maintenant au niveau de version 2.0.0 qui a le classement de service 2.
2. Exécutez la commande **xscmd** pour interroger le classement de services en cours utilisé pour tous les services qui sont utilisés par l'ObjectGrid `Grid` et le groupe de mappes `MapSet`.
   a. Accédez au répertoire suivant :
      ```
      cd wxs_home/bin
      ```
   b. Entrez la commande suivante pour interroger les classements de tous les services qui sont utilisés par `Grid` et `MapSet`:
      ```
      ./xscmd.sh -c osgiCurrent -g Grid -ms MapSet
      ```

      La sortie suivante s'affiche :
      ```
      OSGi Service Name Current Ranking ObjectGrid Name MapSet Name Server Name
      ----------------- --------------- --------------- ----------- -----------
      myProtoBufSerializer 2 Grid MapSet collocatedServer
      myShardListener 2 Grid MapSet collocatedServer

      CWXSI0040I: The command osgiCurrent has completed successfully.
      ```

Point de contrôle de la leçon :

Dans cette leçon, vous avez mis à jour les classements des services myShardListener et myProtoBufSerializer.

.

Chapitre 2. Scénarios

Un scénario présente des exemples pour aider l'utilisateur à comprendre un concept ou à accomplir une tâche. Le scénario utilise des informations du monde réel pour construire une image complète.

Utilisation d'un environnement OSGi pour développer et exécuter les plug-ins eXtreme Scale

Utilisez ces scénarios pour effectuer les tâches courantes dans un environnement OSGi. Par exemple, l'infrastructure OSGi est idéale pour le démarrage des serveurs et des clients dans un conteneur OSGi, ce qui vous permet d'ajouter et de mettre à jour dynamiquement les plug-ins WebSphere eXtreme Scale dans l'environnement d'exécution.

Les scénarios suivants concernent la création et l'exécution de plug-ins dynamiques, ce qui permet d'installer, de démarrer, d'arrêter, de modifier et de désinstaller dynamiquement des plug-ins dynamiques. Vous pouvez également suivre un autre scénario probable, ce qui permet d'utiliser la structure OSGi sans fonctions dynamiques. Vous pouvez toujours modulariser vos applications comme des ensembles définis par et communiqués via des services. Ces ensembles basés sur des services offrent plusieurs avantages, notamment des fonctions de développement et de déploiement plus efficaces.

Objectifs des scénarios

Après avoir suivi les leçons de ce module, vous saurez :

- Générer des plug-ins dynamiques eXtreme Scale utilisables dans un environnement OSGi.
- Exécuter de conteneurs eXtreme Scale dans un environnement OSGi sans fonctions dynamiques.

Conditions prérequises

Consultez la rubrique «Présentation de l'infrastructure OSGi» pour en savoir plus sur la prise en charge d'OSGi et ses avantages.

Présentation de l'infrastructure OSGi

OSGi définit un système de module dynamique pour Java. La plateforme de service OSGi présente une architecture à couches et elle est conçue pour s'exécuter dans plusieurs profils standard Java. Vous pouvez démarrer les serveurs et les clients WebSphere eXtreme Scale dans un conteneur OSGi.

Avantages de l'exécution des applications dans le conteneur OSGi

Le support WebSphere eXtreme Scale OSGi permet de déployer le produit dans l'infrastructure OSGi Eclipse Equinox. Auparavant, si vous souhaitiez mettre à niveau les plug-ins utilisés par eXtreme Scale, vous deviez redémarrer la machine virtuelle Java (JVM) pour appliquer les nouvelles versions des plug-ins. Avec le support de plug-ins dynamiques fourni par l'infrastructure OSGi, vous pouvez

désormais mettre à jour les classes de plug-in sans redémarrer la machine virtuelle Java. Ces plug-ins sont exportés par ensembles d'utilisateur comme services. WebSphere eXtreme Scale accède au(x) service(s) en les recherchant dans le registre OSGi.

Les conteneurs eXtreme Scale peuvent être configurés pour démarrer plus aisément et dynamiquement le service d'administration de configuration OSGi ou avec OSGi Blueprint. Si vous voulez déployer une nouvelle grille avec sa stratégie de placement, vous pouvez le faire en créant une configuration OSGi ou en déployant un ensemble avec des fichiers descripteurs XML eXtreme Scale. Avec le support OSGi, les ensembles d'applications contenant des données de configuration eXtreme Scale peuvent être installés, démarrés, mis à jour et désinstallés sans redémarrer tout le système. Avec cette fonction, vous pouvez mettre à niveau l'application sans perturber la grille de données.

Les beans de plug-in et les services peuvent être configurés avec des portées de fragment personnalisées pour permettre une intégration précise d'options aux autres services exécutés dans la grille. Chaque plug-in peut utiliser des classements OSGi Blueprint pour vérifier que chaque instance activée du plug-in correspond au niveau de version correct. Un bean géré OSGi (MBean) et l'utilitaire **xscmd** sont fournis pour vous permettre d'exécuter des requêtes sur les services OSGi de plug-in eXtreme Scale et leurs classements.

Avec cette fonction, les administrateurs peuvent identifier rapidement les erreurs potentielles de configuration et d'administration et mettre à jour les classements de service de plug-in utilisés par eXtreme Scale .

Ensembles OSGi

Pour interagir avec les plug-ins et déployer des plug-ins dans l'infrastructure OSGi, vous devez utiliser des *ensembles*. Dans la plateforme de service OSGi, un ensemble est un fichier d'archive JAR (Java archive) qui contient du code Java, des ressources et un manifeste qui décrit l'ensemble et ses dépendances. L'ensemble représente l'unité de déploiement d'une application. Le produit eXtreme Scale prend en charge les types d'ensembles suivants :

Ensemble de serveur
L'ensemble de serveur est le fichier `objectgrid.jar`. Il est installé avec le serveur autonome eXtreme Scale et il est nécessaire pour exécuter les serveurs eXtreme Scale. Vous pouvez également l'utiliser pour exécuter les clients eXtreme Scale ou les mémoires caches internes locales. L'ID d'ensemble pour le fichier `objectgrid.jar` est com.ibm.websphere.xs.server_`<version>`, où la version a le format `<Version>.<Release>.<Modification>`. Par exemple, l'ensemble de serveur pour eXtreme Scale version 7.1.1 est com.ibm.websphere.xs.server_7.1.1.

Ensemble de client
L'ensemble de client est le fichier `ogclient.jar`. Il est installé en même temps que les installations client et autonome eXtreme Scale et il est utilisé pour exécuter les clients eXtreme Scale ou les mémoires caches internes locales. L'ID d'ensemble du fichier `ogclient.jar` est com.ibm.websphere.xs.client_`version`, où la version a le format `<Version>.<Release>.<Modification>`. Par exemple, l'ensemble de client pour eXtreme Scale version 7.1.1 est com.ibm.websphere.xs.client_7.1.1.

Limitations

Vous ne pouvez pas redémarrer l'ensemble eXtreme Scale, car vous ne pouvez pas redémarrer l'ORB (object request broker). Pour redémarrer le serveur eXtreme Scale, vous devez redémarrer l'infrastructure OSGi.

Tâches associées:

«Installation de l'infrastructure OSGi Eclipse Equinox avec Eclipse Gemini pour les clients et les serveurs»
Si vous souhaitez déployer WebSphere eXtreme Scale dans la structure OSGi, vous devez configurer l'environnement Eclipse Equinox.

«Gestion des cycles de vie du plug-in», à la page 298
Vous pouvez gérer les cycles de vie de plug-in avec des méthodes spécialisées de chaque plug-in, qui peuvent être appelées à des points fonctionnels déterminés. Les deux méthodes initialize et destroy définissent le cycle de vie des plug-ins qui sont contrôlés par leurs objets *propriétaire*. 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.

Référence associée:

Fichier de propriétés du serveur
Le fichier de propriétés du serveur contient plusieurs propriétés qui définissent différents paramètres pour votre serveur, tels que les paramètres de trace, la consignation et la configuration de la sécurité. Le fichier de propriétés du serveur est utilisé par les services de catalogue et les serveurs de conteneur dans les serveurs autonomes et les serveurs hébergés dans WebSphere Application Server.

Information associée:

«Introduction : Démarrage et configuration du serveur eXtreme Scale et du conteneur pour exécuter les plug-ins dans la structure OSGi», à la page 19
Dans ce tutoriel, vous allez démarrer un serveur eXtreme Scale dans l'infrastructure OSGi, démarrer un conteneur eXtreme Scale et connecter les exemples de plug-ins avec l'environnement d'exécution eXtreme Scale.

documentation sur les API

Installation de l'infrastructure OSGi Eclipse Equinox avec Eclipse Gemini pour les clients et les serveurs

Si vous souhaitez déployer WebSphere eXtreme Scale dans la structure OSGi, vous devez configurer l'environnement Eclipse Equinox.

Pourquoi et quand exécuter cette tâche

La tâche nécessite que vous téléchargiez et installiez l'infrastructure Blueprint qui permet de configurer ensuite les JavaBeans et de les exposer en tant que services. L'utilisation de services est importante, car vous pouvez exposer des plug-ins en tant que services OSGi pour qu'ils puissent être utilisés par l'environnement d'exécution eXtreme Scale. Le produit prend en charge deux conteneurs Blueprint dans l'infrastructure OSGi principale Eclipse Gemini et Apache Aries. Utilisez cette procédure pour configurer le conteneur Gemini Eclipse.

Procédure

1. Téléchargez Eclipse Equinox SDK Version 3.6.1 ou la version suivante à partir du site Web Eclipse. Créez un répertoire pour l'infrastructure Equinox, par exemple, `/opt/equinox`. Ces instructions font référence à ce répertoire sous la forme `equinox_root`. Extrayez le fichier compressé dans le répertoire `equinox_root`.

2. Téléchargez le fichier compressé gemini-plan d'incubation 1.0.0 depuis le site Web Eclipse. Extrayez le contenu du fichier dans un répertoire temporaire et copiez les fichiers extraits suivants vers le répertoire `equinox_root/plugins` :

   ```
   dist/gemini-blueprint-core-1.0.0.jar
   dist/gemini-blueprint-extender-1.0.0.jar
   dist/gemini-blueprint-io-1.0.0.jar
   ```

3. Téléchargez Spring Framework Version 3.0.5 à partir de la page Web SpringSource http://www.springsource.com/download/community. Extrayez le contenu du fichier dans un répertoire temporaire et copiez les fichiers extraits suivants vers le répertoire `equinox_root/plugins` :

   ```
   org.springframework.aop-3.0.5.RELEASE.jar
   org.springframework.asm-3.0.5.RELEASE.jar
   org.springframework.beans-3.0.5.RELEASE.jar
   org.springframework.context-3.0.5.RELEASE.jar
   org.springframework.core-3.0.5.RELEASE.jar
   org.springframework.expression-3.0.5.RELEASE.jar
   ```

4. Téléchargez le fichier AOP Alliance Java archive (JAR) depuis la page Web SpringSource. Copiez com.springsource.org.aopalliance-1.0.0.jar vers le répertoire `equinox_root/plugins` .

5. Téléchargez le fichier JAR Apache commons logging 1.1.1 JAR depuis la page Web SpringSource. Copiez le fichier `com.springsource.org.apache.commons.logging-1.1.1.jar` vers le répertoire `equinox_root/plugins`.

6. Téléchargez le client de ligne de commande Luminis OSGi Configuration Admin. Utilisez cet ensemble pour gérer les configurations d'administration OSGi. Vous pouvez télécharger le fichier JAR depuis la page Web https://opensource.luminis.net/wiki/display/SITE/OSGi+Configuration+Admin+command+line+client. Copiez le fichier `net.luminis.cmc-0.2.5.jar` vers le répertoire `equinox_root/plugins`.

7. Téléchargez l'ensemble Apache Felix file installation Version 3.0.2 depuis la page Web http://felix.apache.org/site/index.html. Copiez le fichier `org.apache.felix.fileinstall-3.0.2.jar` vers le répertoire `equinox_root/plugins`.

8. Créez un répertoire de configuration dans le répertoire `equinox_root/plugins`, par exemple :

   ```
   mkdir equinox_root/plugins/configuration
   ```

9. Créez le fichier `config.ini` suivant dans le répertoire `equinox_root/plugins/configuration` en remplaçant *equinox_root* par le chemin absolu dans le chemin du répertoire `equinox_root` en supprimant tous les espaces après la barre oblique inverse dans chaque ligne. Vous devez placer une ligne blanche à la fin du fichier, par exemple :

   ```
   osgi.noShutdown=true
   osgi.java.profile.bootdelegation=none
   org.osgi.framework.bootdelegation=none
   eclipse.ignoreApp=true
   osgi.bundles=\
   org.eclipse.osgi.services_3.2.100.v20100503.jar@1:start, \
   org.eclipse.osgi.util_3.2.100.v20100503.jar@1:start, \
   org.eclipse.equinox.cm_1.0.200.v20100520.jar@1:start, \
   com.springsource.org.apache.commons.logging-1.1.1.jar@1:start, \
   com.springsource.org.aopalliance-1.0.0.jar@1:start, \
   org.springframework.aop-3.0.5.RELEASE.jar@1:start, \
   org.springframework.asm-3.0.5.RELEASE.jar@1:start, \
   org.springframework.beans-3.0.5.RELEASE.jar@1:start, \
   org.springframework.context-3.0.5.RELEASE.jar@1:start, \
   org.springframework.core-3.0.5.RELEASE.jar@1:start, \
   org.springframework.expression-3.0.5.RELEASE.jar@1:start, \
   org.apache.felix.fileinstall-3.0.2.jar@1:start, \
   net.luminis.cmc-0.2.5.jar@1:start, \
   gemini-blueprint-core-1.0.0.jar@1:start, \
   gemini-blueprint-extender-1.0.0.jar@1:start, \
   gemini-blueprint-io-1.0.0.jar@1:start
   ```

Si vous avez déjà configuré l'environnement, vous pouvez nettoyer le référentiel de plug-in Equinox en supprimant le répertoire `equinox_root\plugins\configuration\org.eclipse.osgi`.

10. Exécutez la commande suivante pour démarrer la console Equinox.

    Si vous exécutez une version différente d'Equinox, le nom du fichier JAR est différent de celui de l'exemple ci-dessous :

    ```
    java -jar plugins\org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
    ```

Concepts associés:

«Présentation de l'infrastructure OSGi», à la page 37
OSGi définit un système de module dynamique pour Java. La plateforme de service OSGi présente une architecture à couches et elle est conçue pour s'exécuter dans plusieurs profils standard Java. Vous pouvez démarrer les serveurs et les clients WebSphere eXtreme Scale dans un conteneur OSGi.

Référence associée:

Fichier de propriétés du serveur
Le fichier de propriétés du serveur contient plusieurs propriétés qui définissent différents paramètres pour votre serveur, tels que les paramètres de trace, la consignation et la configuration de la sécurité. Le fichier de propriétés du serveur est utilisé par les services de catalogue et les serveurs de conteneur dans les serveurs autonomes et les serveurs hébergés dans WebSphere Application Server.

Information associée:

«Introduction : Démarrage et configuration du serveur eXtreme Scale et du conteneur pour exécuter les plug-ins dans la structure OSGi», à la page 19
Dans ce tutoriel, vous allez démarrer un serveur eXtreme Scale dans l'infrastructure OSGi, démarrer un conteneur eXtreme Scale et connecter les exemples de plug-ins avec l'environnement d'exécution eXtreme Scale.

Installation des ensembles eXtreme Scale

WebSphere eXtreme Scale inclut des ensembles qui peuvent être installés dans une infrastructure OSGi Eclipse Equinox. Ces ensembles sont nécessaires pour démarrer les serveurs eXtreme Scale ou utiliser les clients eXtreme Scale dans OSGi.

Avant de commencer

Cette tâche suppose que les produits suivants ont été installés :

- Infrastructure OSGi Eclipse Equinox
- Client ou serveur autonome eXtreme Scale

Pourquoi et quand exécuter cette tâche

eXtreme Scale inclut deux ensembles. Un seul des ensembles suivants est nécessaire dans une infrastructure OSGi :

objectgrid.jar
L'ensemble de serveur est le fichier `objectgrid.jar`. Il est installé avec l'installation de serveur autonome eXtreme Scale et il est nécessaire pour exécuter les serveurs eXtreme Scale servers. Il peut être aussi utilisé pour exécuter les clients eXtreme Scale ou les mémoires caches internes locales. L'ID d'ensemble du fichier `objectgrid.jar` est com.ibm.websphere.xs.server_`<version>`, où la version a le format `<Version>.<Release>.<Modification>`. Par exemple, l'ensemble de serveur pour eXtreme Scale version 7.1.1 est com.ibm.websphere.xs.server_7.1.1.

ogclient.jar

L'ensemble `ogclient.jar` est installé avec les installations client et autonomes eXtreme Scale et il est utilisé pour exécuter les clients eXtreme Scale ou les mémoires caches internes locales. L'ID d'ensemble du fichier `ogclient.jar` est com.ibm.websphere.xs.client_<version>, où la version a le format <Version>_<Release>_<Modification. Par exemple, l'ensemble client pour eXtreme Scale Version 7.1.1 est com.ibm.websphere.xs.client_7.1.1.

Pour plus d'informations sur le développement de plug-ins eXtreme Scale, voir la rubrique API système et plug-ins.

Procédure

Pour installer l'ensemble client ou serveur eXtreme Scale dans l'infrastructure OSGi Eclipse Equinox en utilisant la console OSGi :

1. Démarrez l'infrastructure Eclipse Equinox avec la console activée. Par exemple :
   ```
   rép_base_java/bin/java -jar <equinox_root>/plugins/
   org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
   ```
2. Installez l'ensemble serveur ou client eXtreme Scale dans la console Equinox :
   ```
   osgi> install file:///<path to bundle>
   ```
3. Equinox affiche l'ID d'ensemble du nouvel ensemble installé :
   ```
   Bundle id is 25
   ```
4. Démarrez l'ensemble dans la console Equinox, où <id> est l'ID affecté à l'ensemble lors de son installation :
   ```
   osgi>  start <id>
   ```
5. Extrayez l'état du service dans la console Equinox pour vérifier que l'ensemble a démarré. Par exemple :
   ```
   osgi> ss
   ```

   Lorsque l'ensemble a démarré correctement, il affiche l'état ACTIVE, par exemple :
   ```
   25      ACTIVE      com.ibm.websphere.xs.server_7.1.1
   ```

Installez l'ensemble client ou serveur eXtreme Scale dans l'infrastructure OSGi Eclipse Equinox en utilisant le fichier `config.ini` :

6. Copiez l'ensemble client ou serveur eXtreme Scale (objectgrid.jar ou ogclient.jar) de <wxs_install_root>/ObjectGrid/lib vers le répertoire Eclipse Equinox, par exemple : <equinox_root>/plugins
7. Modifiez le fichier de configuration Eclipse Equinox `config.ini` et ajoutez l'ensemble à la propriété osgi.bundles, par exemple :
   ```
   osgi.bundles=\
   org.eclipse.osgi.services_3.2.100.v20100503.jar@1:start, \
   org.eclipse.osgi.util_3.2.100.v20100503.jar@1:start, \
   org.eclipse.equinox.cm_1.0.200.v20100520.jar@1:start, \
   objectgrid.jar@1:start
   ```

   Important : Vérifiez qu'une ligne blanche existe après le dernier nom d'ensemble. Chaque ensemble est séparé par une virgule.
8. Démarrez l'infrastructure Eclipse Equinox avec la console activée. Par exemple :
   ```
   rép_base_java/bin/java -jar <equinox_root>/plugins/
   org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
   ```
9. Extrayez l'état du service dans la console Equinox pour vérifier que l'ensemble a démarré :

```
osgi> ss
```

Lorsque l'ensemble a démarré correctement, il affiche l'état ACTIVE, par exemple :

```
25      ACTIVE      com.ibm.websphere.xs.server_7.1.1
```

Résultats

L'ensemble client ou serveur eXtreme Scale est installé et démarré dans l'infrastructure OSGi Eclipse Equinox.

Génération et exécution de plug-ins dynamiques eXtreme Scale pour une utilisation dans un environnement OSGi

Tous les plug-ins eXtreme Scale peuvent être configurés pour un environnement OSGi. Les plug-ins dynamiques offrent pour principal avantages de pouvoir les mettre à niveau sans fermer la grille. Cela permet de faire évoluer une application sans redémarrer les processus conteneur de la grille.

Pourquoi et quand exécuter cette tâche

Le support OSGi WebSphere eXtreme Scale permet de déployer le produit dans une infrastructure OSGi, telle que Eclipse Equinox. Auparavant, si vous souhaitiez mettre à niveau les plug-ins utilisés par eXtreme Scale, vous deviez redémarrer la machine virtuelle Java (JVM) pour appliquer les nouvelles versions des plug-ins. Avec le support des plug-ins dynamiques fourni par eXtreme Scale et la possibilité de mettre à jour les ensembles que l'infrastructure OSGi fournit, vous pouvez désormais mettre à jour les classes de plug-in sans redémarrer la machine JVM. Ces plug-ins sont exportés par *ensembles* comme services. WebSphere eXtreme Scale accède au service en consultant le registre OSGi. Dans la plateforme de service OSGi, un ensemble est un fichier archive Java (JAR) qui contient du code Java, des ressources et un manifeste qui décrit le regroupement et ses dépendances. L'ensemble représente l'unité de déploiement d'une application.

Procédure

1. Créer des plug-ins dynamiques eXtreme Scale.
2. Configurer les plug-ins eXtreme Scale avec OSGi Blueprint.
3. Installer et démarrer les plug-ins OSGi.

Génération de plug-ins dynamiques eXtreme Scale

WebSphere eXtreme Scale inclut les plug-ins ObjectGrid et BackingMap. Ces plug-ins sont implémentés dans Java et configurés en utilisant le fichier XML descripteur ObjectGrid. Pour créer un plug-in dynamique qui peut être mis à niveau dynamiquement, il doit connaître les événements de cycle de vie ObjectGrid et BackingMap, car il peut être nécessaire qu'il exécute des actions lors d'une mise à jour. L'amélioration d'un module d'extension avec des méthodes de rappel de cycle de vie, des programmes d'écoute d'événements ou les deux permet aux plug-ins d'effectuer ces actions au moment opportun.

Avant de commencer

Cette rubrique suppose que vous avez créé le plug-in approprié. Pour plus d'informations sur le développement de plug-ins eXtreme Scale, voir la rubrique API système et plug-ins.

Pourquoi et quand exécuter cette tâche

Tous les plug-ins eXtreme Scale s'appliquent à une instance BackingMap ou ObjectGrid. De nombreux plug-ins interagissent également avec d'autres plug-ins. Par exemple, un chargeur et un plug-in TransactionCallback fonctionnent ensemble pour interagir correctement avec une transaction de base de données et les divers appels JDBC de base de données. Certains modules d'extension peut également s'avérer nécessaires pour mettre en mémoire cache les données de configuration des autres plug-ins pour améliorer les performances.

Les plug-ins BackingMapLifecycleListener et ObjectGridLifecycleListener fournissent des opérations de cycle de vie pour les instances BackingMap et ObjectGrid correspondantes. Ce processus permet aux plug-ins d'être avertis lorsque le parent BackingMap ou ObjectGrid et ses plug-ins correspondants peuvent être modifiés. Les plug-ins BackingMap implémentent l'interface BackingMapLifecyleListener et les plug-ins ObjectGrid implémentent l'interface ObjectGridLifecycleListener. Ces plug-ins sont appelés automatiquement lorsque le cycle de vie du parent BackingMap ou ObjectGrid change. Pour plus d'informations sur les plug-ins de cycle de vie, voir la rubrique «Gestion des cycles de vie du plug-in», à la page 298.

Vous améliorerez les ensembles en utilisant les méthodes de cycle de vie ou les programmes d'écoute dans les tâches communes suivantes :

- Démarrage et arrêt des ressources, telles que les unités d'exécution ou les abonnés de messagerie.
- Indication qu'une notification se produit lorsque des plug-ins homologues ont été mis à jour, ce qui permet d'accéder directement au plug-in et de détecter les modifications.

Lorsque vous accédez directement à un autre plug-in, accédez-y via le conteneur OSGi pour que tous les composants du système fassent référence au plug-in correct. Si, par exemple, un composant dans l'application fait directement référence, ou met en mémoire cache, une instance d'un plug-in, il conserve sa référence à cette version du plug-in, même lorsque le plug-in est mis à jour dynamiquement. Ce comportement peut causer des problèmes au niveau de l'application ainsi que des fuites de mémoire. Par conséquent, écrivez le code qui dépend des plug-ins dynamiques qui obtient sa référence en utilisant OSGi, la sémantique getService(). Si l'application doit mettre en cache un ou plusieurs plug-ins, elle écoute les événements de cycle de vie en utilisant les interfaces ObjectGridLifecycleListener et BackingMapLifecycleListener. L'application doit pouvoir également régénérer sa mémoire cache lorsque cela est nécessaire en sécurisant les unités d'exécution.

Tous les plug-ins eXtreme Scale utilisés avec OSGi doivent également implémenter l'interface BackingMapPlugin ou ObjectGridPlugin correspondante. Les nouveaux plug-ins, tels que l'interface MapSerializerPlugin, appliquent cette pratique. Ces interfaces fournissent à l'environnement d'exécution eXtreme Scale et OSGi une interface cohérente pour injecter l'état dans le plug-in et contrôler son cycle de vie.

Utilisez cette tâche pour spécifier qu'une notification se produit lorsque des plug-ins homologues sont mis à jour. Vous pouvez créez une fabrique d'écoute qui produit une instance de programme d'écouter.

Procédure

- Mettez à jour la classe de plug-in ObjectGrid pour implémenter l'interface ObjectGridPlugin. Cette interface contient des méthodes qui permettent à

eXtreme Scale d'initialiser et définir l'instance ObjectGrid et détruire le plug-in. Reportez-vous à l'exemple de code suivant :

```
package com.mycompany;
import com.ibm.websphere.objectgrid.plugins.ObjectGridPlugin;
...

public class MyTranCallback implements TransactionCallback, ObjectGridPlugin {

    private ObjectGrid og = null;

    private enum State {
        NEW, INITIALIZED, DESTROYED
    }

    private State state = State.NEW;

    public void setObjectGrid(ObjectGrid grid) {
        this.og = grid;
    }

    public ObjectGrid getObjectGrid() {
        return this.og;
    }
    void initialize() {
        // Traiter l'initialisation de plug-in ici.  Cela peut être appelé par
        // eXtreme Scale et non pas par le gestionnaire de bean OSGi.
        state = State.INITIALIZED;
    }
    boolean isInitialized() {
        return state == State.INITIALIZED;
    }

    public void destroy() {
        // Détruire le plug-in et libérer les ressources.  Cela
        peut être appelé par le gestionnaire de bean OSGi ou eXtreme Scale.
        state = State.DESTROYED;
    }

    public boolean isDestroyed() {
        return state == State.DESTROYED;
    }
}
```

- Mettez à jour la classe de plug-in ObjectGrid pour implémenter l'interface ObjectGridLifecycleListener. Reportez-vous à l'exemple de code suivant :

```
package com.mycompany;
import com.ibm.websphere.objectgrid.plugins.ObjectGridLifecycleListener;
      import com.ibm.websphere.objectgrid.plugins.ObjectGridLifecycleListener.LifecycleEvent;
      ...

      public class MyTranCallback implements TransactionCallback, ObjectGridPlugin, ObjectGridLifecycleListener{
          public void objectGridStateChanged(LifecycleEvent event) {
              switch(event.getState()) {
              case NEW:
              case DESTROYED:
              case DESTROYING:
              case INITIALIZING:
                  break;
              case INITIALIZED:
                  // Rechercher un chargeur ou un plug-in MapSerializerPlugin en utilisant
                  // OSGi ou directement depuis l'instance ObjectGrid.
                  lookupOtherPlugins()
                  break;
              case STARTING:
              case PRELOAD:
                  break;
              case ONLINE:
                  if (event.isWritable()) {
                      startupProcessingForPrimary();
                  } else {
                      startupProcessingForReplica();
                  }
                  break;
              case QUIESCE:
                  if (event.isWritable()) {
                      quiesceProcessingForPrimary();
                  } else {
                      quiesceProcessingForReplica();
                  }
                  break;
              case OFFLINE:
                  shutdownShardComponents();
                  break;
              }
          }
          ...
      }
```

- Mettez à jour un plug-in BackingMap. Mettez à jour la classe de plug-in BackingMap pour implémenter l'interface de plug-in BackingMap. Cette

interface inclut des méthodes qui permettent à eXtreme Scale d'initialiser, définir l'instance BackingMap et détruire le plug-in. Reportez-vous à l'exemple de code suivant :

```
package com.mycompany;
import com.ibm.websphere.objectgrid.plugins.BackingMapPlugin;
...

public class MyLoader implements Loader, BackingMapPlugin {

    private BackingMap bmap = null;

    private enum State {
        NEW, INITIALIZED, DESTROYED
    }

    private State state = State.NEW;

    public void setBackingMap(BackingMap map) {
        this.bmap = map;
    }

    public BackingMap getBackingMap() {
        return this.bmap;
    }
    void initialize() {
        // Traiter l'initialisation de plug-in ici.  Cela peut être appelé par
        // eXtreme Scale et non pas par le gestionnaire de bean OSGi.
        state = State.INITIALIZED;
    }
    boolean isInitialized() {
        return state == State.INITIALIZED;
    }

    public void destroy() {
        // Détruire le plug-in et libérer les ressources.  Cela
        peut être appelé par le gestionnaire de bean OSGi ou eXtreme Scale.
        state = State.DESTROYED;
    }

    public boolean isDestroyed() {
        return state == State.DESTROYED;
    }
}
```

- Mettez à jour la classe de plug-in BackingMap pour implémenter une interface BackingMapLifecycleListener. Reportez-vous à l'exemple de code suivant :

```
package com.mycompany;

      import com.ibm.websphere.objectgrid.plugins.BackingMapLifecycleListener;
      import com.ibm.websphere.objectgrid.plugins.BackingMapLifecycleListener.LifecycleEvent;
      ...

      public class MyLoader implements Loader, ObjectGridPlugin, ObjectGridLifecycleListener{
          ...
          public void backingMapStateChanged(LifecycleEvent event) {
              switch(event.getState()) {
              case NEW:
              case DESTROYED:
              case DESTROYING:
              case INITIALIZING:
                  break;
              case INITIALIZED:
                  // Rechercher un plug-in MapSerializerPlugin en utilisant
                  // OSGi ou directement depuis l'instance ObjectGrid.
                  lookupOtherPlugins()
                  break;
              case STARTING:
              case PRELOAD:
                  break;
              case ONLINE:
                  if (event.isWritable()) {
                      startupProcessingForPrimary();
                  } else {
                      startupProcessingForReplica();
                  }
                  break;
              case QUIESCE:
                  if (event.isWritable()) {
                      quiesceProcessingForPrimary();
                  } else {
                      quiesceProcessingForReplica();
                  }
                  break;
              case OFFLINE:
                  shutdownShardComponents();
                  break;
              }
          }
          ...
      }
```

Résultats

En implémentant l'interface ObjectGridPlugin ou BackingMapPlugin, eXtreme Scale peut contrôler le cycle de vie du plug-in au moment opportun.

En implémentant l'interface ObjectGridLifecycleListener ou BackingMapLifecycleListener, le plug-in est enregistré automatiquement comme programme d'écoute des événements de cycle de vie ObjectGrid ou BackingMap associés. L'événement INITIALIZING permet de signaler que tous les plug-ins ObjectGrid et BackingMap ont été initialisés et peuvent être recherchés et utilisés. L'événement ONLINE est utilisé pour signaler que ObjectGrid est en ligne et prêt à commencer le traitement des événements.

Configuration des plug-ins eXtreme Scale avec OSGi Blueprint

Tous les plug-ins eXtreme Scale ObjectGrid et BackingMap peuvent être définis comme beans et services OSGi en utilisant le service OSGi Blueprint disponible avec Eclipse Gemini ou Apache Aries.

Avant de commencer

Pour pouvoir configurer vos plug-ins comme services OSGi, vous devez regrouper les plug-ins dans un ensemble OSGi et connaître les concepts de base des plug-ins requis. L'ensemble doit importer les modules client ou serveur WebSphere eXtreme Scale et d'autres packages dépendants nécessaires aux plug-ins ou créer une dépendance d'ensemble dans les ensembles de serveur ou de client eXtreme Scale. Cette rubrique explique comment configurer le fichier XML Blueprint XML pour créer des beans de plug-in et les exposer comme services OSGi pour que eXtreme Scale les utilise.

Pourquoi et quand exécuter cette tâche

Les beans et services sont définis dans un fichier XML Blueprint et le conteneur Blueprint découvre, crée et interconnecte les beans et les expose comme services. Le processus rend les beans accessibles aux autres ensembles OSGi, y compris les ensembles de serveur et de client eXtreme Scale.

Lors de la création de services de plug-in personnalisés pour les utiliser avec eXtreme Scale, l'ensemble qui doit héberger les plug-ins doit être configuré pour utiliser Blueprint. En outre, un fichier XML Blueprint doit être créé et stocké dans l'ensemble. Lisez la rubrique relative à la création d'applications OSGi avec la spécification Blueprint Container qui décrit de manière générale la spécification.

Procédure

1. Créez un fichier XML Blueprint. Attribuez-lui un nom de votre choix. Toutefois, vous devez inclure l'espace de nom Blueprint :

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
   ...
   </blueprint>
   ```

2. Créez des définitions de bean dans le fichier XML Blueprint pour chaque plug-in eXtreme Scale.

   Les beans sont définis en utilisant l'élément `<bean>`, ils peuvent être connectés à d'autres références de bean et ils peuvent inclure des paramètres d'initialisation.

Important : Lors de la définition d'un bean, vous devez utiliser la portée correcte. Blueprint prend en charge les portées singleton et prototype. eXtreme Scale prend également en charge une portée de fragment personnalisée.

Définissez la plupart des plug-ins eXtreme Scale comme prototype ou beans à portée de fragment, car tous les beans doivent être uniques pour chaque fragment ObjectGrid ou instance BackingMap auquel ou à laquelle ils sont associés. Les beans à portée de fragment peuvent être utiles lorsque vous utilisez les beans dans d'autres contextes pour pouvoir extraire l'instance correcte.

Pour définir un bean à portée prototype, utilisez l'attribut `scope="prototype"` sur le bean :

```
<bean id="myPluginBean" class="com.mycompany.MyBean" scope="prototype">
...
</bean>
```

Pour définir un beans à portée de fragment, vous devez ajouter l'espace de nom `objectgrid` au schéma XML et utiliser l'attribut `scope="objectgrid:shard"` sur le bean :

```
<?xml version="1.0" encoding="UTF-8"?>

<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:objectgrid="http://www.ibm.com/schema/objectgrid"

    xsi:schemaLocation="http://www.ibm.com/schema/objectgrid
                http://www.ibm.com/schema/objectgrid/objectgrid.xsd">

    <bean id="myPluginBean" class="com.mycompany.MyBean"
   scope="objectgrid:shard">
    ...
    </bean>

...
```

3. Créez des définitions de bean PluginServiceFactory pour chaque bean de plug-in. Tous les beans eXtreme Scale doivent avoir un bean PluginServiceFactory défini pour que la portée de bean correcte puisse être appliquée. eXtreme Scale inclut une fabrique BlueprintServiceFactory que vous pouvez utiliser. Elle contient deux propriétés que vous devez définir. Vous devez affecter à la propriété blueprintContainer la référence `blueprintContainer` et attribuer à la propriété beanId le nom de l'identificateur du bean. Lorsque eXtreme Scale recherche le service pour instancier les beans appropriés, le serveur recherche l'instance du composant bean en utilisant le conteneur Blueprint.

```
bean id="myPluginBeanFactory"
    class="com.ibm.websphere.objectgrid.plugins.osgi.BluePrintServiceFactory">
    <property name="blueprintContainer" ref="blueprintContainer"/>
    <property name="beanId" value="myPluginBean" />
</bean>
```

4. Créez un gestionnaire de service pour chaque bean PluginServiceFactory. Chaque gestionnaire de service expose le bean PluginServiceFactory en utilisant l'élément `<service>`. L'élément de service identifie le nom à exposer à OSGi, la référence au bean PluginServiceFactory et l'interface à exposer, ainsi que le classement du service. eXtreme Scale utilise le classement du gestionnaire de service pour effectuer des mises à niveau de service lorsque la grille eXtreme Scale est active. Si le classement n'est pas défini, l'infrastructure OSGi utilise le classement 0 par défaut. Consultez la rubrique relative à la mise à jour des classements de service pour plus d'informations.

Blueprint contient diverses options de configuration des gestionnaires de service. Pour définir un gestionnaire de service simple pour un bean PluginServiceFactory, créez un élément `<service>` pour chaque bean PluginServiceFactory :

```
<service ref="myPluginBeanFactory"
    interface="com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactory"
    ranking="1">
</service>
```

5. Stockez le fichier XML Blueprint dans l'ensemble de plug-ins. Le fichier XML Blueprint doit être stocké dans le répertoire `OSGI-INF/blueprint` du conteneur Blueprint pour être découvert.

   Pour stocker le fichier XML Blueprint dans un répertoire différent, vous devez définir l'en-tête de manifeste Bundle-Blueprint suivant :

   ```
   Bundle-Blueprint: OSGI-INF/blueprint.xml
   ```

Résultats

Les plug-ins eXtreme Scale sont maintenant configurés pour être exposés dans un conteneur OSGi Blueprint. En outre, le fichier XML descripteur ObjectGrid est configuré pour référencer les plug-ins en utilisant le service OSGi Blueprint.

Installation et démarrage des plug-ins OSGi

Dans cette tâches, vous installez l'ensemble de plug-in dynamique dans l'infrastructure OSGi, puis vous démarrez le plug-in.

Avant de commencer

Cette rubrique suppose que vous avez exécuté les tâches suivantes :

- Vous avez installé l'ensemble serveur ou client eXtreme Scale dans l'infrastructure OSGi Eclipse Equinox. Voir «Installation des ensembles eXtreme Scale», à la page 41.
- Vous avez implémenté un ou plusieurs plug-ins dynamiques BackingMap ou ObjectGrid. Voir «Génération de plug-ins dynamiques eXtreme Scale», à la page 43.
- Vous avez regroupé les plug-ins dynamiques comme services OSGi dans des ensembles OSGi.

Pourquoi et quand exécuter cette tâche

Cette tâche explique comment installer l'ensemble en utilisant la console Eclipse Equinox. L'ensemble peut être installé en utilisant plusieurs méthodes différentes, y compris en modifiant le fichier de configuration `config.ini`. Les produits qui intègrent Eclipse Equinox incluent des méthodes alternatives de gestion des ensembles. Pour plus d'informations sur l'ajout d'ensembles dans le fichier `config.ini` dans Eclipse Equinox, voir les options d'exécution Eclipse.

OSGi permet de démarrer les ensembles ayant des services dupliqués. WebSphere eXtreme Scale utilise le dernier classement de service. Lors du démarrage de plusieurs infrastructures OSGi dans une grille de données eXtreme Scale, vous devez veiller à démarrer les classements de service corrects sur chaque serveur afin que la grille ne soit pas démarrée en utilisant une combinaison de versions différentes.

Pour identifier les versions utilisées par la grille de données, utilisez l'utilitaire xscmd pour vérifier les classements en cours et disponibles. Pour plus

d'informations sur les classements de service disponibles, voir Mise à jour des services OSGi pour les plug-ins eXtreme Scale avec **xscmd**.

Procédure

Installez l'ensemble de plug-in dans l'infrastructure OSGi Eclipse Equinox en utilisant la console OSGi.

1. Démarrez l'infrastructure Eclipse Equinox avec la console activée, par exemple :
   ```
   <java_home>/bin/java -jar <equinox_root>/plugins/org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
   ```
2. Installez l'ensemble de plug-in dans la console Equinox.
   ```
   osgi> install file:///<path to bundle>
   ```
   Equinox affiche l'ID du nouvel ensemble installé :
   ```
   Bundle id is 17
   ```
3. Entrez la ligne suivante pour démarrer l'ensemble dans la console Equinox, où <id> est l'ID d'ensemble affecté lors de l'installation de l'ensemble :
   ```
   osgi>  install <id>
   ```
4. Extrayez l'état du service dans la console Equinox pour vérifier que l'ensemble a démarré :
   ```
   osgi> ss
   ```
   Lorsque l'ensemble a démarré correctement, il affiche l'état ACTIVE, par exemple :
   ```
   17      ACTIVE      com.mycompany.plugin.bundle_VRM
   ```

Installez l'ensemble de plug-in dans l'infrastructure OSGi Eclipse Equonix en utilisant le fichier config.ini file.

5. Copiez l'ensemble de plug-in dans le répertoire Eclipse Equinox plug-ins, par exemple :
   ```
   <equinox_root>/plugins
   ```
6. Modifiez le fichier de configuration Eclipse Equinox `config.ini` et ajoutez l'ensemble à la propriété osgi.bundles, par exemple :
   ```
   osgi.bundles=\
   org.eclipse.osgi.services_3.2.100.v20100503.jar@1:start, \
   org.eclipse.osgi.util_3.2.100.v20100503.jar@1:start, \
   org.eclipse.equinox.cm_1.0.200.v20100520.jar@1:start, \
   com.mycompany.plugin.bundle_VRM.jar@1:start
   ```
   Important : Vérifiez qu'il existe une ligne blanche après le dernier nom d'ensemble. Chaque ensemble est séparé par une virgule.
7. Démarrez l'infrastructure Eclipse Equinox avec la console activée, par exemple :
   ```
   <java_home>/bin/java -jar <equinox_root>/plugins/org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
   ```
8. Extrayez l'état de service dans la console Equinox pour vérifier que l'ensemble est démarré. Par exemple :
   ```
   osgi> ss
   ```
   Une fois l'ensemble démarré, il affiche l'état ACTIVE. Par exemple :
   ```
   17      ACTIVE      com.mycompany.plugin.bundle_VRM
   ```

Résultats

L'ensemble de plug-in est maintenant installé et démarré. Le conteneur ou le client peut être maintenant démarré eXtreme Scale. Pour plus d'informations sur le

développement des plug-ins eXtreme Scale, voir la rubrique API système et plug-ins.

Exécution de conteneurs eXtreme Scale avec des plug-ins dynamiques dans un environnement OSGi

Si l'application est hébergée dans l'infrastructure OSGi Eclipse Equinox avec Eclipse Gemini ou Apache Aries, vous pouvez utiliser cette tâche pour installer et configurer l'application WebSphere eXtreme Scale dans OSGi.

Avant de commencer

Avant de démarrer cette tâche, procédez comme suit :

- Installez l'infrastructure OSGi Eclipse Equinox avec Eclipse Gemini
- Créez et exécutez les plug-ins dynamiques eXtreme Scale pour les utiliser dans un environnement OSGi

Pourquoi et quand exécuter cette tâche

Avec les plug-ins dynamiques, vous pouvez mettre à niveau dynamiquement les plug-ins lorsque la grille est active. Cela vous permet de faire évoluer une application sans redémarrer les processus de conteneur de la grille. Pour plus d'informations sur le développement de plug-ins eXtreme Scale, voir API système et plug-ins.

Procédure

1. Configurez les plug-ins OSGi en utilisant le fichier XML descripteur ObjectGrid.
2. Démarrez les serveurs de conteneur eXtreme Scale en utilisant l'infrastructure OSGi Eclipse Equinox.
3. Administrez les services OSGi pour les plug-ins eXtreme Scale avec l'utilitaire xscmd.
4. Configurez les serveurs avec OSGi Blueprint.

Configuration des plug-ins OSGi en utilisant le fichier descripteur XML ObjectGrid

Dans cette tâche, vous ajoutez des services OSGi existants à un fichier XML descripteur pour que les conteneurs WebSphere eXtreme Scale puissent reconnaître et charger correctement les plug-ins OSGi.

Avant de commencer

Pour configurer vos plug-ins, veillez à :

- Créer le module et activer les plug-ins dynamiques du déploiement OSGi.
- Disposer des noms des services OSGi qui représentent les plug-ins.

Pourquoi et quand exécuter cette tâche

Vous avez créé un service OSGi pour encapsuler le plug-in. Maintenant, ces services doivent être définis dans le fichier `objectgrid.xml` pour que les conteneurs eXtreme Scale puissent charger et configurer le ou les plug-ins

Procédure

1. Les plug-ins de grille, tels que TransactionCallback, doivent être définis sous l'élément objectGrid. Voir l'exemple suivant du fichier `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="MyGrid" txTimeout="60">
            <bean id="myTranCallback" osgiService="myTranCallbackFactory"/>
            ...
        </objectGrid>
        ...
    </objectGrids>
    ...
/objectGridConfig>
```

Important : La valeur d'attribut `osgiService` doit correspondre à la valeur d'attribut `ref` définie dans le fichier XML `blueprint`, où le service a été défini pour myTranCallback PluginServiceFactory.

2. Les plug-ins de mappe, tels que les chargeurs ou les sérialiseurs, doivent être définis dans l'élément backingMapPluginCollections et référencés depuis l'élément backingMap. Voir l'exemple suivant du fichier `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="MyGrid" txTimeout="60">
            <backingMap name="MyMap1" lockStrategy="PESSIMISTIC"
                copyMode="COPY_TO_BYTES" nullValuesSupported="false"
                pluginCollectionRef="myPluginCollectionRef1"/>
            <backingMap name="MyMap2" lockStrategy="PESSIMISTIC"
                copyMode="COPY_TO_BYTES" nullValuesSupported="false"
                pluginCollectionRef="myPluginCollectionRef2"/>
            ...
        </objectGrid>
        ...
    </objectGrids>
    ...
    <backingMapPluginCollections>
        <backingMapPluginCollection id="myPluginCollectionRef1">
            <bean id="MapSerializerPlugin" osgiService="mySerializerFactory"/>
        </backingMapPluginCollection>
        <backingMapPluginCollection id="myPluginCollectionRef2">
            <bean id="MapSerializerPlugin" osgiService="myOtherSerializerFactory"/>
            <bean id="Loader" osgiService="myLoader"/>
        </backingMapPluginCollection>
        ...
    </backingMapPluginCollections>
    ...
</objectGridConfig>
```

Résultats

Le fichier `objectgrid.xml` dans cet exemple demande à eXtreme Scale de créer la grille `MyGrid` avec les deux mappes `MyMap1` et `MyMap2`. La mappe `MyMap1` utilise le sérialiseur encapsulé par le service OSGi, mySerializerFactory. La mappe `MyMap2` utilise un sérialiseur depuis le service OSGi, myOtherSerializerFactory, et un chargeur depuis le service OSGi, myLoader.

Démarrage des serveurs eXtreme Scale en utilisant l'infrastructure OSGi Eclipse Equinox

Les serveurs de conteneur WebSphere eXtreme Scale peuvent être démarrés dans une infrastructure OSGi Eclipse Equinox en utilisant plusieurs méthodes.

Avant de commencer

Pour pouvoir démarrer un conteneur eXtreme Scale, vous devez exécuter les tâches suivantes :

1. L'ensemble de serveur WebSphere eXtreme Scale doit être installé dans Eclipse Equinox.
2. L'application doit être placée dans un ensemble OSGi.
3. Les plug-ins WebSphere eXtreme Scale (s'il en existe) doivent être placés dans un ensemble OSGi. Ils peuvent se trouver dans le même ensemble que l'application ou dans des ensembles séparés.

Pourquoi et quand exécuter cette tâche

Cette tâche explique comment démarrer un serveur de conteneur eXtreme Scale dans une infrastructure OSGi Eclipse Equinox. Vous pouvez utiliser n'importe laquelle des méthodes suivantes pour démarrer les serveurs de conteneur en utilisant l'implémentation Eclipse Equinox :

- Service OSGi Blueprint

  Vous pouvez inclure toute la configuration et toutes les métadonnées dans un ensemble OSGi. Voir l'illustration suivante pour comprendre le processus Eclipse Equinox de cette méthode :

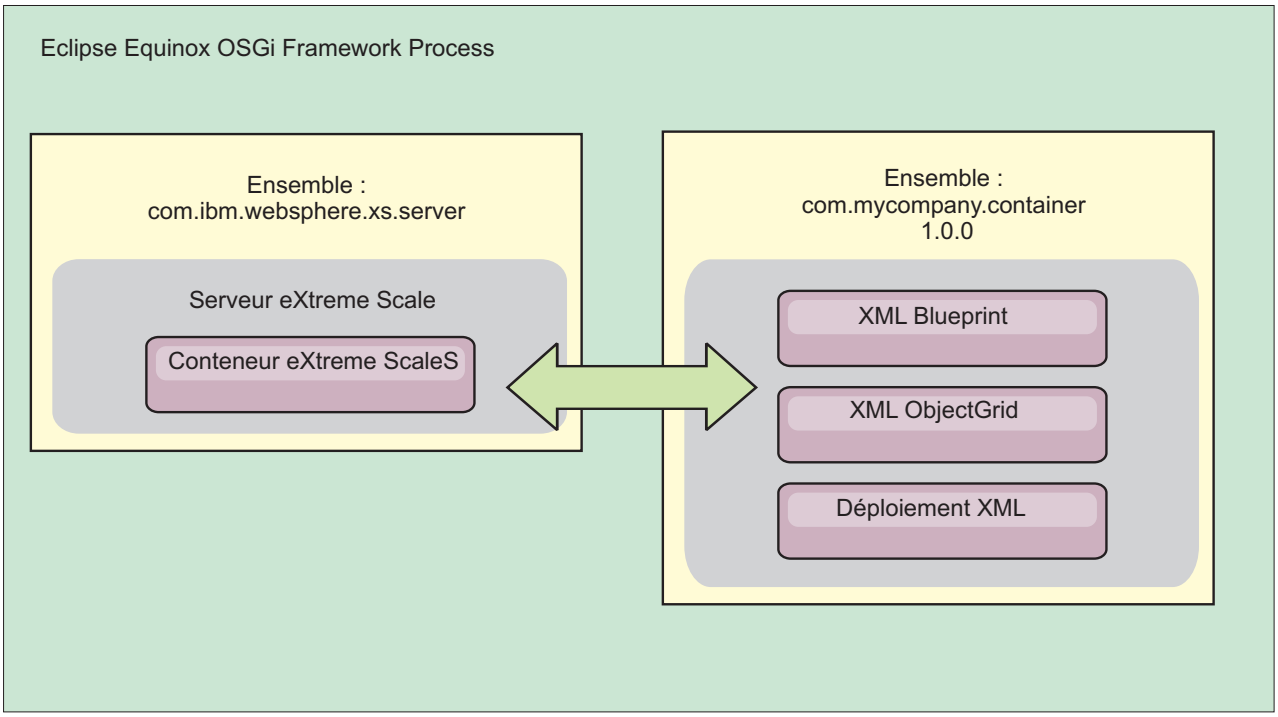


Figure 2. Processus Eclipse Equinox pour inclure toute la configuration et toutes les métadonnées dans un ensemble OSGi

- Service Admin de configuration OSGi

  Vous pouvez définir la configuration et les métadonnées en dehors d'un ensemble OSGi. Voir l'image suivante pour comprendre le processus Eclipse Equinox pour cette méthode :

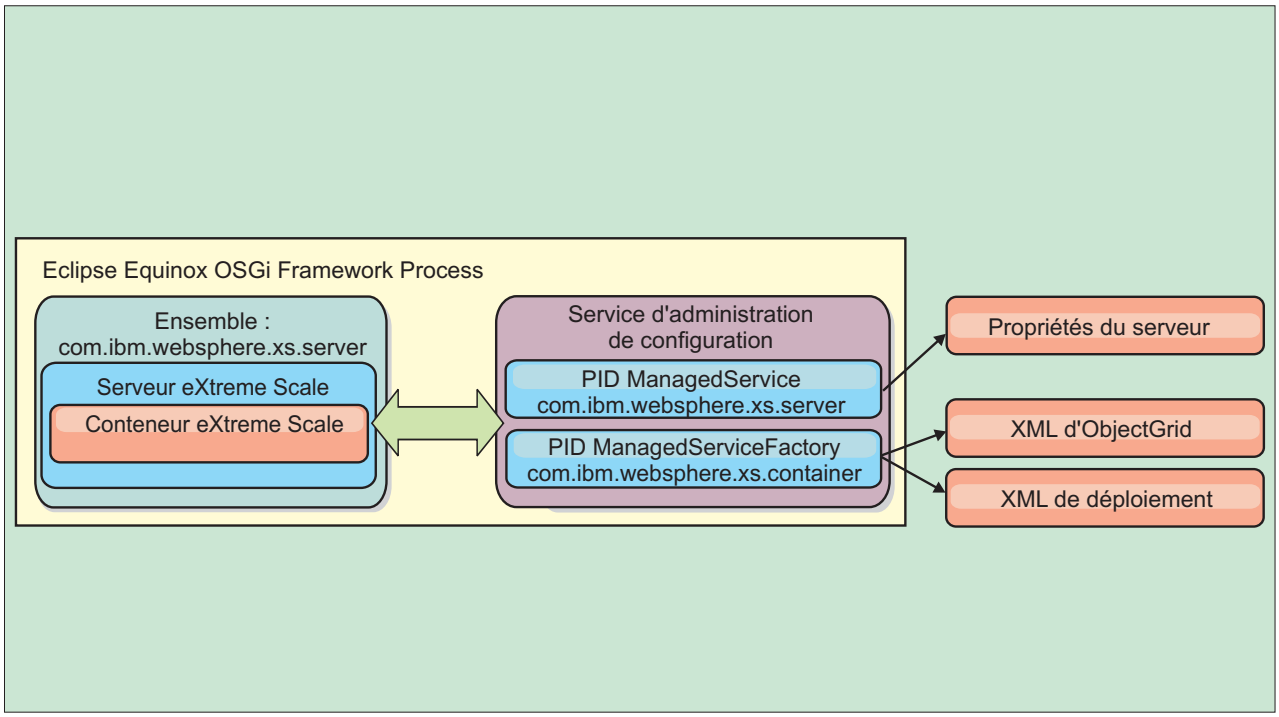


Figure 3. Processus Eclipse Equinox pour définir la configuration et les métadonnées en dehors d'un ensemble OSGi

- A l'aide d'un programme

  Prend en charge les solutions de configuration personnalisées.

Dans chaque cas, un singleton de serveur eXtreme Scale est configuré et un ou plusieurs conteneurs sont configurés.

L'ensemble de serveur eXtreme Scale, `objectgrid.jar`, contient toutes les bibliothèques nécessaires pour démarrer et exécuter un conteneur de grille eXtreme Scale dans une infrastructure OSGi. L'environnement d'exécution du serveur communique avec les plug-ins fournis par l'utilisateur et les objets de données en utilisant le gestionnaire de service OSGi.

Important : Après que l'ensemble de serveur eXtreme Scale a été démarré et le serveur eXtreme Scale initialisé, il ne peut pas être redémarré. Le processus Eclipse Equinox doit être redémarré pour redémarrer le serveur eXtreme Scale.

Vous pouvez utiliser le support eXtreme Scale pour l'espace de nom Spring pour configurer les serveurs de conteneur eXtreme Scale dans un fichier XML Blueprint. Lorsque les éléments XML de serveur et de conteneur sont ajoutés au fichier XML Blueprint, le gestionnaire d'espace de nom eXtreme Scale démarre automatiquement un serveur de conteneur en utilisant les paramètres définis dans le fichier XML Blueprint lors du démarrage de l'ensemble. Le gestionnaire arrête le conteneur lorsque l'ensemble s'arrête.

Pour configurer les serveurs de conteneur eXtreme Scale avec XML Blueprint, procédez comme suit :

Procédure

- Démarrez un serveur de conteneur eXtreme Scale en utilisant OSGi Blueprint.
  1. Créez un ensemble de conteneur.

2. Installez l'ensemble de conteneur dans l'infrastructure OSGi Eclipse Equinox. Voir «Installation et démarrage des plug-ins OSGi», à la page 49.
3. Démarrez l'ensemble de conteneur.

- Démarrez un serveur de conteneur eXtreme Scale en utilisant l'administrateur de configuration OSGi.
  1. Configurez le serveur et le conteneur en utilisant l'administrateur de configuration.
  2. Lorsque l'ensemble de serveur eXtreme Scale est démarré ou que les PID (persistent identifier) sont créé avec l'administrateur de configuration, le serveur et le conteneur démarrent automatiquement.
- Démarrez un serveur de conteneur eXtreme Scale en utilisant l'API ServerFactory. Voir la documentation d'API de serveur.
  1. Créez une classe d'activateur d'ensemble OSGi et utilisez l'API eXtreme Scale ServerFactory pour démarrer un serveur.

Administration des services OSGi en utilisant l'utilitaire `xscmd`

Vous pouvez utiliser l'utilitaire **xscmd** pour exécuter des tâches d'administration, telles qu'afficher les serveurs et leurs classements utilisés par chaque conteneur, et mettre à niveau l'environnement d'exécution pour utiliser les nouvelles versions des ensembles.

Pourquoi et quand exécuter cette tâche

Avec l'infrastructure Eclipse Equinox OSGi, vous pouvez installer plusieurs versions d'un même ensemble et vous pouvez mettre à jour ces ensemble lors de l'exécution. WebSphere eXtreme Scale est un environnement distribué qui exécute les serveurs de conteneur dans une multitude d'instances de l'infrastructure OSGi.

Les administrateurs doivent copier, installer et démarrer manuellement les ensembles dans l'infrastructure OSGi. eXtreme Scale contient un personnaliseur ServiceTrackerCustomizer OSGi pour suivre les services identifiés comme plug-ins eXtreme Scale dans le fichier XML descripteur. Utilisez l'utilitaire **xscmd** pour valider la version utilisée du plug-in, les versions pouvant être utilisées et exécuter des mises à niveau d'ensemble.

eXtreme Scale utilise le numéro de classement de service pour identifier la version de chaque service. Lorsque au moins deux services sont chargés avec la même référence, eXtreme Scale utilise automatiquement le service ayant le classement le plus élevé.

Procédure

- Exécutez la commande **osgiCurrent** et vérifiez que chaque serveur eXtreme Scale utilise le classement de service de plug-in correct.

  Comme eXtreme Scale choisit automatiquement la référence de service ayant le classement le plus élevé, il se peut que la grille de données démarre avec plusieurs classements d'un service de plug-in.

  Si la commande détecte une discordance de classements ou qu'elle ne trouve pas un service, un niveau d'erreur différent de zéro est défini. Si la commande aboutit, le niveau d'erreur 0 est défini.

  L'exemple suivant montre la sortie de la commande **osgiCurrent** lorsque deux plus-ins sont installés dans une grille sur quatre serveurs. Le plug-in loaderPlugin utilise le classement 1 et le plug-in txCallbackPlugin, le classement 2.

```
OSGi Service Name Current Ranking ObjectGrid Name MapSet Name Server Name
----------------- --------------- --------------- ----------- -----------
loaderPlugin      1               MyGrid          MapSetA     server1
loaderPlugin      1               MyGrid          MapSetA     server2
loaderPlugin      1               MyGrid          MapSetA     server3
loaderPlugin      1               MyGrid          MapSetA     server4
txCallbackPlugin  2               MyGrid          MapSetA     server1
txCallbackPlugin  2               MyGrid          MapSetA     server2
txCallbackPlugin  2               MyGrid          MapSetA     server3
txCallbackPlugin  2               MyGrid          MapSetA     server4
```

L'exemple suivant montre la sortie de la commande **osgiCurrent** lorsque le serveur 2 a été démarré avec un nouveau classement du plug-in loaderPlugin :

```
OSGi Service Name Current Ranking ObjectGrid Name MapSet Name Server Name
----------------- --------------- --------------- ----------- -----------
loaderPlugin      1               MyGrid          MapSetA     server1
loaderPlugin      2               MyGrid          MapSetA     server2
loaderPlugin      1               MyGrid          MapSetA     server3
loaderPlugin      1               MyGrid          MapSetA     server4
txCallbackPlugin  2               MyGrid          MapSetA     server1
txCallbackPlugin  2               MyGrid          MapSetA     server2
txCallbackPlugin  2               MyGrid          MapSetA     server3
txCallbackPlugin  2               MyGrid          MapSetA     server4
```

- Exécutez la commande **osgiAll** pour vérifier que les services de plug-in ont été correctement démarrés sur chaque serveur de conteneur eXtreme Scale.

  Lorsque des ensembles contenant des services référencés par une configuration ObjectGrid démarrent, l'environnement d'exécution eXtreme Scale suit le plug-in, mais il ne l'utilise pas immédiatement. La commande **osgiAll** montre les plug-ins disponibles pour chaque serveur.

  Lorsqu'elle est exécutée sans paramètres, tous les services de toutes les grilles et de tous les serveurs sont indiqués. Des filtres supplémentaires, notamment le filtre **-serviceName** <service_name>, peuvent être définis pour limiter la sortie à un seul service ou sous-ensemble de la grille de données.

  L'exemple suivant montre la sortie de la commande **osgiAll** lorsque deux plug-ins sont démarrés sur deux serveurs. Les classements 1 et 2 du plug-in loaderPlugin sont démarrés et le classement 1 du plug-in txCallbackPlugin est démarré. Le résumé à la fin de la sortie indique que les deux serveurs voient les mêmes classements de service :

```
Server: server1
   OSGi Service Name   Available Rankings
   -----------------   ------------------
   loaderPlugin        1, 2
   txCallbackPlugin    1

Server: server2
   OSGi Service Name   Available Rankings
   -----------------   ------------------
   loaderPlugin        1, 2
   txCallbackPlugin    1

Summary - All servers have the same service rankings.
```

  L'exemple suivant montre la sortie de la commande **osgiAll** lorsque l'ensemble qui contient le plug-in loaderPlugin avec le classement 1 est arrêté sur le serveur 1. Le résumé à la fin de la sortie indique que le serveur n'a pas le plug-in loaderPlugin avec le classement 1 :

```
Server: server1
   OSGi Service Name   Available Rankings
   -----------------   ------------------
   loaderPlugin        2
   txCallbackPlugin    1
```

```
Server: server2
   OSGi Service Name   Available Rankings
   -----------------   ------------------
   loaderPlugin        1, 2
   txCallbackPlugin    1

Summary - The following servers are missing service rankings:
   Server  OSGi Service Name Missing Rankings
   ------  ---------------- ----------------
   server1 loaderPlugin      1
```

L'exemple suivant montre la sortie si le nom de service est défini avec l'argument **-sn** et que le service n'existe pas.

```
Server: server2
   OSGi Service Name Available Rankings
   ----------------- ------------------
   invalidPlugin     No service found

Server: server1
   OSGi Service Name Available Rankings
   ----------------- ------------------
   invalidPlugin     No service found

Summary - All servers have the same service rankings.
```

- Exécutez la commande **osgiCheck** pour vérifier les groupes de services de plug-in et de classements s'ils sont disponibles.

  La commande **osgiCheck** accepte un ou plusieurs groupes de classements de service de la manière suivante `-serviceRankings <service name>;<ranking>[,<serviceName>;<ranking>]`

  Lorsque les classements sont tous disponibles, la méthode retourne un niveau d'erreur 0. Si un ou plusieurs classements sont indisponibles, un niveau d'erreur différent de zéro et la table de tous les serveurs qui ne contiennent pas les classements de service définis sont spécifiés. Des filtres supplémentaires peuvent être utilisés pour limiter la vérification des services à un sous-ensemble des serveurs disponibles dans le domaine eXtreme Scale.

  Par exemple, si le classement ou le service est absent, le message suivant s'affiche :

```
Server  OSGi Service Unavailable Rankings
------  ------------ --------------------
server1 loaderPlugin 3
server2 loaderPlugin 3
```

- Exécutez la commande **osgiUpdate** pour mettre à jour le classement d'un ou de plusieurs plug-ins pour tous les serveurs dans un seul ObjectGrid et MapSet dans une seule opération.

  La commande accepte un ou plusieurs groupes de classements de service de la manière suivante : `-serviceRankings <service name>;<ranking>[,<serviceName>;<ranking>] -g <grid name> -ms <mapset name>`

  Avec cette commande, vous pouvez exécuter les opérations suivantes :

  – Vérifier que les services spécifiés sont disponibles pour la mise à niveau sur chacun des serveurs.

  – Mettre la grille hors ligne en utilisant l'interface StateManager. Pour plus d'informations, voir Gestion de la disponibilité ObjectGrid. Ce processus met au repos la grille et attend la fin des transactions en cours en interdisant le démarrage de nouvelles transactions. Ce processus indique également aux programmes d'écoute ObjectGridLifecycleListener et BackingMapLifecycleListener d'arrêter toute activité transactionnelle. Voir

«Plug-in de programme d'écoute d'événement», à la page 317 pour plus d'informations sur les plug-ins de programme d'écoute.

– Mettre à jour chaque conteneur eXtreme Scale exécuté dans une infrastructure OSGi pour utiliser les nouvelles versions de service.
– Mettre la grille en ligne pour reprendre l'exécution des transactions.

Le processus de mise à jour est idempotent de sortie que si un client n'exécute pas une tâche, l'opération est annulée. Si un client ne peut pas exécuter l'annulation ou qu'il est interrompu pendant la mise à jour, la même commande peut être réexécutée et elle reprend à l'étape appropriée.

Si le client ne peut pas continuer et que le processus est redémarré depuis un autre client, utilisez l'option `-force` pour permettre au client d'exécuter la mise à jour. La commande **`osgiUpdate`** empêche plusieurs clients de mettre à jour simultanément un même groupe de mappes. Pour plus d'informations sur la commande **`osgiUpdate`**, voir Mise à jour des services OSGi pour les plug-ins eXtreme Scale avec **`xscmd`**.

Configuration des serveurs avec OSGi Blueprint

Vous pouvez configurer les serveurs de conteneur WebSphere eXtreme Scale en utilisant un fichier XML OSGi Blueprint qui permet de simplifier le regroupement et le développement d'ensembles de serveur autonomes.

Avant de commencer

Cette rubrique suppose que vous avez exécuté les tâches suivantes :

- L'infrastructure OSGi Eclipse Equinox a été installée et démarrée avec le conteneur Eclipse Gemini ou Apache Aries Blueprint.
- L'ensemble de serveur eXtreme Scale a été installé et démarré.
- L'ensemble de plug-ins dynamiques eXtreme Scale a été créé.
- Le fichier XML descripteur eXtreme Scale ObjectGrid et le fichier XML de stratégie de déploiement ont été créés.

Pourquoi et quand exécuter cette tâche

Cette tâche explique comment configurer un serveur eXtreme Scale avec un conteneur en utilisant un fichier XML Blueprint. Le résultat de la procédure est un ensemble de conteneur. Lorsque l'ensemble de conteneur est démarré, l'ensemble de serveur eXtreme Scale suit l'ensemble, analyse le fichier XML de serveur et démarre un serveur et un conteneur.

Un ensemble de conteneur peut être éventuellement combiné à l'application et aux plug-ins eXtreme Scale lorsque des mises à jour de plug-ins dynamiques ne sont pas nécessaires ou que les plug-ins ne prennent pas en charge la mise à jour dynamique.

Procédure

1. Créez un fichier XML Blueprint avec l'espace de nom `objectgrid` inclut. Vous pouvez affecter le nom de votre choix au fichier. Toutefois, il doit inclure l'espace de nom Blueprint :

```
<?xml version="1.0" encoding="UTF-8"?>

<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:objectgrid="http://www.ibm.com/schema/objectgrid"
```

```
          xsi:schemaLocation="http://www.ibm.com/schema/objectgrid
               http://www.ibm.com/schema/objectgrid/objectgrid.xsd">
...
</blueprint>
```

2. Ajoutez la définition XML du serveur eXtreme Scale avec les propriétés de serveur appropriées. Voir le fichier XML descripteur Spring pour plus d'informations sur toutes les propriétés de configuration disponibles. Voir l'exemple suivant de définition de fichier XML :

```
objectgrid:server
    id="xsServer"
   tracespec="ObjectGridOSGi=all=enabled"
   tracefile="logs/osgi/wxsserver/trace.log"
    jmxport="1199"
   listenerPort="2909">
    <objectgrid:catalog host="catserver1.mycompany.com" port="2809" />
    <objectgrid:catalog host="catserver2.mycompany.com" port="2809" />
</objectgrid:server>
```

3. Ajoutez la définition XML du conteneur eXtreme Scale avec la référence à la définition de serveur et les fichiers XML descripteur d'ObjectGrid et de déploiement d'ObjectGrid regroupés dans l'ensemble. Par exemple :

```
<objectgrid:container id="container"
    objectgridxml="/META-INF/objectGrid.xml"
    deploymentxml="/META-INF/objectGridDeployment.xml"
    server="xsServer" />
```

4. Stockez le fichier XML Blueprint dans l'ensemble de conteneur. Le fichier XML Blueprint doit être stocké dans le répertoire `OSGI-INF/blueprint` du conteneur Blueprint pour être trouvé.

   Pour stocker le fichier XML Blueprint dans un répertoire différent, vous devez définir l'en-tête du manifeste Bundle-Blueprint. Par exemple :

```
Bundle-Blueprint: OSGI-INF/blueprint.xml
```

5. Regroupez les fichiers dans un fichier JAR d'ensemble unique. Voir l'exemple suivant de hiérarchie de répertoires d'ensemble :

```
MyBundle.jar
    /META-INF/manifest.mf
    /META-INF/objectGrid.xml
    /META-INF/objectGridDeployment.xml
    /OSGI-INF/blueprint/blueprint.xml
```

Résultats

Un ensemble de conteneur eXtreme Scale est maintenant créé et peut être installé dans Eclipse Equinox. Lorsque l'ensemble de conteneur est démarré, l'environnement d'exécution du serveur eXtreme Scale dans l'ensemble de serveur eXtreme Scale démarre automatiquement le serveur eXtreme Scale de singleton en utilisant les paramètres définis dans l'ensemble et démarre un serveur de conteneur. L'ensemble peut être arrêté et démarré, ce qui arrête et redémarre le conteneur. Le serveur est un singleton et ne s'arrête pas lorsque l'ensemble est démarré pour la première fois.

Chapitre 3. Mise en route

Après avoir installé le produit, vous pouvez utiliser l'exemple de mise en route pour tester l'installation et utiliser le produit pour la première fois.

Tutoriel : Démarrer avec WebSphere eXtreme Scale

Après avoir installé WebSphere eXtreme Scale dans un environnement autonome, vous pouvez utiliser l'exemple d'application de démarrage qui présente clairement ses fonctions comme grille de données en mémoire.

Objectifs d'apprentissage

- Description des fichiers descripteur XML ObjectGrid de stratégie de déploiement et des fichiers descripteurs XML utilisés pour configurer l'environnement
- Démarrage des serveurs de catalogue et de conteneur à l'aide des fichiers de configuration
- En savoir plus sur le développement d'une application client
- Exécution de l'application client pour insérer des données dans la grille de données
- Surveillance des grilles de données à l'aide de la console Web

Durée

60 minutes

Leçon 1 du tutoriel d'initialisation : Définition de grilles de données avec des fichiers de configuration

Pour configurer des grilles de données simples, vous pouvez utiliser les fichiers `objectgrid.xml` et `deployment.xml` fournis dans l'exemple d'initialisation.

L'exemple utilise les fichiers `objectgrid.xml` et `deployment.xml` qui se trouvent dans le répertoire *racine_install_wxs*`/ObjectGrid/gettingstarted/xml`. Ces fichiers sont envoyés aux commandes de démarrage pour démarrer les serveurs de conteneur et un serveur de catalogue. Le fichier `objectgrid.xml` est le fichier XML descripteur d'ObjectGrid. Le fichier `deployment.xml` est le fichier XML descripteur de la stratégie de déploiement ObjectGrid. Ensemble, ces fichiers définissent une topologie répartie.

Référence associée:

Fichier XML du descripteur d'ObjectGrid
Pour configurer WebSphere eXtreme Scale, utilisez un fichier XML de descripteur d'ObjectGrid et l'API ObjectGrid.

Fichier XML du descripteur de la règle de déploiement
Pour configurer une règle de déploiement, utilisez un fichier XML de descripteur de règle de déploiement.

Fichier XML 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 la liste des configurations de

mappes de sauvegarde. Ces mappes de sauvegarde stockent les 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. Cet exemple de fichier définit l'ObjectGrid `Grid` 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 stratégie de déploiement est transmis au serveur de conteneur 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. Pour chaque élément objectgridDeployment dans la stratégie de déploiement, vous devez disposer d'un élément ObjectGrid correspondant dans le 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 1. La valeur doit être adaptée à la capacité anticipée de la grille de données.

L'attribut minSyncReplicas de l'élément mapSet vise à indiquer le nombre minimal de fragments réplique 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 réplique 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 réplique 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 réplique 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 réplique synchrones pour cette partition spécifique. L'ajout de conteneurs prenant en charge cette grille d'objets peut entraîner un nombre croissant de fragments réplique 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.

Point de contrôle de la leçon

Dans cette leçon, vous avez appris à :

- Définir des mappes qui stockent les données dans le fichier XML de descripteur d'ObjectGrid.
- Utiliser le fichier XML descripteur de déploiement pour définir le nombre de partitions et de répliques de la grille de données.

Leçon 2 du tutoriel d'initiation : Création d'une application client

Pour pouvoir insérer, supprimer, mettre à jour et extraire des données dans votre grille de données, vous devez écrire une application client. L'exemple d'initiation inclut une application client que vous pouvez utiliser pour en savoir plus sur la création de votre propre application client.

Le fichier `Client.java` dans le répertoire *racine_install_wxs*`/ObjectGrid/gettingstarted/client/src/` est le programme client qui montre comment se connecter à un serveur de catalogue, obtenir l'instance ObjectGrid et utiliser l'API ObjectMap. L'API ObjectMap stocke les données comme paires clé-valeur et elle est idéale pour la mise en cache d'objets qui n'ont aucune relation.

Si devez mettre en cache des objets qui ont des relations, utilisez l'API EntityManager.

1. Connectez-vous au service de catalogue en obtenant une instance ClientClusterContext.

   Pour établir la connexion au serveur de catalogues, utilisez la méthode connect de l'API ObjectGridManager. La méthode connect utilisée requiert seulement un noeud final de serveur de catalogue au format *`nom_hôte:port`*. Vous pouvez indiquer plusieurs noeuds finaux de serveur de catalogue en séparant les valeurs *`hostname:port`* par une virgule. Le fragment de code suivant montre comment se connecter à un serveur de catalogue et obtenir une instance ClientClusterContext :

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

Si les connexions aux serveurs de catalogue aboutissent, la méthode connect retourne une instance ClientClusterContext. L'instance ClientClusterContext est requise pour obtenir l'ObjectGrid à partir de l'API ObjectGridManager.

2. Obtenez 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 de grille de données. L'instance ClientClusterContext est obtenue pendant la connexion au serveur de catalogue. Le nom de l'instance ObjectGrid est `Grid` ; ce nom est spécifié dans le fichier `objectgrid.xml`. Le fragment de code suivant montre comment obtenir la grille de données en appelant la méthode getObjectGrid de l'API ObjectGridManager.

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

3. Obtenez une instance Session.

   Vous pouvez obtenir une session de l'instance ObjectGrid obtenue. Une instance Session est indispensable 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. Obtenez une instance ObjectMap.

   Après avoir obtenu une instance Session, vous pouvez obtenir une instance ObjectMap depuis une instance Session en appelant la méthode getMap de l'API Session. Vous devez transmettre le nom de la mappe comme paramètre à la méthode getMap pour obtenir l'instance ObjectMap. Le fragment de code suivant montre comment obtenir ObjectMap en appelant la méthode getMap de l'API Session.

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

5. Utilisez les méthodes ObjectMap.

   Une fois une instance ObjectMap obtenue, vous pouvez utiliser l'API ObjectMap. N'oubliez pas que l'interface ObjectMap est une mappe transactionnelle et qu'elle requiert une démarcation de transaction à l'aide des méthodes begin et commit de l'API Session. Faute de démarcation de transaction explicite, les opérations ObjectMap s'exécutent avec des transactions de validation automatique.

   Le fragment de code suivant montre comment utiliser l'API ObjectMap avec une transaction 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();
   ```

Concepts associés:

«Mise en cache d'objets sans aucune relation impliquée (API ObjectMap)», à la page 155
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.

Tâches associées:

«Initiation au développement d'applications», à la page 70
Pour commencer à développer des applications WebSphere eXtreme Scale, configurez n environnement de développement dans Eclipse.

«Tutoriel : Stockage des informations de commande dans des entités», à la page 7
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.

Point de contrôle de la leçon

Dans cette leçon, vous avez appris à créer une application client simple pour effectuer des opérations de grille de données.

Leçon 3 du tutoriel d'initiation 3 : Exécution de l'exemple d'application client démarrée

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

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`

Pour exécuter l'application, vous devez d'abord démarrer le processus de service de catalogue. Le service de catalogue est le centre de contrôle de la grille de données. Il conserve la trace des emplacements de serveurs de conteneur et contrôle le placement des données sur les serveurs de conteneur hôtes. Une fois que le service de catalogue démarre, vous pouvez démarrer les serveurs de conteneur qui stockent les données d'application de la grille de données. Pour stocker plusieurs copies des données, vous pouvez démarrer plusieurs serveurs de conteneur. Lorsque tous les serveurs sont démarrés, vous pouvez exécuter l'application client pour insérer, mettre à jour, supprimer et extraire des données de la grille de données.

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

   `cd` *`racine_install_wxs`*`/ObjectGrid/gettingstarted`

   Remplacez *racine_install_wxs* par le chemin d'accès au répertoire racine d'installation d'eXtreme Scale ou le chemin d'accès de l'évaluation eXtreme Scale extraite *racine_install_wxs*.

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.

   Vous pouvez également démarrer le service de catalogue avec la commande **startOgServer**. Exécutez **startOgServer** depuis le répertoire *racine_install_wxs*/ObjectGrid/bin :
   - UNIX Linux `startOgServer.sh cs0 -catalogServiceEndPoints cs0:localhost:6600:6601 -listenerPort 2809`
   - Windows `startOgServer.bat cs0 -catalogServiceEndPoints cs0:localhost:6600:6601 -listenerPort 2809`
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 cette étape avec un nom de serveur différent si vous voulez démarrer plus d'instances de serveurs de conteneur pour prendre en charge la réplication.

   Vous pouvez également démarrer les serveurs de conteneur avec la commande **startOgServer**. Exécutez **startOgServer** depuis le répertoire *racine_install_wxs*/ObjectGrid/bin :
   - UNIX Linux `startOgServer.sh c0 -catalogServiceEndPoints localhost:2809 -objectgridFile gettingstarted\xml\objectgrid.xml -deploymentPolicyFile gettingstarted\xml\deployment.xml`
   - Windows `startOgServer.bat c0 -catalogServiceEndPoints localhost:2809 -objectgridFile gettingstarted\xml\objectgrid.xml -deploymentPolicyFile gettingstarted\xml\deployment.xml`
5. Ouvrez une autre session terminal ou fenêtre de ligne de commande pour exécuter le client.

   Le script `runclient.sh|bat` exécute le client CRUD et démarre l'opération voulue. Le script `runclient.sh|bat` est exécuté avec les paramètres suivants :
   - UNIX Linux `./runclient.sh` *`commande valeur1 valeur2`*
   - Windows `runclient.bat` *`command value1 value2`*

   Pour *commande*, utilisez l'une des options suivantes :
   - Définissez `i` pour insérer *value2* dans la grille de données avec la clé *value1*
   - 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*

   a. Ajoutez des données à la grille de données :
      - UNIX Linux `./runclient.sh i key1 helloWorld`
      - Windows `runclient.bat i key1 helloWorld`

   b. Recherchez et affichez la valeur :

- UNIX Linux `./runclient.sh g key1`
- Windows `runclient.bat g key1`

c. Mettez la valeur à jour :
   - UNIX Linux `./runclient.sh u key1 goodbyeWorld`
   - Windows `runclient.bat u key1 goodbyeWorld`

d. Supprimez la valeur :
   - UNIX Linux `./runclient.sh d key1`
   - Windows `runclient.bat d key1`

Tâches associées:

Démarrage et arrêt des serveurs sécurisés
Vous pouvez démarrer et arrêter les serveurs de catalogue autonome et de conteneur avec les scripts **startOgServer** et **stopOgServer** ou l'API de serveur intégré.

Référence associée:

Script **startOgServer**
Le script **startOgServer** arrête les serveurs de catalogue et de conteneur. Vous pouvez utiliser divers paramètres lorsque vous démarrez vos serveurs pour activer la trace, spécifiez des numéros de port, etc.

Point de contrôle de la leçon

Dans cette leçon, vous avez appris à :

- Démarrer les serveurs de catalogue et les serveurs de conteneur
- Exécuter l'exemple d'application client

Leçon 4 du tutoriel du guide de démarrage : Surveillance de l'environnement

Vous pouvez utiliser l'utilitaire **xscmd** et les outils de la console Web pour surveiller votre environnement de grille de données.

Tâches associées:

Affichage des statistiques avec la console Web
Vous pouvez surveiller les statistiques et d'autres informations de performances avec la console Web.

Surveillance à l'aide de la console Web
Avec la console Web, vous pouvez générer des graphiques des statistiques actuelles et historiques. Cette console fournit un certain nombre de graphiques préconfigurés pour des présentations générales et elle comporte une page de rapports personnalisés que vous pouvez utiliser pour élaborer des graphiques à partir des statistiques disponibles. Les fonctionnalités graphiques de la console de surveillance de WebSphere eXtreme Scale permettent de visualiser les performances globales des grilles des données présentes dans votre environnement.

Démarrage et consignation sur la console Web
Démarrez le serveur de la console en exécutant la commande **startConsoleServer** et en vous connectant au serveur en utilisant l'ID utilisateur et le mot de passe par défaut.

Connexion de la console Web aux serveurs de catalogue
Pour démarrer les statistiques d'affichage dans la console Web, vous devez d'abord vous connecter aux serveurs de catalogue que vous voulez surveiller. Des étapes supplémentaires sont requises si la sécurité est activée sur les serveurs de catalogue.

Surveillance avec l'utilitaire **xscmd**
L'utilitaire **xscmd** remplace l'exemple d'utilitaire **xsadmin** comme outil de surveillance et d'administration complètement pris en charge. Avec l'utilitaire **xscmd**, vous pouvez afficher des informations textuelles relatives à votre topologie WebSphere eXtreme Scale.

Administration avec l'utilitaire **xscmd**
Avec **xscmd**, vous pouvez effectuer des tâches d'administration dans l'environnement, telles qu'établir des liens de réplication multimaître, remplacer un quorum et arrêter des groupes de serveurs avec la commande teardown.

Référence associée:

Statistiques de la console Web
En fonction de la vue que vous utilisez dans la console Web, vous pouvez afficher différentes statistiques relatives à votre configuration. Ces statistiques incluent la mémoire utilisée, les grilles de données les plus utilisées et le nombre d'entrées en mémoire cache.

Script stopOgServer
Le script **stopOgServer** arrête les serveurs de catalogue et de conteneur.

Surveillance à l'aide de la console Web

Avec la console Web, vous pouvez générer des graphiques des statistiques actuelles et historiques. Cette console fournit un certain nombre de graphiques préconfigurés pour des présentations générales et elle comporte une page de rapports personnalisés que vous pouvez utiliser pour élaborer des graphiques à partir des statistiques disponibles. Les fonctionnalités graphiques de la console de surveillance de WebSphere eXtreme Scale permettent de visualiser les performances globales des grilles des données présentes dans votre environnement.

Installez la console Web comme fonction facultative lorsque vous exécutez l'assistant d'installation.

1. Démarrez le serveur de la console. Le script **startConsoleServer.bat|sh** de démarrage du serveur de la console se trouve dans le répertoire *racine_install_wxs*/ObjectGrid/bin de votre installation.

2. Connectez-vous à la console.
   a. Dans votre navigateur Web, accédez à `https://your.console.host:7443`, en remplaçant `your.console.host` par le nom de l'hôte du serveur sur lequel vous avez installé la console.
   b. Connectez-vous à la console.
      - **ID utilisateur :** `admin`
      - **Mot de passe :** `admin`

      La page d'accueil de la console s'affiche.
3. Modifiez la configuration de la console. Cliquez sur **Paramètres** > **Configuration** pour afficher la configuration de la console. La configuration de la console comprend ce type d'informations :
   - la chaîne de trace pour le client WebSphere eXtreme Scale, comme `*=all=disabled`
   - le nom et le mot de passe de l'administrateur
   - son adresse e-mail
4. Créez et maintenez des connexions aux serveurs de catalogue que vous voulez surveiller. Répétez les étapes suivantes pour ajouter chaque serveur de catalogue à la configuration.
   a. Cliquez sur **Paramètres** > **Serveurs de catalogue eXtreme Scale**.
   b. Ajoutez un nouveau serveur de catalogue.

      1) Cliquez sur l'icône Ajouter () pour enregistrer un serveur de catalogue existant.
      2) Fournissez des informations, telles que le nom d'hôte et le port d'écoute. Voir Planification des ports réseau pour plus d'informations sur la configuration des ports et les valeurs par défaut.
      3) Cliquez sur **OK**.
      4) Vérifiez que le serveur de catalogue a bien été ajouté à l'arborescence de navigation.
5. Visualisez le statut de la connexion La zone **Domaine en cours** indique le nom du domaine de services de catalogue qui est actuellement utilisé pour afficher des informations dans la console Web. L'état de la connexion s'affiche en regard du nom du domaine de services de catalogue.
6. Affichez les statistiques des grilles de données et des serveurs ou créez un rapport personnalisé.

Surveillance avec l'utilitaire xscmd

1. Ouvrez une fenêtre de ligne de commande. Sur la ligne de commande, définissez les variables d'environnement appropriées.
   a. Définissez la variable d'environnement *CLIENT_AUTH_LIB* :
      - Windows `set CLIENT_AUTH_LIB=<path_to_security_JAR_or_classes>`
      - UNIX `set CLIENT_AUTH_LIB=<path_to_security_JAR_or_classes>`
        `export CLIENT_AUTH_LIB`
2. Accédez au répertoire *rép_base_wxs*`/bin`.

   `cd rép_base_wxs/bin`
3. Exécutez plusieurs commandes pour afficher des informations sur votre environnement.

- Afficher tous les serveurs de conteneur en ligne pour la grille de données de la grille et le groupe de mappes mapSet :
  ```
  xscmd -c showPlacement -g Grid -ms mapSet
  ```
- Afficher les informations de routage de la grille de données.
  ```
  xscmd -c routetable -g Grid
  ```
- Afficher le nombre d'entrées de mappe dans la grille de données.
  ```
  xscmd -c showMapSizes -g Grid -ms mapSet
  ```

Arrêt des serveurs

Une fois que vous avez fini d'utiliser l'application client et de surveiller l'exemple d'environnement du guide de démarrage, vous pouvez arrêter les serveurs.

- Si vous avez utilisé les fichiers script pour démarrer les serveurs, utilisez <ctrl+c> pour arrêter le processus de service de catalogue et les serveurs de conteneur dans les fenêtres correspondantes.
- Si vous avez utilisé la commande **startOgServer** pour démarrer vos serveurs, utilisez la commande **stopOgServer** pour les arrêter.

  Arrêtez le serveur de conteneur :

  – UNIX Linux
  ```
  stopOgServer.sh c0 -catalogServiceEndPoints
  localhost:2809
  ```
  – Windows
  ```
  stopOgServer.bat c0 -catalogServiceEndPoints
  localhost:2809
  ```

  Arrêtez le serveur de conteneur :

  – UNIX Linux
  ```
  stopOgServer.sh cs1 -catalogServiceEndPoints
  localhost:2809
  ```
  – Windows
  ```
  stopOgServer.bat cs1 -catalogServiceEndPoints
  localhost:2809
  ```

Point de contrôle de la leçon

Dans cette leçon, vous avez appris à :

- démarrer la console Web et la connecter au serveur de catalogue ;
- surveiller les statistiques de la grille et des serveurs ;
- arrêter les serveurs.

Initiation au développement d'applications

Pour commencer à développer des applications WebSphere eXtreme Scale, configurez n environnement de développement dans Eclipse.

Pourquoi et quand exécuter cette tâche

Lorsque vous développez des applications WebSphere eXtreme Scale , vous pouvez utiliser les API de serveur intégrées pour créer et démarrer les serveurs, les instances ObjectGrid, et pour insérer des données dans la grille de données. Vous pouvez tester les unités de votre application et la configuration associée directement dans l'environnement Eclipse.

Lorsque vous êtes prêt à transférer votre application vers un environnement plus large, vous pouvez créer des fichiers XML de configuration que vous importez pour créer votre déploiement.

Procédure

1. Configurez un environnement de développement dans Eclipse.

   En ajoutant les fichiers JAR (Java archive) WebSphere eXtreme Scale à l'environnement de développement, vous pouvez commencer à utiliser les API pour développer vos applications.

   Informations complémentaires : «Configuration d'un environnement de développement autonome», à la page 125

2. Créer une application simple qui démarre les serveurs, crée une instance ObjectGrid, et insère des données dans la grille de données.

   a. Utilisez l'API ServerFactory pour démarrer et arrêter les serveurs.

      Informations complémentaires : Utilisation de l'API de serveur embarqué pour démarrer et arrêter les serveurs

   b. Utilisez l'API ObjectGridManager pour extraire l'instance ObjectGrid que vous avez créée.

      Informations complémentaires : «Interaction avec un objet ObjectGrid en utilisant l'interface ObjectGridManager», à la page 136

   c. Utilisez l'API ObjectMap pour insérer des données dans la grille de données.

      Informations complémentaires : «Mise en cache d'objets sans aucune relation impliquée (API ObjectMap)», à la page 155L'API ObjectMap est le moyen le plus simple pour accéder à la grille de données et y écrire des données. Si vos objets ont des relations complexes, vous pouvez utiliser les API suivantes pour lire, écrire, et mettre à jour les données :

      - «Accès aux données avec des index (API Index)», à la page 144
      - «Mise en cache d'objets et de leurs relations (API EntityManager)», à la page 166
      - «Extraction d'entités et d'objets (API Query)», à la page 202
      - «Accès aux données avec le service de données REST», à la page 268

      Pour plus d'informations sur le choix entre les différentes API, voir Chapitre 5, «Développement d'applications», à la page 131.

3. Testez les unités de l'application.

   Vous pouvez également utiliser l'utilitaire **xscmd** pour afficher des informations sur les serveurs en cours d'exécution, les répliques, etc. Pour plus d'informations, voir Administration avec l'utilitaire **xscmd**.

4. Lorsque vous êtes satisfait de votre application dans l'environnement de développement, créez des fichiers de configuration XML et mettez à jour votre application pour qu'elle utilise la configuration. L'exemple d'application d'initiation fournit des exemples de ces fichiers de configuration et une application Java simple qui utilise les fichiers de configuration.

   Informations complémentaires :«Tutoriel : Démarrer avec WebSphere eXtreme Scale», à la page 61

5. Exécutez votre application en utilisant les fichiers de configuration XML. La façon dont vous démarrez vos serveurs dépend de l'environnement que vous utilisez.

   Vous pouvez exécuter votre application dans l'un des conteneurs suivants :

   - Machine JVM (Java Virtual Machine) autonome
   - Tomcat
   - WebSphere Application Server
   - OSGi

Concepts associés:

«Mise en cache d'objets sans aucune relation impliquée (API ObjectMap)», à la page 155
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.

Information associée:

«Leçon 2 du tutoriel d'initiation : Création d'une application client», à la page 63
Pour pouvoir insérer, supprimer, mettre à jour et extraire des données dans votre grille de données, vous devez écrire une application client. L'exemple d'initiation inclut une application client que vous pouvez utiliser pour en savoir plus sur la création de votre propre application client.

Chapitre 4. Planification

Avant d'installer WebSphere eXtreme Scale et de déployer vos applications de grille de données, vous devez choisir votre topologie de mise en cache, planifier la capacité, vérifier les configurations matérielle et logicielle requises et les paramètres de réseau et d'optimisation, etc. Vous pouvez également utiliser la liste de contrôle opérationnelle pour vérifier que votre environnement est prêt pour le déploiement d'applications.

Vous trouverez une discussion des pratiques recommandées pour la conception d'applications WebSphere eXtreme Scale dans l'article suivant de developerWorks : Principles and best practices for building high performing and highly resilient WebSphere eXtreme Scale applications.

Planification de la topologie

Avec WebSphere eXtreme Scale, l'architecture de votre système peut utiliser la mise en cache des données locales en mémoire ou la mise en cache des données client-serveur réparties. L'architecture peut avoir des relations différentes avec vos bases de données. Vous pouvez également configurer la topologie pour l'étendre à plusieurs centres de données.

WebSphere eXtreme Scale requiert une infrastructure supplémentaire minimale pour pouvoir fonctionner. Cette infrastructure consiste en des scripts permettant d'installer, de démarrer et d'arrêter une application Java Platform, Enterprise Edition sur un serveur. Les données mises en cache sont stockées dans les serveurs de conteneur et les clients se connectent à distance au serveur.

Environnements internes

Lors du déploiement dans un environnement interne, WebSphere eXtreme Scale s'exécute dans une seule machine virtuelle Java et il n'est pas répliqué. Pour configurer un environnement local, vous pouvez utiliser un fichier XML ObjectGrid ou les API ObjectGrid.

Environnement réparti

Lorsque vous effectuez le déploiement dans un environnement réparti, WebSphere eXtreme Scale s'exécute dans un ensemble de machines virtuelles Java, ce qui améliore les performances, la disponibilité et l'évolutivité. Dans cette configuration, vous pouvez utiliser les fonctions de réplication et de partitionnement des données. Vous pouvez également ajouter d'autres serveurs sans redémarrer les serveurs eXtreme Scale existants. Comme dans le cas d'un environnement local, un fichier XML ObjectGrid ou une configuration par programmation équivalente est nécessaire dans un environnement réparti. Vous devez également fournir un fichier XML de stratégie de déploiement contenant les détails de la configuration.

Il est possible de créer des déploiements simples ou des déploiements plus vastes se chiffrant en téraoctets et comptant plusieurs milliers de serveurs.

Cache interne local

Dans le cas le plus simple, WebSphere eXtreme Scale peut être utilisé comme cache de grille de données locale (non répartie) en mémoire. Cette mise en cache locale peut s'avérer particulièrement utile pour les applications au nombre d'accès simultanés élevé où plusieurs unités d'exécution doivent accéder aux données temporaires et les modifier. Les données conservées dans une grille de données locale peuvent être indexées et extraites à l'aide de requêtes. Les requêtes permettent d'utiliser des jeux de données volumineux en mémoire. Le support fourni avec machine virtuelle Java (JVM), qui est prêt à être utilisé, dispose d'une structure de données limitées.

La topologie de cache local en mémoire de WebSphere eXtreme Scale permet d'octroyer un accès cohérent et transactionnel aux données temporaires dans une machine virtuelle Java unique.

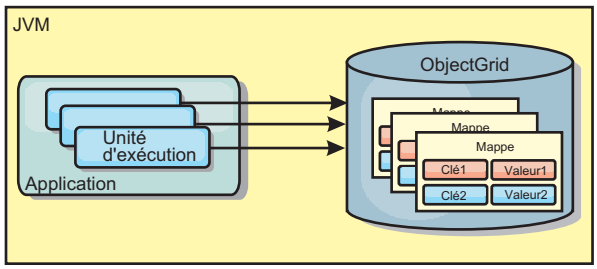


Figure 4. Scénario de cache local en mémoire

Avantages

- Configuration simple : une ObjectGrid peut être créée à l'aide d'un programme ou de manière déclarative avec le fichier XML du descripteur de déploiement ObjectGrid ou à l'aide d'une autre structure telle que Spring.
- Rapide : chaque mappe de sauvegarde peut être ajustée de façon indépendante pour optimiser l'utilisation de la mémoire et des accès simultanés.
- Configuration idéale pour les topologies de machine virtuelle Java dotées de petits jeux de données ou pour la mise en cache de données fréquemment consultées.
- Transactionnelle. Les mises à jour de mappe de sauvegarde peuvent être regroupées dans la même unité d'oeuvre et peuvent être intégrées en dernier lieu aux transactions constituées de deux phases telles que les transactions JTA (Java Transaction Architecture).

Inconvénients

- Aucune tolérance de panne.
- Les données ne sont pas répliquées. Les mémoires cache internes se prêtent aux données de référence en lecture seule.
- Non évolutive. La quantité de mémoire requise par la base de données peut dépasser la capacité de la machine virtuelle Java.
- Problèmes survenant lors de l'ajout de machines virtuelles Java :
  - Les données ne peuvent pas être facilement partitionnées ;
  - Nécessité de répliquer manuellement l'état entre les machines virtuelles Java ou chaque instance de cache peut présenter différentes versions des mêmes données.

- L'invalidation est coûteuse.
- Chaque cache doit être préchauffé indépendamment. Le préchauffage est la période de chargement d'un jeu de données permettant de remplir le cache avec des données valides.

Utilisation

La topologie de déploiement de la mémoire cache interne locale ne doit être utilisée que lorsque la quantité de données à mettre en cache est limitée (peut être abritée par une seule machine virtuelle Java) et est relativement stable. Cette approche doit tolérer les données obsolètes. L'utilisation d'expulseurs pour conserver les données les plus fréquemment ou récemment utilisées dans le cache peut contribuer à réduire la taille du cache et à accroître la pertinence des données.

Cache local répliqué sur des homologues

Vous devez vous assurer que le cache est synchronisé si plusieurs processus avec des instances indépendantes de cache existent. Pour vérifier que les instances de cache sont synchronisées, activez un cache répliqué sur des homologues avec JMS (Java Message Service).

WebSphere eXtreme Scale comprend deux plug-in qui propagent automatiquement les modifications de transactions entre les instances ObjectGrid homologues. Le plug-in JMSObjectGridEventListener propage automatiquement les modifications eXtreme Scale à l'aide de JMS.

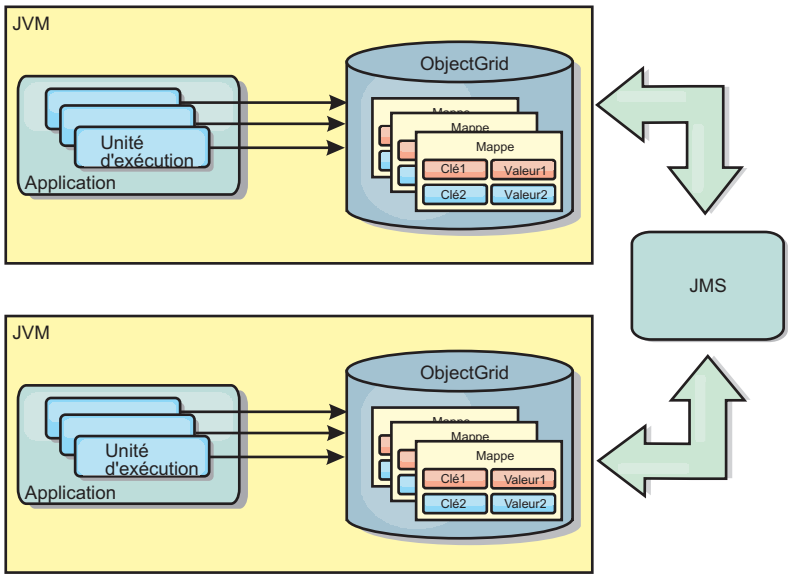


Figure 5. Cache répliqué sur des homologues avec des modifications qui sont propagées à l'aide de JMS

Si vous exécutez un environnement WebSphere Application Server, le plug-in TranPropListener est aussi disponible. Il utilise le gestion HA (high availability) pour propager les modifications à chaque instance de cache homologue.

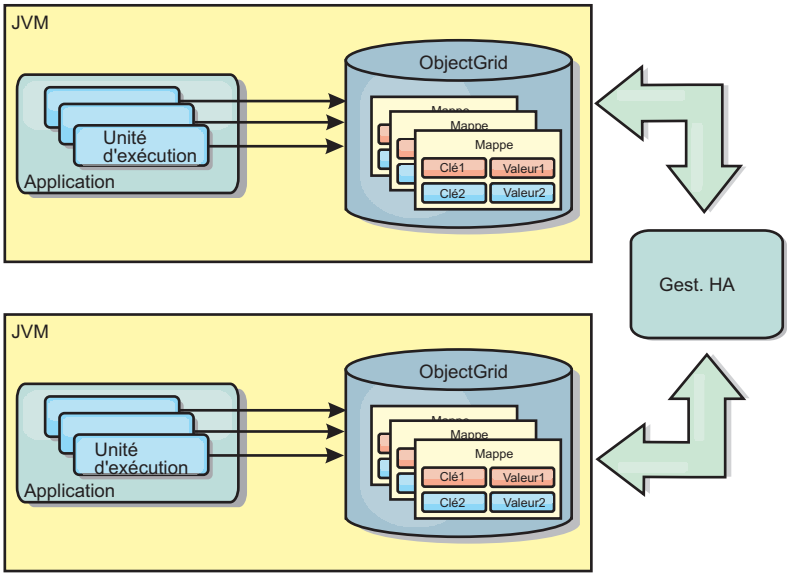


Figure 6. Cache répliqué sur des homologues avec des modifications qui sont propagées à l'aide du gestionnaire de haute disponibilité

Avantages

- Plus grande validité des données car celles-ci sont actualisées plus souvent.
- Avec le plug-in TranPropListener, tout comme avec l'environnement local, il est possible de créer la grille de données eXtreme Scale par programmation ou de manière déclarative avec le fichier XML du descripteur de déploiement d'eXtreme Scale ou avec d'autres structures de travail comme Spring. L'intégration au gestionnaire de haute disponibilité s'effectue automatiquement.
- Chaque mappe de sauvegarde peut être optimisée indépendamment en termes d'utilisation de la mémoire et de simultanéité des accès.
- Il est possible de regrouper en une seule unité d'oeuvre les mises à jour des mappes de sauvegarde qui peuvent être intégrées comme derniers participants de transactions en deux phases comme le sont les transactions Java Transaction Architecture (JTA).
- Idéal pour les topologies comprenant un nombre restreint de machines virtuelles Java avec un dataset de taille raisonnablement réduite ou pour la mise en cache des données à accès fréquent.
- Les modifications de la grille de données eXtreme Scale sont répliquées à toutes les instances eXtreme Scale homologues. Les modifications sont cohérentes tant qu'un abonnement durable est utilisé.

Inconvénients

- La configuration et la maintenance du plug-in JMSObjectGridEventListener peut s'avérer une tâche complexe. Il est possible de créer la grille de données eXtreme Scale par programmation ou de manière déclarative avec le fichier XML du descripteur de déploiement d'eXtreme Scale ou avec d'autres structures de travail comme Spring.
- Pas d'extensibilité : la quantité de mémoire requise par la base de données risque de submerger la machine virtuelle Java.
- Fonctionne de manière incorrecte lorsqu'on ajoute des machines virtuelles Java :
  - les données ne sont pas facilement partitionnées
  - l'invalidation est onéreuse

– chaque cache doit être prérempli de manière indépendante

Quand l'utiliser

Utilisez la topologie de déploiement uniquement lorsque la quantité de données à mettre en cache est faible, peut tenir sur une seule machine virtuelle Java, et relativement stable.

Cache imbriqué

Les grilles WebSphere eXtreme Scale peuvent s'exécuter dans des processus existants, tels que des serveurs eXtreme Scale intégrés ou vous pouvez les gérer comme des processus externes.

Les grilles imbriquées sont utiles lorsque l'exécution se fait dans un serveur d'applications tel que WebSphere Application Server. Vous pouvez démarrer les serveurs eXtreme Scale non imbriqués à l'aide de scripts de ligne de commande et les exécuter dans un processus Java.

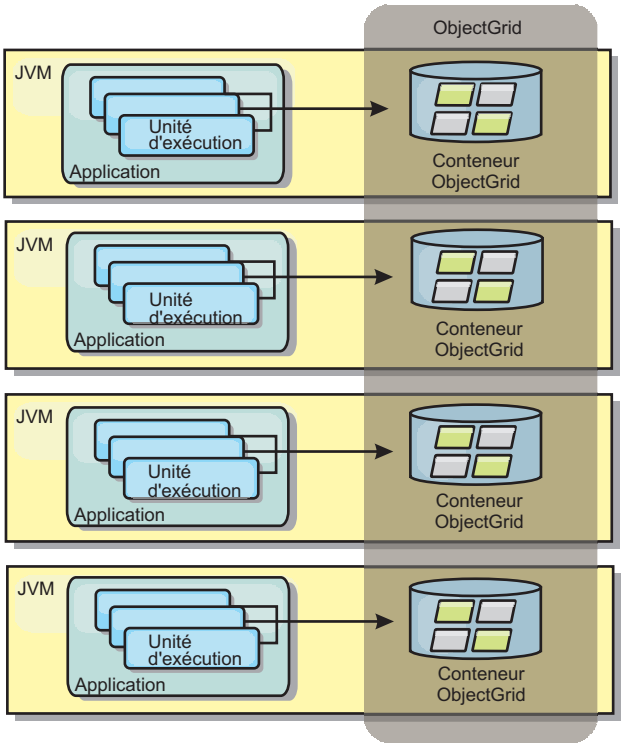


Figure 7. Cache imbriqué

Avantages

- simplification de l'administration en raison du nombre inférieur de processus à gérer
- simplification du déploiement d'application car la grille utilise le chargeur de classe de l'application client
- prise en charge du partitionnement et de la haute disponibilité.

Inconvénients

- augmentation de l'encombrement mémoire dans le processus client car toutes les données sont regroupées dans le processus
- augmentation de l'utilisation de l'unité centrale en vue de la gestion des demandes des clients
- plus grande difficulté à gérer les mises à niveau des applications car les clients utilisent les mêmes fichiers d'archive Java que les serveurs
- moindre flexibilité. Les clients et les serveurs de grille ne peuvent évoluer au même rythme. Lorsque des serveurs sont définis en externe, la gestion du nombre de processus devient plus flexible

Utilisation

Utilisez les grilles imbriquées lorsqu'une grande quantité de mémoire est disponible dans le processus client pour les données de la grille et pour les données de basculement.

Plus d'informations, voir la rubrique relative à l'activation du mécanisme d'invalidation de client dans *Guide d'administration*.

Cache réparti

La plupart du temps, WebSphere eXtreme Scale est utilisé en tant que cache partagé permettant un accès transactionnel aux données de plusieurs composants là où une base de données classique aurait été nécessaire. Avec le cache partagé, il n'est plus nécessaire de configurer une base de données.

Cohérence de la mémoire cache

Le cache est cohérent car tous les clients y voient les mêmes données. Chaque donnée est stockée dans le cache sur un seul serveur ce qui permet d'éviter la coexistence de plusieurs copies d'enregistrements risquant de contenir des versions différentes des données. Un cache cohérent contient un nombre croissant de données au fur et à mesure que l'on ajoute des serveurs à la grille et le cache évolue de manière linéaire au fur et à mesure que la taille de la grille augmente. Comme les clients accèdent aux données de cette grille de données avec des appels de procédure distante, cette mémoire est également appelée cache distant ou éloigné. Grâce au partitionnement des données, chaque processus contient un sous-ensemble unique de données. Les grandes grilles peuvent contenir davantage de données et traiter plus de demandes pour ces données. Par ailleurs la cohérence évite d'avoir à envoyer les données d'invalidation autour de la grille de données, car aucune donnée périmée n'existe. Le cache cohérent contient uniquement la copie la plus récente de chaque donnée.

Si vous exécutez un environnement WebSphere Application Server, le plug-in TranPropListener est aussi disponible. Il utilise le composant de haute disponibilité (gestionnaire HA) de WebSphere Application Server pour propager les modifications à chaque instance de cache ObjectGrid homologue.

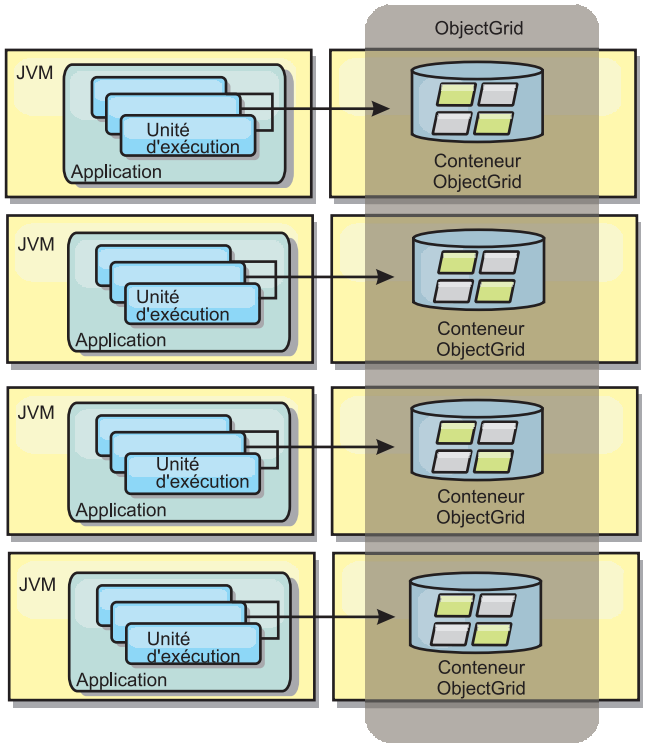


Figure 8. Cache réparti

Cache local

Lorsqu'eXtreme Scale est utilisé dans le cadre d'une topologie répartie, les clients peuvent éventuellement disposer d'un cache local en ligne. L'on appelle cache local ce cache facultatif. Il s'agit d'un ObjectGrid indépendant, présent sur chaque client et faisant office de cache du cache distant côté serveur. Il est activé par défaut lorsque le verrouillage est configuré sur OPTIMISTIC ou sur NONE. Son utilisation est impossible lorsque le verrouillage est configuré sur PESSIMISTIC.

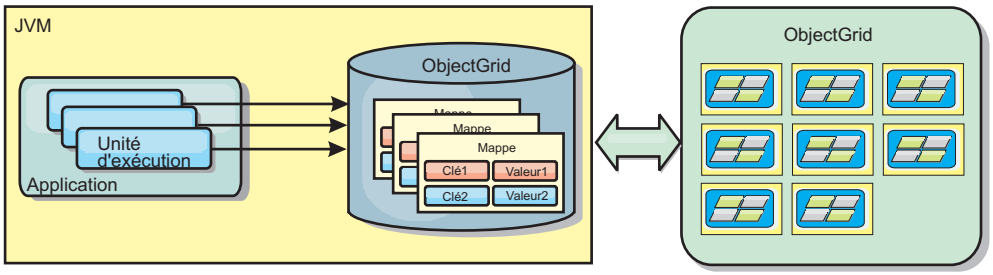


Figure 9. Cache local

Le cache local est très rapide car il offre un accès en mémoire à un sous-ensemble des données stockées à distance sur les serveurs eXtreme Scale. Il n'est pas partitionné et contient des données provenant de n'importe quelle partition eXtreme Scale distante. Jusqu'à trois groupes de caches peuvent exister dans WebSphere eXtreme Scale :

1. Le cache du groupe des transactions contient toutes les modifications apportées à une même transaction. Il contient une copie de travail des données jusqu'à ce que la transaction soit validée. Lorsqu'une transaction client demande des données à une ObjectMap, la transaction est vérifiée en priorité.

2. Le cache local du groupe des clients contient un sous-ensemble des données du groupe des serveurs. Lorsque le groupe des transactions ne contient pas les données, les données sont extraites du niveau client, si elles sont disponibles et insérées dans le cache des transactions.
3. La grille de données dans le groupe des serveurs contient la majorité des données et elle est partagée entre tous les clients. Le groupe des serveurs peut être partitionné, ce qui permet la mise en cache d'un grand nombre de données. Lorsque le cache local ne contient pas de données, celles-ci sont extraites du groupe des serveurs et insérées dans le cache du client. Le groupe des serveurs peut aussi avoir un plug-in Loader. Lorsque la grille ne contient pas les données demandées, le chargeur est appelé et les données résultantes sont insérées dans la grille à partir du magasin de données dorsal.

Pour désactiver le cache local, donnez la valeur 0 à l'attribut numberOfBuckets dans la configuration du descripteur eXtreme Scale des remplacements par le client. Pour plus d'informations sur les stratégies de verrouillage dans eXtreme Scale, consultez la rubrique relative au verrouillage des entrées de mappe. Le cache local peut également être configuré de façon à utiliser d'autres règles d'expulsion et des plug-in différents qui utilisent une configuration de descripteur eXtreme Scale des remplacements par les clients.

Avantage

- Rapidité du temps de réponse, car tous les accès aux données se font localement La recherche de données dans le cache local évite de consulter la grille des serveurs et rend les données distantes accessibles localement.

Inconvénients

- Augmentation de la durée des données obsolètes, car le cache local à chaque niveau est peut-être désynchronisé avec les données en cours dans la grille de données.
- Basé sur un expulseur pour invalider les données afin d'éviter de manquer de mémoire.

Utilisation

A utiliser lorsque le temps de réponse est élevé et que la présence de données périmées est tolérée.

Intégration de la base de données : caches avec écriture différée, caches en ligne et caches secondaires

WebSphere eXtreme Scale est utilisé pour servir de frontal à une base de données classiques et ainsi éliminer l'activité de lecture qui est normalement envoyée vers la base de données. Un cache cohérent peut être utilisé avec une application soit directement, soit indirectement en passant alors par un associateur relationnel d'objets (ORM). Le cache cohérent peut décharger des tâches de lecture la base de données ou le dorsal. Dans un scénario un tout petit peu plus complexe, comme celui d'un accès transactionnel à un dataset dans lequel seules certaines données requièrent des garanties de persistance classique, il est possible d'utiliser le filtrage pour décharger même les transactions d'écriture.

Vous pouvez configurer WebSphere eXtreme Scale pour qu'il fonctionne en tant qu'espace extrêmement flexible de traitement de base de données interne. Cela dit, WebSphere eXtreme Scale n'est pas un associateur relationnel d'objets. Il ne sait pas d'où les données de la grille de données proviennent. Une application ou un

associateur relationnel d'objets peuvent placer des données sur un serveur eXtreme Scale. C'est à la source de données qu'il incombe de vérifier la cohérence des données avec leur base de données d'origine. En d'autres termes, eXtreme Scale ne peut pas invalider les données qu'il a extraites automatiquement d'une base de données. C'est à l'application ou à l'associateur de fournir cette fonction et de gérer les données stockées dans eXtreme Scale.

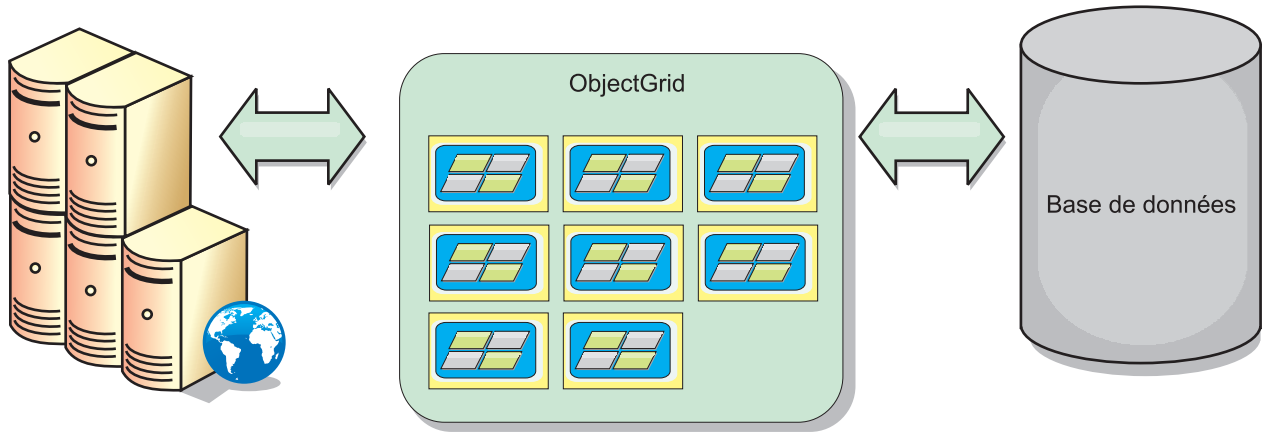


Figure 10. ObjectGrid en tant que mémoire tampon de base de données

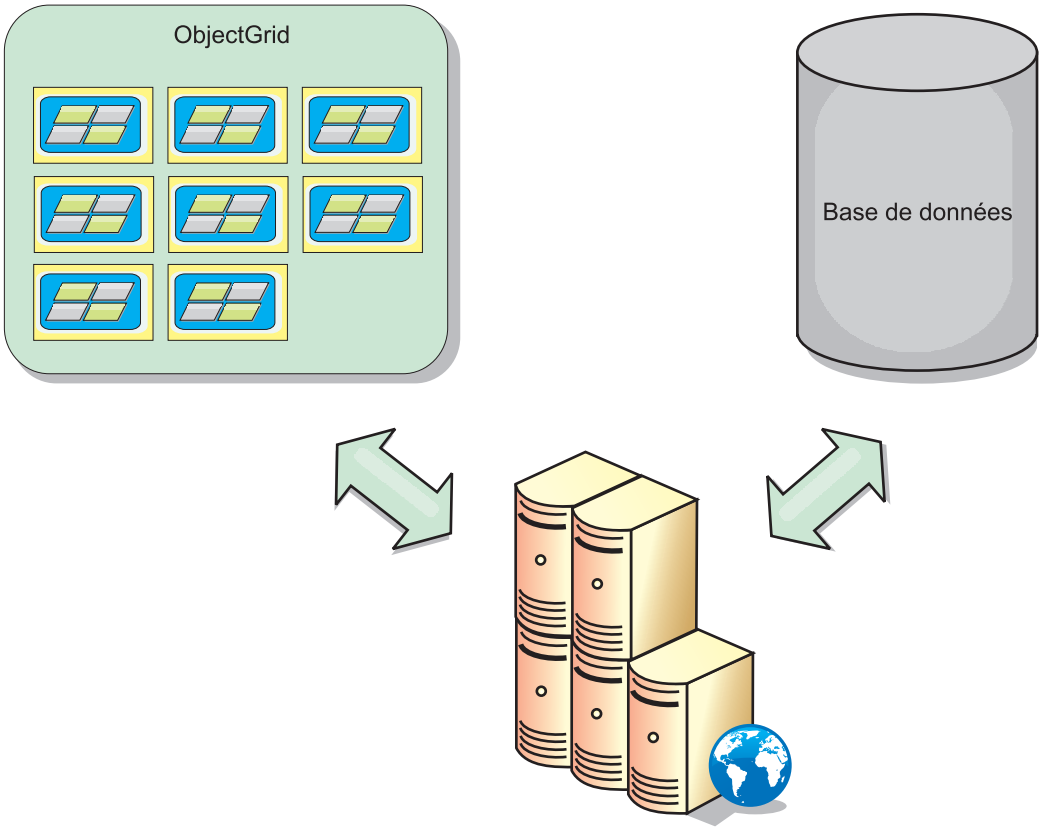


Figure 11. ObjectGrid en tant que cache secondaire

Cache partiel et cache complet

WebSphere eXtreme Scale peut s'utiliser en tant que cache partiel ou que cache complet. Un cache partiel ne conserve qu'un sous-ensemble des données totales,

alors qu'un cache complet conserve toutes les données et peut être rempli en différé en fonction des besoins en données. Les caches partiels sont normalement accessibles à l'aide de clés (et non pas d'index ou de requêtes), car les données sont partiellement disponibles uniquement.

Cache partiel

Si une clé est absente dans un cache partiel ou que les données ne sont pas disponibles et qu'un échec de cache se produit, le niveau suivant est appelé. Les données sont extraites d'une base de données, par exemple, et elles sont insérées au groupe de caches de grille de données. Si vous utilisez une requête ou un index, seules les valeurs actuellement chargées sont accessibles et les requêtes ne sont pas transférées aux autres groupes.

Cache complet

Un cache complet comporte toutes les données requises et il est possible d'y accéder à l'aide d'attributs non-clés avec des index ou des requêtes. Un cache complet est préchargé avec des données de la base de données avant que l'application tente d'accéder aux données. Un cache complet peut fonctionner sous la forme d'un remplacement de base de données une fois que les données sont chargées. Etant donné que toutes les données sont disponibles, les requêtes et les index peuvent être utilisés pour rechercher et agréger les données.

Cache secondaire

Lorsque WebSphere eXtreme Scale est utilisé en tant que cache secondaire, le système dorsal est utilisé avec la grille de données.

Cache secondaire

Vous pouvez configurer le produit en tant que cache secondaire pour la couche d'accès aux données d'une application. Dans ce scénario, WebSphere eXtreme Scale permet de stocker temporairement des objets qui seraient normalement extraits d'une base de données dorsale. Les applications vérifient si la grille de données contient les données. Si les données se trouvent dans la grille de données, ces données sont renvoyées à l'appelant. Si elles n'existent pas, elles sont extraites de la base de données dorsale. Elles sont ensuite insérées dans la grille de données afin que la demande suivante puisse utiliser la copie mise en cache. Le diagramme suivant montre comment WebSphere eXtreme Scale peut être utilisé en tant que cache secondaire à l'aide d'une couche d'accès aux données arbitraire, telle qu'OpenJPA ou Hibernate.

Plug-in de cache secondaire pour Hibernate et OpenJPA

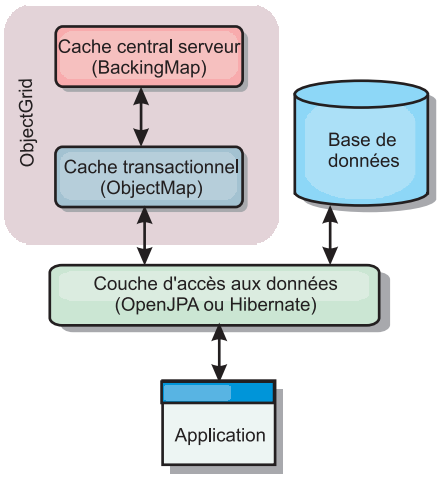


Figure 12. Cache secondaire

Les plug-in de cache pour OpenJPA et Hibernate sont inclus dans WebSphere eXtreme Scale pour que vous puissiez utiliser le produit comme cache secondaire automatique. L'utilisation d'WebSphere eXtreme Scale en tant que fournisseur de cache améliore les performances lors de la lecture et de l'interrogation des données et réduit la charge pesant sur la base de données. WebSphere eXtreme Scale présente plusieurs avantages par rapport à des implémentations de cache pré-intégrées car le cache est automatiquement répliqué entre tous les processus. Lorsqu'un client met une valeur en mémoire cache, tous les autres clients peuvent l'utiliser.

Cache en ligne

Vous pouvez configurer la mise en cache en ligne pour un système dorsal de base de données ou en tant que cache secondaire pour une base de données. La mise en cache en ligne utilise eXtreme Scale comme moyen principal pour interagir avec les données. Lorsque eXtreme Scale est utilisé en tant que cache en ligne, l'application interagit avec le système dorsal à l'aide d'un plug-in Loader.

Cache en ligne

Lorsque WebSphere eXtreme Scale est utilisé en tant que cache en ligne, il interagit avec le système dorsal à l'aide d'un plug-in Loader. Ce scénario permet de simplifier l'accès aux données car les applications peuvent accéder aux API eXtreme Scale directement. Plusieurs scénarios de cache sont pris en charge dans eXtreme Scale pour assurer la synchronisation des données dans le cache et des données dans le système dorsal. Le diagramme suivant illustre l'interaction entre le cache en ligne, l'application et le système dorsal.

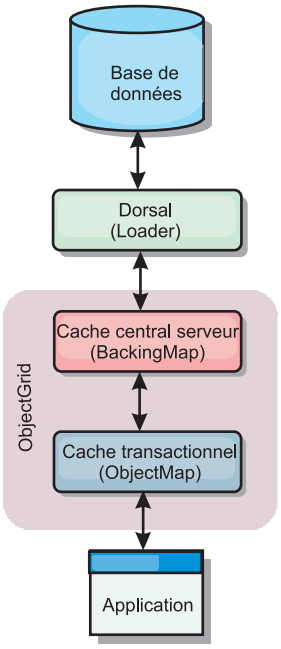


Figure 13. Cache en ligne

L'option de mise en cache en ligne simplifie l'accès aux données en permettant aux applications d'accéder directement aux API eXtreme Scale. WebSphere eXtreme Scale prend en charge plusieurs scénarios de mise en cache en ligne, comme suit.

- Sans interruption
- Ecriture immédiate
- Post-écriture

Scénario de mise en cache sans interruption

Un cache sans interruption est un cache partiel chargeant en lazy loading à partir d'une clé les entrées de données au fur et à mesure que ces entrées sont demandées. Cette opération peut se dérouler sans que l'appelant sache comment sont renseignées les entrées. Si les données sont introuvables dans le cache eXtreme Scale, eXtreme Scale récupère les données manquantes auprès du plug-in Loader qui charge les données provenant de la base de données d'arrière plan et les insère dans le cache. Les requêtes suivantes pour la même clé de données se trouveront dans le cache, jusqu'à ce qu'elles soient supprimées, invalidées ou expulsées.

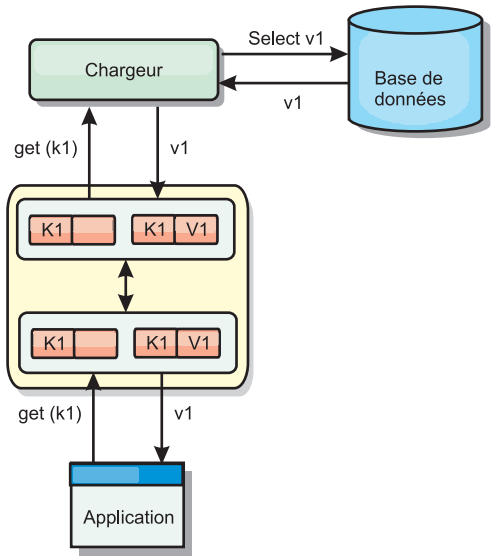


Figure 14. Mise en cache sans interruption

Scénario de mise en cache à écriture immédiate

Dans un cache à écriture immédiate, chaque écriture dans le cache est inscrite de manière synchrone dans la base de données à l'aide du chargeur. Cette méthode permet la cohérence avec le système dorsal, mais réduit les performances d'écriture étant donné que l'opération de base de données est synchrone. Le cache et la base de données étant tous deux mis à jour, les lectures suivantes à la recherche des mêmes données auront lieu dans le cache, évitant ainsi de faire appel à la base de données. Un cache à écriture immédiate est souvent utilisé conjointement à un cache sans interruption.

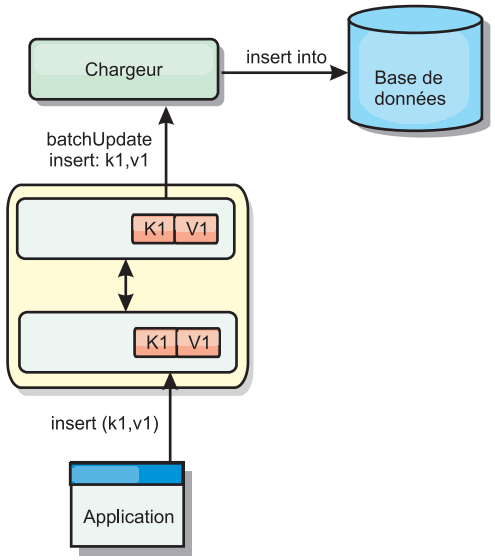


Figure 15. Mise en cache à écriture immédiate

Scénario de mise en cache en écriture différée

La synchronisation de la base de données peut être améliorée en écrivant les modifications de manière asynchrone. Cette opération est appelée mise en cache en

écriture différée. Les modifications, normalement écrites de manière synchrone dans le chargeur, sont mises en mémoire tampon dans eXtreme Scale et écrites dans la base de données à l'aide d'une unité d'exécution en arrière-plan. Les performances d'écriture sont considérablement améliorées, car l'opération de base de données est supprimée de la transaction client et les écritures de la base de données peuvent être comprimées.

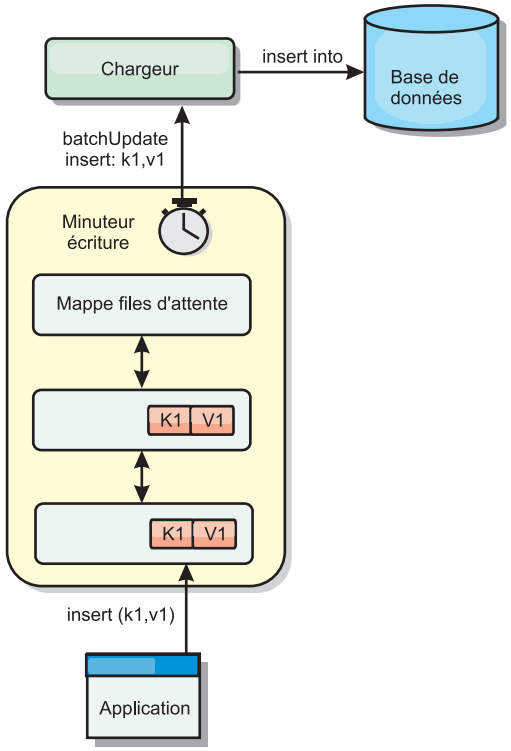


Figure 16. Mise en cache en écriture différée

Mise en cache en écriture différée

Vous pouvez utiliser la mise en cache en écriture différée pour réduire le temps système supplémentaire nécessaire lors de la mise à jour d'une base de données utilisée en tant que base de données dorsale.

Présentation de la mise en cache en écriture différée

La mise en cache en écriture différée met en file d'attente de manière asynchrone les mises à jour du plug-in Loader. Vous pouvez améliorer les performances en déconnectant les mises à jour, les insertions et les suppressions au sein d'une mappe, le temps système pour la mise à jour de la base de données dorsale. La mise à jour asynchrone est effectuée après un retard (de cinq minutes, par exemple) ou après un certain nombre d'entrées (1 000 entrées).

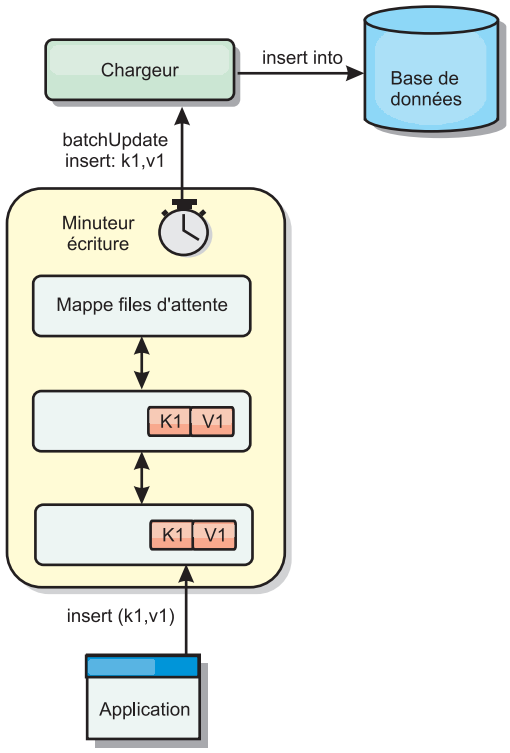


Figure 17. Mise en cache en écriture différée

La configuration à écriture différée sur une mappe de sauvegarde crée une unité d'exécution entre le Loader et la mappe. Le Loader délègue alors les demandes de données via l'unité d'exécution en fonction des paramètres de configuration de la méthode BackingMap.setWriteBehind. Lorsqu'une transaction eXtreme Scale insère, met à jour ou supprime une entrée dans une mappe, un objet LogElement est créé pour chacun de ces enregistrements. Ces éléments sont envoyés au Loader à écriture différée et mis en file d'attente dans une ObjectMap spéciale appelée mappe de files d'attente. Chaque mappe de sauvegarde pour laquelle le paramètre d'écriture différée est activé a ses propres mappes de files d'attente. L'unité d'exécution à écriture différée supprime périodiquement les données mises en file d'attente des mappes correspondantes et les insère dans le Loader dorsal.

Le chargeur à écriture différée envoie uniquement les types insertion, mise à jour et suppression des objets LogElement au chargeur réel. Tous les autres types, par exemple le type EVICT, sont ignorés.

La prise en charge de l'écriture différée *est* une extension du plug-in Loader, qui vous permet d'intégrer eXtreme Scale à la base de données. A ce sujet, vous pouvez consulter avec profit les explications Configuration des chargeurs JPA sur la configuration d 'un chargeur JPA.

Avantages

L'activation de l'écriture différée présente les avantages suivants :

- **Isolement en cas d'arrêt anormal de la base de données dorsale :** la mise en cache à écriture différée propose une couche d'isolement en cas d'arrêt anormal de la base de données dorsale. Les mises à jour sont alors placées dans la mappe de files d'attente. Les applications peuvent continuer à envoyer des transactions

vers eXtreme Scale. Lors de la reprise du système dorsal, les données contenues dans la mappe de files d'attente sont insérées dans celui-ci.

- **Réduction de la charge du système dorsal :** le chargeur à écriture différée fusionne les mises à jour en fonction des clés de façon qu'une seule mise à jour fusionnée par clé existe dans la mappe de files d'attente. Cette fusion diminue le nombre de mises à jour dans la base de données dorsale.
- **Amélioration des performances de la transaction :** la durée de chaque transaction eXtreme Scale est réduite car la transaction n'a plus à attendre que les données soient synchronisées avec le système dorsal.

Considérations liées à la conception d'applications

L'activation de l'écriture différée est une opération simple, mais la création d'une application devant utiliser l'écriture différée requiert une attention particulière. Sans écriture différée, la transaction ObjectGrid encadre la transaction dorsale. La transaction ObjectGrid démarre avant la transaction dorsale et se termine après celle-ci.

Lorsque la prise en charge de l'écriture différée est activée, la transaction ObjectGrid se termine avant le début de la transaction dorsale. La transaction ObjectGrid et la transaction dorsale sont dissociées.

Contraintes d'intégrité référentielle

Chaque mappe de sauvegarde configurée avec écriture différée dispose de sa propre unité d'exécution d'écriture différée pour envoyer les données vers le système dorsal. Les données mises à jour dans les différentes mappes d'une transaction ObjectGrid sont mises à jour dans le système dorsal via différentes transactions dorsales. Par exemple, la transaction T1 met à jour la clé key1 dans la mappe Map1 et la clé key2 dans la mappe Map2. La clé key1 mise à jour dans la mappe Map1 est actualisée vers le système dorsal dans une transaction expéditrice et la clé key2 actualisée dans la mappe Map2 l'est dans une autre transaction expéditrice ; cette mise à jour vers le dorsal est effectuée dans deux unités d'exécution différentes, toutes deux en écriture différée. Si les données stockées dans Map1 et Map2 ont des liens, tels que des contraintes de clé externe dans le système dorsal, les mises à jour sont susceptibles d'échouer.

Lors de la conception de contraintes d'intégrité référentielle dans votre base de données dorsale, vérifiez que les mises à jour désordonnées sont autorisées.

Verrouillage d'une mappe de files d'attente

Le comportement des transactions en matière de verrouillage constitue une autre différence notable. La grille d'objets prend en charge trois stratégies de verrouillage différentes : PESSIMISTIC, OPTIMISITIC et NONE. Les mappes de files d'attente à écriture différée utilisent la stratégie de verrouillage pessimiste, quelle que soit la stratégie de verrouillage configurée pour leur mappe de sauvegarde. Deux types différents d'opérations permettant d'acquérir un verrou sur la mappe de files d'attente existent :

- Lorsqu'une transaction ObjectGrid est validée ou qu'un vidage (de mappe ou de session) se produit, la transaction lit la clé de la mappe de files d'attente et place un verrou S sur la clé.
- Lorsqu'une transaction ObjectGrid est validée, elle tente de mettre à niveau le verrou S vers un verrou X sur la clé.

Ce comportement de mappe de files d'attente supplémentaire vous permet de voir quelques différences dans le comportement de verrouillage.

- Si la mappe de l'utilisateur est configurée comme stratégie de verrouillage pessimiste, la différence dans le comportement de verrouillage n'est pas grande. Chaque fois qu'un vidage ou qu'une validation est appelée, un verrou S est placé sur la même clé dans la mappe de files d'attente. Au moment de la validation, un verrou X est acquis pour la clé dans la mappe de l'utilisateur, mais également pour la clé dans la mappe de files d'attente.
- Si la mappe de l'utilisateur est configurée comme stratégie de verrouillage optimiste ou inexistante, la transaction utilisateur suit le modèle de la stratégie pessimiste. Chaque fois qu'un vidage ou qu'une validation est appelée, un verrou S est acquis sur la même clé dans la mappe de files d'attente. Au moment de la validation, un verrou X est acquis pour la clé dans la mappe de files d'attente utilisant la même transaction.

Nouvelles tentatives de transaction du chargeur

ObjectGrid ne prend pas en charge les transactions à 2 phases ou transactions XA. L'unité d'exécution à écriture différée supprime les enregistrements de la mappe de files d'attente et met à jour les enregistrements dans le système dorsal. En cas d'échec du serveur au milieu de la transaction, certaines mises à jour dorsales risquent d'être perdues.

Le chargeur à écriture différée tente automatiquement d'écrire à nouveau les transactions ayant échoué et envoie une LogSequence en attente de validation au système dorsal pour éviter toute perte de données. Pour que l'exécution de cette action soit possible, le chargeur doit être idempotent, ce qui signifie que lorsque Loader.batchUpdate(TxId, LogSequence) est appelé deux fois avec la même valeur, le résultat est le même que s'il était appelé une fois. Les implémentations du chargeur doivent implémenter l'interface RetryableLoader pour activer cette fonction. Pour plus de détails, consultez la documentation relative à l'API.

Echecs du chargeur

Le plug-in du chargeur (Loader) risque d'échouer lorsqu'il ne parvient pas à communiquer avec le dorsal de base de données. Cela peut se produire si le serveur de base de données ou la connexion réseau est arrêté(e). Le chargeur en écriture différée met en file d'attente les mises à jour et tente d'envoyer régulièrement les données modifiées au chargeur. Ce dernier doit signaler le problème de connectivité à l'environnement d'exécution ObjectGrid en générant une exception LoaderNotAvailableException.

L'implémentation du chargeur doit donc pouvoir distinguer un échec lié aux données d'une défaillance physique du chargeur. En cas d'échec lié aux données, une exception LoaderException ou OptimisticCollisionException doit être générée, alors qu'en cas de défaillance physique du chargeur, une exception LoaderNotAvailableException doit être générée. ObjectGrid gère ces deux exceptions de manière différente :

- Si une exception LoaderException est interceptée par le chargeur en écriture différée, celui-ci considère que l'échec est dû à une défaillance de données, telle qu'une erreur de clé en double. Le chargeur dégroupe la mise à jour et tente de ne mettre à jour qu'un enregistrement à la fois, afin d'isoler la défaillance de données. Si une exception {{LoaderException}} est à nouveau détectée lors de la mise à jour de l'enregistrement concerné, un enregistrement d'échec de la mise à jour est créé et consigné dans la mappe des mises à jour ayant échoué.

- Si une exception LoaderNotAvailableException est interceptée par le chargeur en écriture différée, celui-ci considère que l'échec est dû à l'impossibilité de se connecter à la base de données, par exemple, lorsque la base de données dorsale est inactive, lorsque la connexion à une base de données est indisponible ou lorsque le réseau est inactif. Le chargeur attend 15 secondes, puis tente à nouveau la mise à jour par lots de la base de données.

L'erreur courante est d'émettre une exception LoaderException à la place d'une exception LoaderNotAvailableException. Tous les enregistrements du chargeur à écriture différée deviennent alors des enregistrements d'échec de la mise à jour, ce qui réduit à néant l'objectif de l'isolement en cas d'arrêt anormal du système dorsal.

Remarques sur les performances

En supprimant de la transaction la mise à jour du chargeur, la mise en cache en écriture différée augmente le temps de réponse. Elle augmente également la capacité de traitement de la base de données, car les mises à jour de base de données sont combinées. Il est important de comprendre le temps système supplémentaire généré par l'unité d'exécution à écriture différée, qui permet de retirer les données de la mappe de files d'attente et de les insérer dans le chargeur.

Le temps de mise à jour maximal et le nombre de mises à jour maximal doivent être ajustés en fonction de l'environnement et des types d'utilisation prévus. Si la valeur du temps ou du nombre de mises à jour maximal est trop petite, le temps système de l'unité d'exécution d'écriture différée peut dépasser les avantages tirés. Définir une valeur élevée pour ces deux paramètres peut également augmenter l'utilisation de la mémoire pour la mise en file d'attente des données et retarder le moment de péremption des enregistrements de bases de données.

Pour des performances optimales, réglez les paramètres d'écriture différée en fonction des facteurs suivants :

- ratio des transactions de lecture et d'écriture
- fréquence de mise à jour d'enregistrements identiques
- temps d'attente pour la mise à jour de la base de données

Référence associée:

«Exemple : écriture d'une classe dumper en écriture différée», à la page 362
Cet exemple de code source montre comment écrire un programme de surveillance (dumper) pour gérer les mises à jour d'écriture différée ayant échoué.

Chargeurs

Avec un plug-in Loader, une mappe de grille de données peut se comporter comme un cache pour les données généralement conservées dans un magasin persistant 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) peut également être utilisée comme source des données, ce qui permet de créer des caches basés sur un concentrateur à l'aide d'eXtreme Scale. Un chargeur peut lire et écrire des données vers un stockage persistant ou à partir de celui-ci.

Présentation

Les chargeurs 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). Le chargeur est appelé lorsque le cache ne peut pas satisfaire une demande de clé, offrant ainsi une fonction de lecture et un remplissage laborieux du cache. Un chargeur permet également les mises à jour de la base de données lorsque les valeurs du cache viennent à changer. Toutes les modifications dans une transaction sont regroupées pour réduire le nombre d'interactions de base de données. Un plug-in TransactionCallback est utilisé conjointement avec le chargeur pour déclencher la démarcation de la transaction principale. L'utilisation de ce plug-in est importante lorsque plusieurs mappes sont incluses dans une seule transaction ou lorsque les données de transaction sont vidées dans le cache sans validation.

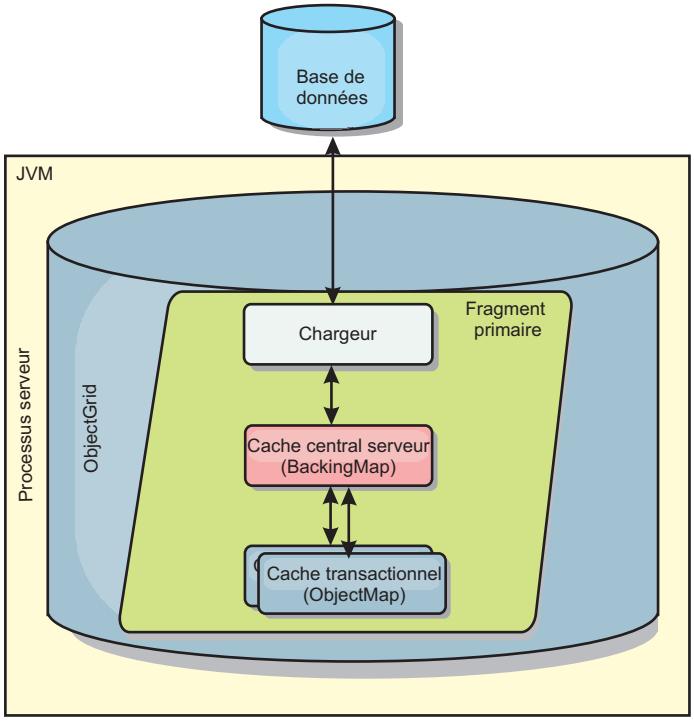


Figure 18. Chargeur

Le chargeur peut donc utiliser les mises à jour sur-qualifiées pour éviter le verrouillage intempestif de la base de données. En stockant un attribut de version dans la valeur du cache, le chargeur peut distinguer l'image de la valeur avant et après la mise à jour dans le cache. Cette valeur peut ensuite être utilisée lors de la mise à jour de la base de données ou du programme d'arrière plan pour vérifier que les données n'ont pas été mises à jour. Un chargeur peut également être configuré pour précharger la grille de données lorsqu'elle démarre. Lorsqu'elle est partitionnée, une instance de chargeur est associée à chaque partition. Si la mappe de la société comporte dix partitions, il existe dix instances de chargeur, une pour chaque partition principale. Lorsque le fragment primaire de la mappe est activé, la méthode preloadMap du chargeur est appelée de manière synchrone ou asynchrone, ce qui déclenche le chargement automatique de la partition de la mappe avec les données du programme d'arrière plan. Lorsqu'il est appelé de manière synchrone, toutes les transactions client sont bloquées, ce qui empêche tout accès incohérent à la grille de données. Sinon, un préchargeur client peut être utilisé pour charger l'intégralité de la grille de données.

Deux chargeurs pré-intégrés peuvent simplifier considérablement l'intégration aux dorsaux de bases de données relationnelles. Les chargeurs JPA utilisent les

fonctions du mappage objet-relationnel(ORM) des implémentations OpenJPA et Hibernate des spécifications JPA (Java Persistence API). Pour plus d'informations, voir «Chargeurs JPA», à la page 396.

Si vous utilisez des chargeurs dans une configuration à plusieurs centre de données, vous devez étudier la façon dont les données de révision et la cohérence de la mémoire cache est conservée entre les grilles de données. Pour plus d'informations, voir «Remarques sur les chargeurs dans une topologie multimaître» , à la page 106.

Configuration de chargeur

Pour ajouter un chargeur à la configuration BackingMap, vous pouvez utiliser la configuration à l'aide d'un programme ou la configuration XML. Un chargeur a la relation suivante avec une mappe de sauvegarde.

- Une mappe de sauvegarde peut avoir un seul chargeur.
- Une mappe de sauvegarde client (cache local) 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.

Référence associée:

«Remarques sur la programmation de chargeurs JPA», à la page 365
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.

Préchargement et préremplissage des données

Dans la plupart des scénarios qui utilise un chargeur, vous pouvez préparer la grille de données en y préchargeant ses données.

Lorsque vous utilisez la grille de données comme un cache complet, elle doit contenir toutes les données et elle doit être chargée pour que les clients puissent s'y connecter. Lorsque vous utilisez un cache partiel, vous pouvez préparer le cache avec des données pour que les clients puissent avoir accès immédiatement à ces données dès qu'ils se connectent.

Il existe deux approches pour pré-charger des données dans la grille de données ; vous pouvez utiliser un plug-in Loader ou un chargeur client, comme décrit dans les sections suivantes.

Plug-in Loader

Le plug-in Loader est associé à chaque mappe et chargé de synchroniser un fragment primaire de partition avec la base de données. La méthode preloadMap du plug-in Loader est invoquée automatiquement lors de l'activation d'un fragment, Par exemple, vous disposez de 100 partitions, il existe 100 instances Loader, chacune chargeant les données de sa partition. En cas d'exécution synchrone, tous les clients sont bloqués jusqu'à la fin du préchargement.

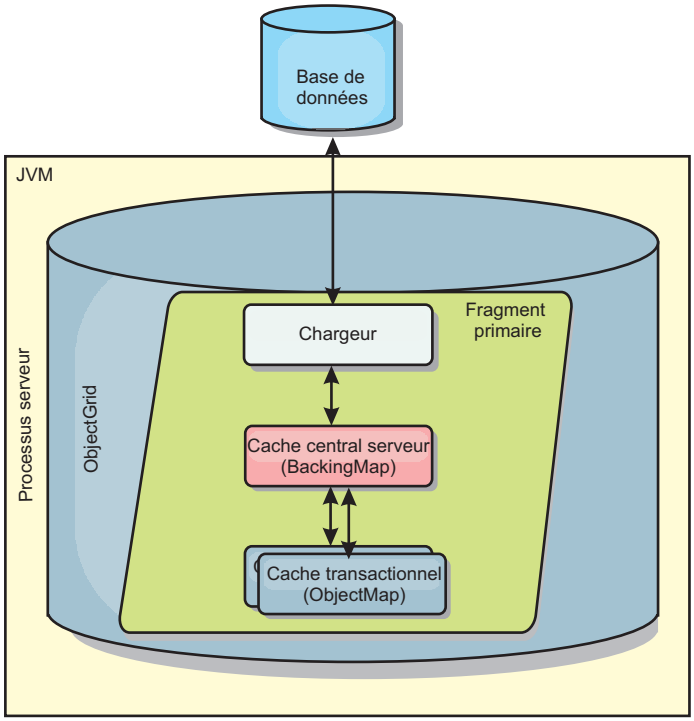


Figure 19. Plug-in Loader

Loader client

Un loader client est un pattern d'utilisation d'un ou plusieurs clients pour charger les données dans la grille. L'utilisation de plusieurs clients pour charger les données de la grille peut s'avérer efficace lorsque le schéma de partition n'est pas stocké dans la base de données. Vous pouvez appeler des chargeurs de client manuellement ou automatiquement lorsque la grille de données démarre. Ces chargeurs peuvent éventuellement utiliser StateManager pour faire passer la grille de données en mode de préchargement pour que les clients ne puissent pas accéder à la grille lorsqu'elle précharge les données. WebSphere eXtreme Scale contient un chargeur JPA (Java Persistence API) que vous pouvez utiliser pour charger automatiquement la grille de données avec le fournisseur JPA OpenJPA ou Hibernate. Pour plus d'informations sur les fournisseurs de cache, voir Plug-in de cache niveau 2 (L2) JPA.

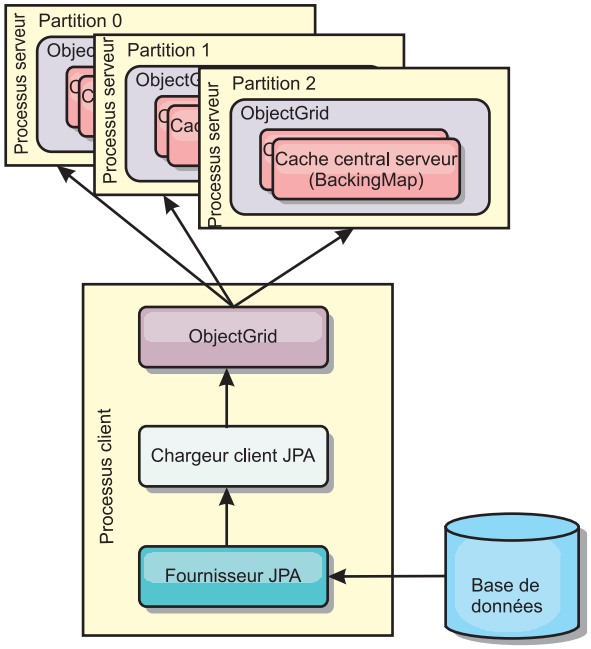


Figure 20. Loader client

Méthodes de synchronisation de base de données

Lorsque WebSphere eXtreme Scale est utilisé en tant que cache, les applications doivent être écrites de sorte qu'elles tolèrent les données périmées si la base de données peut être mise à jour de manière indépendante par rapport à une transaction eXtreme Scale. En tant qu'espace de traitement de base de données en mémoire synchronisé, eXtreme Scale permet d'assurer la mise à jour du cache de plusieurs manières.

Méthodes de synchronisation de base de données

Actualisation régulière

Le cache peut être régulièrement invalidé ou mis à jour de manière automatique à l'aide du programme de mise à jour temporelle de base de données JPA (Java Persistence API).Le programme de mise à jour interroge régulièrement la base de données à l'aide d'un fournisseur JPA, afin de rechercher des mises à jour ou des insertions survenues depuis la mise à jour précédente. Tous les changements détectés sont automatiquement invalidés ou mis à jour lorsqu'ils sont utilisés avec un cache incomplet. S'ils sont utilisés avec un cache complet, les entrées peuvent être détectées et insérées dans le cache. Les entrées ne sont jamais supprimées du cache.

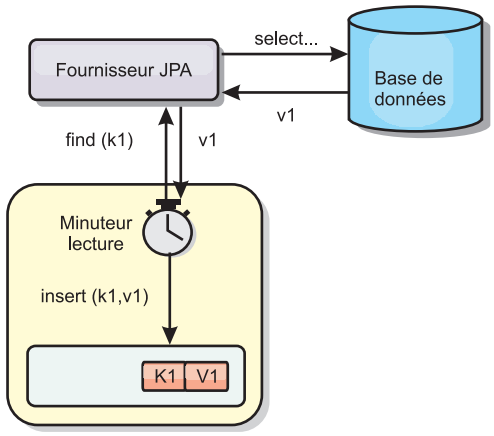


Figure 21. Actualisation régulière

Suppression

Les caches incomplets peuvent utiliser les stratégies de suppression pour supprimer automatiquement les données du cache sans que cela n'affecte la base de données. eXtreme Scale inclut trois stratégies : durée de vie, utilisation la moins récente et utilisation la moins fréquente. Si l'option de suppression en fonction de la mémoire est activée, ces trois stratégies suppriment les données de manière plus agressive à mesure que la mémoire est limitée.

Invalidation en fonction d'événements

Il est possible d'invalider les caches partiels et complets à l'aide d'un générateur d'événements comme JMS (Java Message Service). L'invalidation par le biais de JMS peut être associée de manière manuelle à tout processus qui met à jour le dorsal à l'aide d'un déclencheur de base de données. eXtreme Scale contient un plug-in JMS ObjectGridEventListener qui informe les clients des éventuelles modifications du cache du serveur. Cette procédure peut réduire la durée d'accès du client aux données périmées.

Invalidation par programme

Les API eXtreme Scale permettent l'interaction manuelle du cache local et du cache serveur à l'aide des méthodes des API Session.beginNoWriteThrough(), ObjectMap.invalidate() et EntityManager.invalidate(). Si un processus client ou serveur n'a plus besoin d'une partie des données, les méthodes d'invalidation peuvent être utilisées pour supprimer les données du serveur local ou du serveur cache. La méthode beginNoWriteThrough applique une opération ObjectMap ou EntityManager au cache local sans appeler le programme de chargement. Si l'opération est appelée à partir d'un client, elle s'applique uniquement au cache local (le programme de chargement distant n'est pas appelé). Si elle est appelée sur le serveur, l'opération s'applique uniquement au cache central du serveur sans appeler le programme de chargement.

L'invalidation des données

Pour supprimer les données de la mémoire cache d'évolution, vous pouvez utiliser un mécanisme d'invalidation basé sur les événements ou à l'aide d'un programme.

Invalidation basée sur les événements

Il est possible d'invalider les caches incomplets et complets à l'aide d'un générateur d'événements tel que Java Message Service (JMS). L'invalidation par le biais de JMS peut être associée de manière manuelle à tout processus qui met à jour le dorsal à l'aide d'un déclencheur de base de données. Un plug-in JMS ObjectGridEventListener est fourni dans eXtreme Scale pour permettre aux clients d'être informés des modifications dans le cache du serveur. Ce type de notification peut réduire la durée d'accès du client aux données obsolètes.

L'invalidation basée sur les événements est composée normalement des trois composants suivants.

- **File d'attente des événements :** une file d'attente d'événements stocke les événements de modification des données. Il peut s'agir d'une file d'attente JMS, d'une base de données, d'une file d'attente interne premier entré premier sorti ou de tout autre événement dans la mesure où elle peut gérer les événements de modification des données.
- **Publicateur d'événements :** un publicateur d'événements publie les événements de modification de données dans la file d'attente d'événements. Une publicateur d'événements est généralement une application que vous créez ou une implémentation de plug-in eXtreme Scale. Il sait quand les données ont été modifiées ou il modifie les données lui-même. Lorsqu'une transaction est validée, les événements sont générés pour les données modifiées et le publicateur d'événements publie ces événements dans la file d'attente d'événements.
- **Consommateur d'événements :** un consommateur d'événements consomme les événements de modification de données. Le consommateur d'événements est généralement une application permettant de vérifier la mise à jour des données de la grille cible avec les dernières modifications apportées aux autres grilles. Il interagit avec la file d'attente d'événements pour récupérer les dernières données et applique les modifications apportées aux données dans la grille cible. Les consommateurs d'événements peuvent utiliser les API eXtreme Scale pour invalider les données obsolètes ou mettre à jour la grille avec les dernières données.

Par exemple, JMSObjectGridEventListener comporte une option pour un modèle client-serveur dans lequel la file d'attente d'événements est une destination JMS désignée. Tous les processus serveur sont des publicateurs d'événements. Lorsqu'une transaction est validée, le serveur récupère les modifications apportées aux données et les publie à la destination JMS désignée. Tous les processus client sont des consommateurs d'événements. Ils reçoivent les modifications apportées aux données de la destination JMS désignée et appliquent les modifications au cache local du client.

Pour plus d'informations, consultez la rubrique relative au mécanisme d'invalidation de client dans le *Guide de l'administration* .

Invalidation par programme

Les API WebSphere eXtreme Scale autorise l'interaction manuelle du cache local et du cache serveur à l'aide des méthodes Session.beginNoWriteThrough(), ObjectMap.invalidate() et EntityManager.invalidate(). Si un processus client ou serveur n'a plus besoin d'une partie des données, les méthodes invalidate permettent de supprimer des données d'un cache local ou de serveur. La méthode beginNoWriteThrough applique toutes les opérations ObjectMap ou EntityManager

au cache local sans appeler le chargeur. Si l'opération est appelée à partir d'un client, elle s'applique uniquement au cache local (le programme de chargement distant n'est pas appelé). Si elle est appelée sur le serveur, l'opération s'applique uniquement au cache central du serveur sans appeler le programme de chargement.

Vous pouvez utiliser l'invalidation par programme à l'aide d'autres techniques pour déterminer quand il convient d'invalider les données. Par exemple cette méthode d'invalidation utilise des mécanismes d'invalidation basée sur les événements pour recevoir les événements de modification de données, puis utilise les API pour invalider les données obsolètes.

Indexation

Utilisez le plug-in MapIndexPlugin pour générer un ou plusieurs index dans une mappe BackingMap pour prendre en charge l'accès aux données ne correspondant pas à une clé.

Types d'indexation et configuration d'index

L'indexation est représentée par le plug-in MapIndexPlugin ou Index, en bref. Index est un plug-in BackingMap. Une mappe de sauvegarde peut avoir plusieurs index configurés, dès lors que chacun d'entre eux respecte les règles de configuration d'index.

Vous pouvez utiliser l'indexations pour générer une ou plusieurs index dans une mappe BackingMap. Un index se construit à partir d'un attribut ou d'une liste des attributs d'un objet de la mappe. L'indexation permet aux applications de trouver plus rapidement certains objets. Grâce à elle, en effet, les applications peuvent trouver les objets dont les attributs indexés ont une certaine valeur ou se situent dans une plage de valeurs.

Deux types d'indexation sont possibles : statiques et dynamiques. L'indexation statique oblige à configurer le plug-in d'indexation index dans la mappe de sauvegarde avant d'initialiser l'instance ObjectGrid. Comme pour la mappe de sauvegarde, cela peut se faire par programmation ou via XML. L'indexation statique commence à générer l'index pendant l'initialisation de la grille d'objets. L'index est synchrone en permanence avec la mappe de sauvegarde et il est prêt à être utilisé. Après que l'indexation statique a démarré, la maintenance de l'index fait partie de la gestion des transactions par eXtreme Scale. Lorsque les transactions valident leurs modifications, ces dernières actualisent également l'index statique et les modifications apportées à l'index sont annulées en cas d'annulation de la transaction.

L'indexation dynamique permet de créer un index dans une mappe de sauvegarde avant ou après l'initialisation de l'instance ObjectGrid qui contient cette mappe. Les applications contrôlent le cycle de vie de l'indexation dynamique, ce qui permet de supprimer un index dynamique devenu inutile. Lorsqu'une application crée un index dynamique, cet index n'est pas forcément utilisable immédiatement en raison du temps que met à s'effectuer la génération complète de l'index. Comme la durée dépend de la quantité de données indexées, l'interface DynamicIndexCallback est fournie pour les applications qui souhaitent recevoir des notifications lorsque se produisent certains événements l'indexation, à savoir les événements ready, error et destroy. Les applications peuvent implémenter cette interface de rappel et s'enregistrer auprès de l'indexation dynamique.

Si un plug-in d'indexation est configuré pour une mappe de sauvegarde, il est possible d'obtenir de la mappe d'objet correspondante l'objet proxy de l'index. L'appel de la méthode getIndex dans la mappe et la transmission du nom du plug-in Index renvoie l'objet proxy de l'index. L'objet proxy doit être transtypé vers l'interface d'indexation de l'application utilisée, MapIndex, MapRangeIndex ou une interface d'indexation personnalisée, par exemple. Une fois l'objet proxy obtenu, l'on peut utiliser les méthodes définies dans l'interface d'indexation de l'application afin de trouver des objets mis en cache.

La liste qui suit récapitule la procédure à appliquer pour procéder à l'indexation :

- ajout d'index statiques ou dynamiques dans la mappe de sauvegarde
- obtention d'un objet proxy d'index grâce à la méthode getIndex de la mappe d'objet
- transtypage de l'objet proxy vers l'interface d'indexation de l'application utilisée (MapIndex, MapRangeIndex ou une interface d'indexation personnalisée, par exemple)
- utilisation des méthodes qui sont définies dans l'interface d'indexation de l'application pour rechercher les 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.

Index par défaut

Si vous souhaitez effectuer une itération dans les clés d'une mappe locale, vous pouvez utiliser l'index par défaut. Cet index ne requiert pas de configuration, mais elle doit être utilisée sur le fragment en utilisant un agent ou une instance ObjectGrid extraite de la méthode ShardEvents.shardActivated(ObjectGrid shard).

Indexation et qualité des données obtenues par une requête d'index

Il faut bien avoir présent à l'esprit que les méthodes de requêtes sur les index ne représentent qu'un cliché des données à un instant t. Les entrées de données ne sont pas verrouillées après l'envoi à l'application des résultats de la requête. L'application doit être consciente que les données peuvent très bien être actualisées après lui avoir été retournées. Supposons, par exemple, que l'application obtienne la clé d'un objet mis en cache grâce à la méthode findAll de MapIndex. Cet objet key retourné est associé dans le cache à une entrée de données. L'application doit être capable d'exécuter la méthode get sur la mappe d'objet pour trouver un objet à partir de l'objet key. Si une autre transaction supprime du cache l'objet données juste avant l'appel à la méthode get, le résultat qui sera retourné sera null.

Points à prendre en considération à propos des performances de l'indexation

L'un des objectifs primordiaux de l'indexation est d'améliorer les performances globales de la mappe de sauvegarde. Une utilisation incorrecte de l'indexation peut compromettre les performances de l'application. Avant d'utiliser l'indexation, les facteurs suivants sont à prendre en considération :

- **Le nombre de transactions simultanées en écriture :** l'indexation peut se produire chaque fois qu'une transaction écrit des données dans une mappe de

sauvegarde. Les performances se dégradent si un grand nombre de transactions écrivent en même temps des données dans la mappe au moment où une application lance des requêtes sur l'index.

- **La taille des résultats retournés par une requête :** les performances de la requête déclinent d'autant plus que la taille de ses résultats augmente. Les performances tendent à se dégrader lorsque la taille des résultats atteint 15 % ou plus de la mappe de sauvegarde.
- **Le nombre d'index générés sur la même mappe de sauvegarde :** chaque index consomme des ressources système. Les performances diminuent au fur et à mesure que le nombre d'index augmente sur la mappe de sauvegarde.

Cela dit, l'indexation peut augmenter considérablement les performances des mappes de sauvegarde. C'est particulièrement vrai lorsque la mappe de sauvegarde comporte surtout des opérations de lecture. Les résultats des requêtes représentent alors un faible pourcentage des entrées de la mappe et seul un petit nombre d'index sont générés sur la mappe.

Tâches associées:

«Configuration du plug-in HashIndex», à la page 327
Vous pouvez configurer le plug-in HashIndex intégré, la classe com.ibm.websphere.objectgrid.plugins.index.HashIndex avec un fichier XML à l'aide d'un programme ou d'une annotation d'entité dans une mappe d'entité.

«Accès aux données avec des index (API Index)», à la page 144
Utilisez l'indexation pour améliorer l'accès aux données.

Référence associée:

«Attributs du plug-in HashIndex», à la page 329
Vous pouvez utiliser les attributs suivants pour configurer le plug-in HashIndex Ces attributs définissent des propriétés comme si vous utilisiez un attribut ou un index HashIndex composite ou que l'indexation de plage était activée.

Planification de plusieurs topologies de centre de données

En utilisant la réplication asynchrone multimaître, au moins deux grilles de données peuvent devenir des copies exactes de l'une de l'autre. Chaque grille de données est hébergée dans un domaine de services de catalogue indépendant, avec ses propres de service de catalogue, serveurs de conteneur et un nom unique. Avec la réplication asynchrone multimaître, vous pouvez utiliser des liaisons pour connecter un ensemble de domaine de services de catalogue. Les domaine de services de catalogue sont ensuite synchronisés en utilisant la réplication via ces liaisons. Vous pouvez construire quasiment n'importe quelle topologie via la définition de liaisons entre les domaine de services de catalogue.

Tâches associées:

Configuration de plusieurs topologies de centres de données
Avec la réplication asynchrone multimaître, vous liez un ensemble de domaines de services de catalogue. Les domaines de services de catalogue connectés sont ensuite synchronisés en utilisant la réplication via les liaisons. Vous pouvez définir les liaisons à l'aide de fichiers de propriétés, lors de l'exécution avec des programmes JMX (Java Management Extensions) ou avec les utilitaires de ligne de commande. Le groupe de liaisons actuel d'un domaine est stocké dans le service de catalogue. Vous pouvez ajouter et supprimer des liens sans redémarrer le domaine de services de catalogue qui héberge la grille de données.

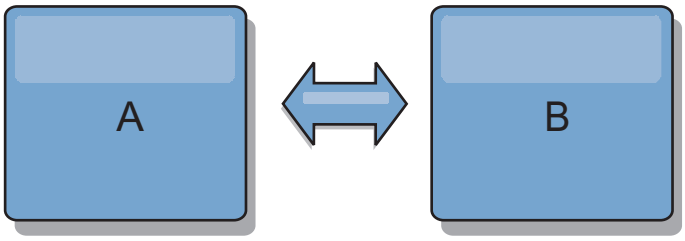
«Développement d'arbitres personnalisés pour la réplication multi-maître», à la page 303
Des collisions entre des modifications peuvent se produire s'il est possible à des enregistrements identiques d'être modifiés simultanément en deux endroits différents. Dans une topologie de réplication multimaître, les domaines de services de catalogue détectent les collisions automatiquement. Lorsqu'un domaine de services de catalogue détecte une collision, il démarre un arbitre. En général, les collisions sont résolues avec l'arbitre de collision par défaut. Mais une application peut très bien fournir un arbitre personnalisé.

Topologies pour la réplication multimaître

Vous disposez de plusieurs options pour choisir la topologie de votre déploiement qui intègre la réplication multimaître.

Liaisons connectant des domaine de services de catalogue

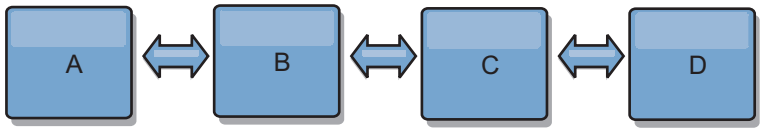
Une infrastructure de grilles de données de réplication est un graphique de domaine de services de catalogue interconnectés avec des liaisons bidirectionnelles. Avec une liaison, deux domaine de services de catalogue peuvent communiquer les modifications de données. Par exemple, la topologie la plus simple est une paire de domaine de services de catalogue avec une liaison unique entre eux. Les domaine de services de catalogue sont nommés par ordre alphabétique: A, B, C, etc., à partir de la gauche. Une liaison peut traverser un réseau WAN (wide area network) pour couvrir une grande distance. Même si la liaison est interrompue, vous pouvez toujours modifier les données dans l'un des domaine de services de catalogue. La topologie rapproche les modifications quand la liaison reconnecte les domaine de services de catalogue. Les liens tentent automatiquement de se reconnecter si la connexion réseau est interrompue.



Après avoir établi les liaisons, eXtreme Scale tente d'abord de rendre chaque domaine de services de catalogue identique. Ensuite, eXtreme Scale tente de maintenir identiques les conditions à mesure que des modifications se produisent dans un domaine de services de catalogue. L'objectif vise à faire de chaque domaine de services de catalogue le miroir exact d'un autre domaine de services de catalogue connecté par les liaisons. Les liaisons de réplication entre les domaine de services de catalogue permettent copier une modification effectuée dans un domaine vers les autres domaines.

Topologies linéaires

Même s'il s'agit d'un déploiement simple, une topologie linéaire montre certaines qualités des liaisons. Tout d'abord, il n'est pas nécessaire qu'un domaine de services de catalogue soit connecté directement à tous les autres domaine de services de catalogue pour pouvoir recevoir les modifications. Le domaine B extrait les modifications du domaine A. Le domaine C reçoit les modifications du domaine A via le domaine B, lequel connecte les domaines A et C. De même, le domaine D reçoit les modifications des autres domaines via le domaine C. De ce fait, la charge de la répartition des modifications est distribuée et elle n'incombe plus à la seule source de ces modifications.



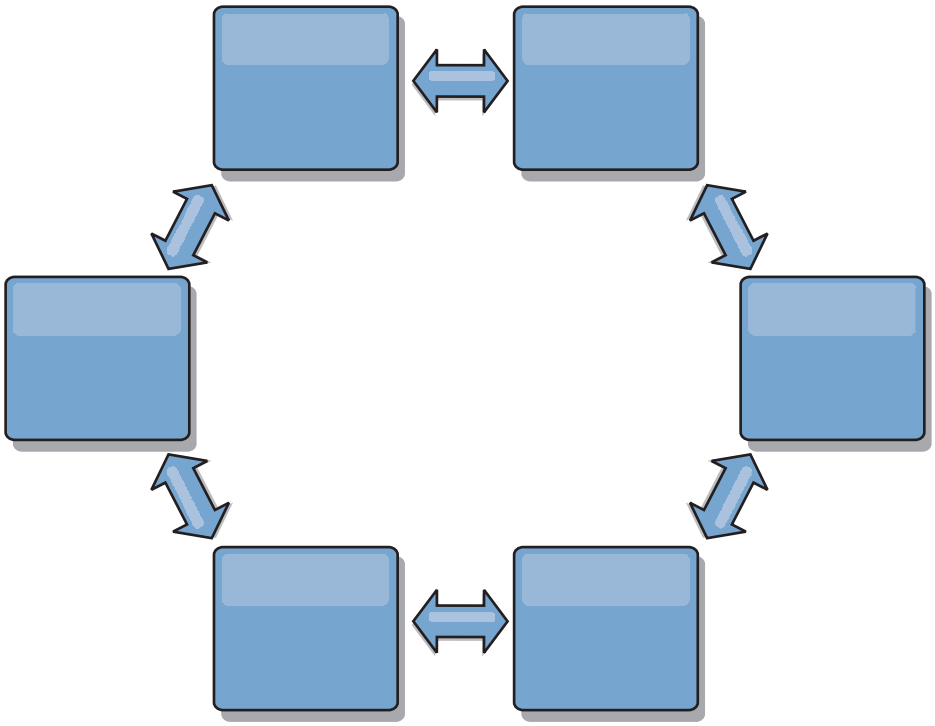
Notez que si le domaine C est défaillant, les actions suivantes se produisent :

1. Le domaine D serait orphelin jusqu'au redémarrage du domaine C.
2. Le domaine C doit se synchroniser avec le domaine B, lequel est une copie du domaine A.
3. Le domaine D utilise le domaine C pour se synchroniser avec les modifications des domaines A et B. Ces modifications se sont produites initialement lorsque le domaine D étaient orphelin (lorsque le domaine C était arrêté).

Enfin, les domaines A, B, C et D, sont de nouveau tous identiques.

Topologies en anneau

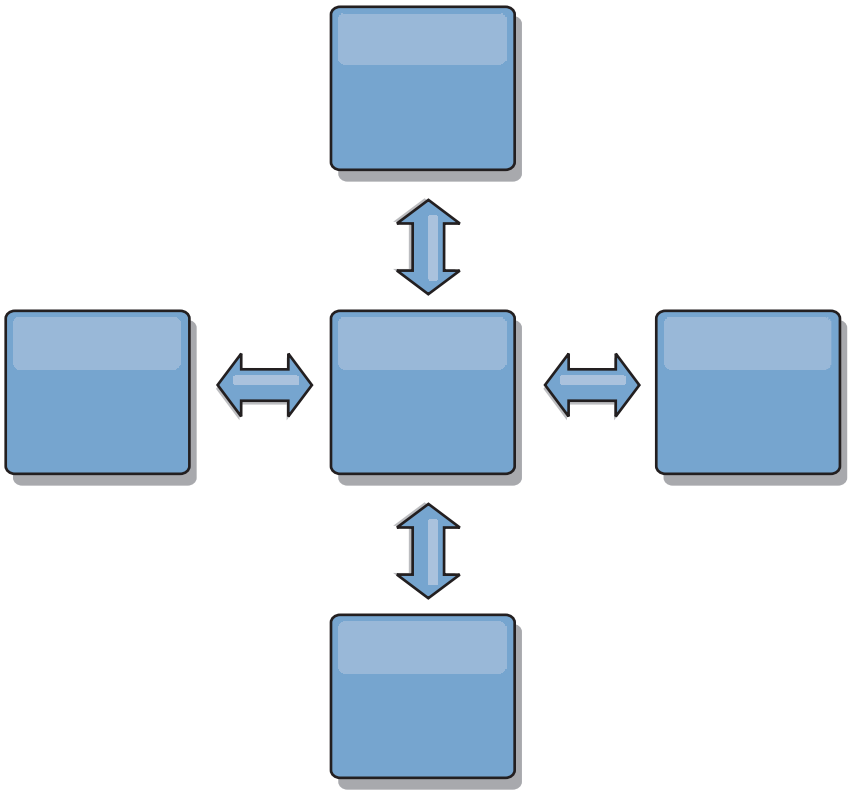
Les topologies en anneau sont un exemple de topologie encore plus résilientes. Lorsqu'un domaine de services de catalogue ou une liaison unique tombe en panne, les domaine de services de catalogue restants peuvent encore obtenir des modifications. Les domaine de services de catalogue parcourent l'anneau en s'éloignant de la défaillance. Chaque domaine de serveur de catalogue a au maximum deux liens vers d'autres domaine de services de catalogue, quelle que soit la taille de la topologie en anneau. Le délai de propagation des modifications peut être important. Les modifications d'un domaine de services de catalogue peuvent devoir traverser plusieurs liaisons pour que tous les domaine de services de catalogue aient les modifications. Une topologie linéaire a la même caractéristique.



Vous pouvez également déployer une topologie en anneau plus sophistiquée, avec un domaine de services de catalogue racine au centre de l'anneau. Le domaine de services de catalogue racine fait office de point central de réconciliation. Les autres domaine de services de catalogue font office de points distants de réconciliation pour les modifications se produisant dans le domaine de services de catalogue racine. Le domaine de services de catalogue racine peut arbitrer les modifications entre les domaine de services de catalogue. Si une topologie en anneau contient plusieurs anneaux autour d'un domaine de services de catalogue racine, le domaine ne peut pas arbitrer les modifications dans la partie interne de l'anneau. Toutefois, les résultats de l'arbitrage sont propagés dans les domaine de services de catalogue des autres anneaux.

Topologies en étoile

Avec une topologie en étoile, les modifications parcourent un domaine de services de catalogue en étoile. Etant donné que le concentrateur est le seul domaine de services de catalogue intermédiaire spécifié, les topologies en étoile ont une latence inférieure. Le domaine du concentrateur est connecté à chaque branche de domaine via une liaison. Le concentrateur distribue les modifications entre les domaine de services de catalogue. Il fait office de point de rapprochement pour les collisions. Dans un environnement soumis à une fréquence élevée de modifications, le concentrateur peut avoir besoin de s'exécuter sur plus de matériels que les branches pour rester synchronisé. WebSphere eXtreme Scale est conçu pour évoluer de manière linéaire, ce qui signifie que l'on peut, si nécessaire, étoffer le concentrateur sans difficultés. Toutefois, si le concentrateur tombe en panne, les modifications ne sont pas distribuées jusqu'à ce qu'il redémarre. Toutes les modifications sur les branches du sous-domaine de services de catalogue seront réparties après la reconnexion du concentrateur.



Vous pouvez également utiliser une stratégie avec les clients intégralement répliqués, une variante de la topologie qui utilise une paire de serveurs eXtreme Scale s'exécutant comme concentrateur. Chaque client crée une grille de données à conteneur unique autonome avec un catalogue dans la machine virtuelle Java client. Un client utilise sa grille de données pour se connecter au catalogue du concentrateur. Cette connexion provoque la synchronisation du client avec le concentrateur dès que le client obtient une connexion au concentrateur.

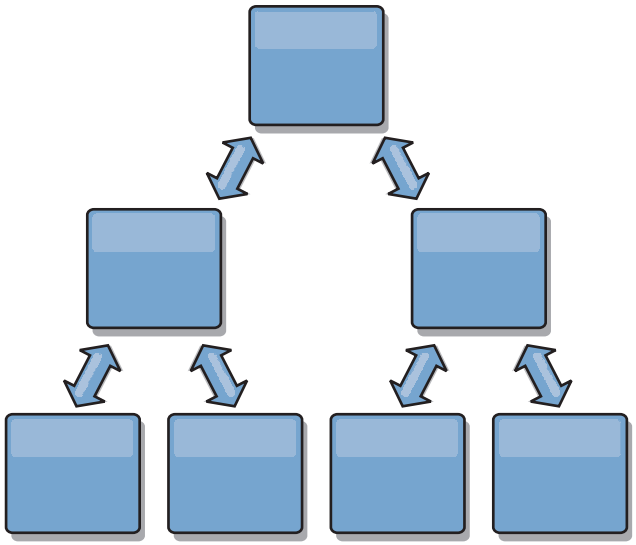
Toutes les modifications effectuées par le client sont locales pour le client et elles sont répliquées vers le concentrateur de manière asynchrone. Le concentrateur joue le rôle de domaine d'arbitrage, répartissant les modifications à tous les clients connectés. La topologie de clients intégralement répliqués fournit un cache L2 fiable pour un associateur relationnel d'objets comme OpenJPA. Les modifications sont réparties rapidement via le concentrateur entre les machines virtuelles client. Si la taille du cache peut être contenue dans le segment de mémoire disponible, la topologie est une architecture fiable pour ce style de cache L2.

Si nécessaire, utilisez plusieurs partitions pour échelonner le domaine concentrateur sur plusieurs machines virtuelles Java. Etant donné que toutes les données doivent toujours tenir sur une seule machine virtuelle Java client, plusieurs partitions augmentent la capacité du concentrateur à répartir et à arbitrer les modifications. Cependant, plusieurs partitions ne changent pas la capacité d'un domaine unique.

Topologies en arbre

Vous pouvez également utiliser un arbre dirigé acyclique. Un arbre acyclique n'a pas de cycles ou de boucles, et une configuration dirigée limite les liaisons aux parents et enfants existants uniquement. Cette configuration peut être utile pour les topologies disposant d'un grand nombre de domaine de services de catalogue, et il

n'est pas pratique d'avoir un concentrateur central connecté à chaque branche. Ce type de topologie peut également être utile lorsque vous devez ajouter des domaine de services de catalogue enfant sans mettre à jour le domaine de services de catalogue racine.



Une topologie en arbre peut toujours avoir un point central de rapprochement dans le domaine de services de catalogue racine. Le deuxième niveau peut toujours fonctionner en tant que point de rapprochement distant pour les modifications se produisant dans le domaine de services de catalogue en dessous. Le domaine de services de catalogue racine peut arbitrer les modifications entre les domaines de services de catalogue sur le deuxième niveau uniquement. Vous pouvez également utiliser des arbres n-aire ayant chacun n enfants à chaque niveau. Chaque domaine de services de catalogue se connecte à *n* liaisons.

Clients intégralement répliqués

Cette variante de la topologie implique une paire de serveurs eXtreme Scale s'exécutant comme concentrateur. Chaque client crée une grille de données à conteneur unique autonome avec un catalogue dans la machine virtuelle Java client. Un client utilise sa grille de données pour se connecter au catalogue du concentrateur, ce qui provoque la synchronisation du client avec le concentrateur dès que le client obtient une connexion au concentrateur.

Toutes les modifications effectuées par le client sont locales pour le client et elles sont répliquées vers le concentrateur de manière asynchrone. Le concentrateur joue le rôle de domaine d'arbitrage, répartissant les modifications à tous les clients connectés. La topologie de clients intégralement répliqués fournit un bon cache de niveau 2 pour un associateur relationnel d'objets comme OpenJPA. Les modifications sont réparties rapidement via le concentrateur entre les machines virtuelles client. Tant que la taille du cache peut être contenue dans l'espace de segment mémoire disponible des clients, cette topologie est une architecture tout à fait indiquée pour ce style de cache de niveau 2.

Si nécessaire, utilisez plusieurs partitions pour échelonner le domaine concentrateur sur plusieurs machines virtuelles Java. Toutes les données devant tenir sur une seule machine virtuelle Java, l'utilisation de partitions multiples augmente la capacité du concentrateur à répartir et à arbitrer les modifications, mais elle ne change pas la capacité d'un domaine unique.

Tâches associées:

Configuration de plusieurs topologies de centres de données
Avec la réplication asynchrone multimaître, vous liez un ensemble de domaines de services de catalogue. Les domaines de services de catalogue connectés sont ensuite synchronisés en utilisant la réplication via les liaisons. Vous pouvez définir les liaisons à l'aide de fichiers de propriétés, lors de l'exécution avec des programmes JMX (Java Management Extensions) ou avec les utilitaires de ligne de commande. Le groupe de liaisons actuel d'un domaine est stocké dans le service de catalogue. Vous pouvez ajouter et supprimer des liens sans redémarrer le domaine de services de catalogue qui héberge la grille de données.

«Développement d'arbitres personnalisés pour la réplication multi-maître», à la page 303
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 multimaître, les domaines de services de catalogue détectent les collisions automatiquement. Lorsqu'un domaine de services de catalogue détecte une collision, il démarre un arbitre. En général, les collisions sont résolues avec l'arbitre de collision par défaut. Mais une application peut très bien fournir un arbitre personnalisé.

Considérations de configuration pour les topologies multimaîtres

Tenez compte des points suivants lorsque vous déterminez l'opportunité et la manière d'utiliser des topologies de réplication multimaîtres.

- **Exigences de groupe de mappes**

  Les groupes de mappes doivent avoir les caractéristiques suivantes pour pouvoir répliquer les modifications dans les liaisons d'un domaine de services de catalogue :

  – Le nom ObjectGrid et le nom de groupe de mappes dans un domaine de services de catalogue doivent correspondre au nom ObjectGrid et au nom de groupe de mappes d'autres domaine de services de catalogue. Par exemple, ObjectGrid "og1" et le groupe de mappes "ms1" doivent être configurés dans les domaine de services de catalogue A et B pour pouvoir répliquer les données dans la mappe entre les domaine de services de catalogue.

  – Est une grille de données `FIXED_PARTITION`. Les grilles de données `PER_CONTAINER` ne peuvent pas être répliquées.

  – A le même nombre de partitions dans chaque domaine de services de catalogue. Le groupe de mappes peut ou peut ne pas avoir le même nombre et le même type de répliques.

  – A les mêmes types de données répliquées dans chaque domaine de services de catalogue.

  – Contient les mêmes mappes et modèles de mappes dynamiques dans chaque domaine de services de catalogue.

  – N'utilise pas le gestionnaire d'entités. Un groupe de mappes contenant une mappe d'entité n'est pas répliqué entre les domaine de services de catalogue.

  – N'utilise pas la mise en cache en écriture différée. Un groupe de mappes contenant une mappe qui est configurée avec la prise en charge de l'écriture différée n'est pas répliqué entre les domaine de services de catalogue.

  Tous les ensembles de mappes ayant les caractéristiques ci-dessus commencent à répliquer après que les domaine de services de catalogue dans la topologie ont été démarrés.

- **Chargeurs de classe avec plusieurs domaine de services de catalogue**

Les domaine de services de catalogue doivent avoir accès à toutes les classes qui sont utilisées comme clés et valeurs Toutes les dépendances doivent être reflétées dans tous les chemins d'accès aux classes des machines virtuelles Java (JVM) de conteneur de la grille de données de tous les domaines. Si un plug-in CollisionArbiter extrait la valeur d'une entrée de cache, les classes correspondant aux valeurs doivent être présentes pour le domaine qui démarre l'arbitre.

Tâches associées:

Configuration de plusieurs topologies de centres de données
Avec la réplication asynchrone multimaître, vous liez un ensemble de domaines de services de catalogue. Les domaines de services de catalogue connectés sont ensuite synchronisés en utilisant la réplication via les liaisons. Vous pouvez définir les liaisons à l'aide de fichiers de propriétés, lors de l'exécution avec des programmes JMX (Java Management Extensions) ou avec les utilitaires de ligne de commande. Le groupe de liaisons actuel d'un domaine est stocké dans le service de catalogue. Vous pouvez ajouter et supprimer des liens sans redémarrer le domaine de services de catalogue qui héberge la grille de données.

«Développement d'arbitres personnalisés pour la réplication multi-maître», à la page 303
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 multimaître, les domaines de services de catalogue détectent les collisions automatiquement. Lorsqu'un domaine de services de catalogue détecte une collision, il démarre un arbitre. En général, les collisions sont résolues avec l'arbitre de collision par défaut. Mais une application peut très bien fournir un arbitre personnalisé.

Remarques sur les chargeurs dans une topologie multimaître

Lorsque vous utilisez des chargeurs dans une topologie multimaître, vous devez envisager les problèmes éventuels de collision et de maintenance des informations de révision. La grille de données conserve les informations de révision sur les éléments de façon à ce que les collisions puissent être détectées lorsque d'autres fragments primaires dans la configuration y écrivent des entrées. Lorsque des entrées sont ajoutées à partir d'un chargeur, ces informations de révision ne sont pas incluses et l'entrée prend une nouvelle révision. Etant donné que la révision de l'entrée semble être une nouvelle insertion, une fausse collision peut se produire si un autre fragment primaire modifie également cet état ou insère les mêmes informations à partir d'un chargeur.

Les modifications de réplication appellent la méthode get sur le chargeur avec la liste des clés qui ne sont pas déjà dans la grille de données, mais qui vont être modifiées lors de la transaction de réplication. Lorsque la réplication se produit, ces entrées sont des entrées de collision. Lorsque les collisions sont arbitrées et que la révision est appliquée, une mise à jour par lots est appelée sur le chargeur pour appliquer les modifications à la base de données. Toutes les mappes qui ont été modifiées dans la fenêtre de révision sont mises à jour dans la même transaction.

L'énigme de préchargement

Supposons une topologie avec les deux centres de données A et B qui ont des bases de données indépendantes, mais seul le centre de données A a une grille active. Lorsque vous établissez une liaison entre les centres de données pour une configuration multimaître, les grilles de données dans le centre de données A commencent à envoyer les données aux nouvelles grilles dans le centre de données B, ce qui crée une collision avec chaque entrée. Un autre problème est l'existence de données dans la base de données du centre de données B, mais qui ne figurent

pas dans la base de données du centre de données A. Ces lignes ne sont pas remplies et arbitrées, ce qui génère des incohérences qui ne sont pas résolues.

Solution de l'énigme de préchargement

Etant donné que les données qui se trouvent uniquement dans la base de données ne peuvent pas comporter des révisions, vous devez toujours précharger complètement la grille de données à partir de la base de données locale pour établir la liaison multimaître. Ensuite, les deux grilles de données peuvent réviser et arbitrer les données, pour atteindre finalement un état cohérent.

L'énigme du cache partiel

Avec un cache partiel, la première application tente de trouver des données dans la grille de données. Si les données ne sont pas dans la grille de données, elles sont recherchées dans la base de données à l'aide du chargeur. Les entrées sont supprimées de la grille de données régulièrement pour maintenir une mémoire cache de petite taille.

Ce type de mémoire cache peut être problématique dans un scénario de configuration multimaître, car les entrées dans la grille de données ont des métadonnées de révision qui permettent de détecter quand des collisions se produisent et de déterminer qui a effectué les modifications. Lorsque des liaisons entre les centres de données ne fonctionnent pas, un centre de données peut mettre à jour une entrée et ensuite éventuellement mettre à jour la base de données et invalider l'entrée dans la grille de données. Lorsque la liaison est rétablie, les centres de données tentent de synchroniser les révisions les unes par rapport aux autres. Toutefois, étant donné que la base de données a été mise à jour et que l'entrée de la grille de données a été invalidée, la modification est perdue du point de vue du centre de données qui s'est arrêté. En conséquence, les deux côtés de la grille de données sont désynchronisés et ne sont pas cohérents.

Solution de l'énigme de cache partiel

Topologie en étoile :

Vous pouvez exécuter le chargeur uniquement dans la topologie en étoile pour maintenir la cohérence des données lors de l'extension de la grille de données. Toutefois, si vous envisagez ce déploiement, notez que les chargeurs peuvent permettre à la grille de données d'être partiellement chargée, ce qui implique qu'un expulseur a été configuré. Si les rayons de la configuration sont des caches partiels, les échecs en mémoire cache n'ont aucun moyen d'extraire des données de la base de données. En raison de cette restriction, vous devez utiliser une topologie de cache complètement remplie avec une configuration en étoile.

Invalidations et expulsion

L'invalidation crée des incohérences entre la grille de données et la base de données. Les données peuvent être supprimées de la grille de données, à l'aide d'un programme ou par l'expulsion. Lorsque vous développez votre application, sachez que le traitement des révisions ne réplique pas les modifications invalidées, ce qui provoque des incohérences entre les fragments primaires.

Les événements d'invalidation ne sont pas des modifications de l'état du cache et n'entraînent pas de réplication. Tous les expulseurs configurés s'exécutent indépendamment des autres expulseurs dans la configuration. Par exemple, vous

pouvez avoir un expulseur configuré pour un seuil de mémoire dans un domaine de services de catalogue, mais un type d'expulseur différent moins agressif dans l'autre domaine de services de catalogue lié. Lorsque des entrées de grille de données sont supprimées en raison de la règle de seuil de mémoire, les entrées dans l'autre domaine de services de catalogue ne sont pas affectées.

Mises à jour de la base de données et invalidation de la grille de données

Des problèmes se produisent lorsque vous mettez à jour la base de données directement en arrière-plan lors de l'appel de l'invalidation dans la grille de données pour les entrées mises à jour dans une configuration multimaître. Ce problème se produit, car la grille de données ne peut pas répliquer la modifications dans les autres fragments primaires jusqu'à ce qu'un accès de cache transfère l'entrée vers la grille de données.

Plusieurs programmes d'écriture dans une seule base de données logique

Lorsque vous utilisez une seule base de données avec plusieurs fragments primaires qui sont connectés par l'intermédiaire d'un chargeur, des conflits transactionnels se produisent. Votre implémentation de chargeur doit gérer ces types de scénarios.

Mise en miroir des données à l'aide de la réplication multimaître

Vous pouvez configurer des bases de données indépendantes qui sont connectées à des domaine de services de catalogue indépendants. Dans cette configuration, le chargeur peut envoyer les modifications d'un centre de données vers un autre.

Tâches associées:

Configuration de plusieurs topologies de centres de données
Avec la réplication asynchrone multimaître, vous liez un ensemble de domaines de services de catalogue. Les domaines de services de catalogue connectés sont ensuite synchronisés en utilisant la réplication via les liaisons. Vous pouvez définir les liaisons à l'aide de fichiers de propriétés, lors de l'exécution avec des programmes JMX (Java Management Extensions) ou avec les utilitaires de ligne de commande. Le groupe de liaisons actuel d'un domaine est stocké dans le service de catalogue. Vous pouvez ajouter et supprimer des liens sans redémarrer le domaine de services de catalogue qui héberge la grille de données.

«Développement d'arbitres personnalisés pour la réplication multi-maître», à la page 303
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 multimaître, les domaines de services de catalogue détectent les collisions automatiquement. Lorsqu'un domaine de services de catalogue détecte une collision, il démarre un arbitre. En général, les collisions sont résolues avec l'arbitre de collision par défaut. Mais une application peut très bien fournir un arbitre personnalisé.

Considérations de conception pour la réplication multimaître

Lors de l'implémentation de la réplication multimaître, vous devez tenir compte de divers éléments dans votre conception, tels que l'arbitrage, les liaisons et les performances.

Points concernant l'arbitrage à prendre en considération dans la conception des topologies

Des collisions entre des modifications peuvent se produire s'il est possible à des enregistrements identiques d'être modifiés simultanément en deux endroits différents. Configurez chaque domaine de services de catalogue pour que les domaines aient le même nombre de processeurs, la même quantité de mémoire et le même nombre de ressources réseau. Vous remarquerez sans doute que des domaine de services de catalogue d'exécution gérant les collisions de modifications (arbitrage) utilisent plus de ressources que d'autres domaine de services de catalogue. Les collisions sont détectées de manière automatique. Elles sont traitées avec l'un des deux mécanismes suivants :

- **Arbitre par défaut** : le protocole par défaut doit utiliser les modifications du domaine de services de catalogue occupant la position la moins basse alphabétiquement. Par exemple, si les domaine de services de catalogue A et B génèrent un conflit pour un enregistrement, la modification du domaine de services de catalogue B est ignorée. Le domaine de services de catalogue A conserve sa version et l'enregistrement dans le domaine de services de catalogue B est modifié pour qu'il corresponde à l'enregistrement du domaine de services de catalogue A. Ce comportement s'applique également aux applications où les utilisateurs ou les sessions sont normalement liés ou ont une affinité à l'une des grilles de données.
- **Arbitre personnalisé** : les applications peuvent fournir un arbitre personnalisé. Lorsqu'un domaine de services de catalogue détecte une collision, il démarre l'arbitre. Pour plus d'informations sur le développement d'un arbitre personnalisé utile, voir «Développement d'arbitres personnalisés pour la réplication multi-maître», à la page 303.

Pour les topologies dans lesquelles les collisions sont possibles, songez à implémenter une topologie en étoile ou en arbre. Les deux topologies sont propices à éviter les collisions constantes, ce qui peut se produire dans les scénarios suivants :

1. Plusieurs domaine de services de catalogue sont affectés par une collision.
2. Chaque domaine de services de catalogue gère la collision en local, ce qui produit des révisions.
3. Les révisions entrent en collision, d'où des révisions de révisions.

Pour éviter les collisions, choisissez un domaine de services de catalogue spécifique, appelé *domaine de services de catalogue d'arbitrage* comme arbitre des collisions d'un sous-ensemble de domaine de services de catalogue. Par exemple, une topologie en étoile pourra utiliser le concentrateur comme gestionnaire de collisions. Le gestionnaire de collisions ignore toutes les collisions qui sont détectées par les sous-domaine de services de catalogue. Le domaine de services de catalogue du concentrateur crée des révisions, empêchant les révisions de collisions inattendues. Le domaine de services de catalogue qui est affecté à la gestion des collisions doit se lier à tous les domaines dont il est chargé de traiter les collisions. Dans une topologie en arbre, tous les domaines parent internes traitent les collisions pour leurs enfants immédiats. En revanche, si vous utilisez une topologie en anneau, vous ne pouvez pas désigner un domaine de services de catalogue dans le fichier comme arbitre.

Le tableau qui suit récapitule les approches en matière d'arbitrage qui sont les plus compatibles avec les diverses topologies.

Tableau 1. Approches en matière d'arbitrage. Ce tableau énonce si l'arbitrage entre applications est compatible avec les diverses topologies.

Topologie	Arbitrage d'application	Notes
Ligne de deux domaine de services de catalogue	Oui	Choisissez un domaine de services de catalogue comme arbitre.
Ligne de trois domaine de services de catalogue	Oui	Le domaine de services de catalogue du milieu doit être l'arbitre. Assimilez ce domaine de services de catalogue au concentrateur dans une topologie en étoile simple.
Ligne de plus de trois domaine de services de catalogue	Non	L'arbitrage d'application n'est pas pris en charge.
Concentrateur avec n "rayons"	Oui	Le concentrateur avec des liens vers toutes les branches doit être le domaine de services de catalogue d'arbitrage.
Anneau de N domaine de services de catalogue	Non	L'arbitrage d'application n'est pas pris en charge.
Arbre dirigé acyclique (arbre n-aire)	Oui	Tous les noeuds racine doivent évaluer leurs descendants directs uniquement.

Points concernant les liens à prendre en considération dans la conception des topologies

Dans l'idéal, une topologie comprend le minimum de liens tout en optimisant les compromis entre les temps d'attente des modifications, la tolérance aux pannes et les caractéristiques de performances.

- **Temps d'attente des modifications**

  Le temps d'attente de modification est déterminé par le nombre de domaine de services de catalogue intermédiaires par lequel un changement doit passer avant d'arriver à un domaine de services de catalogue spécifique.

  Une topologie a le meilleur temps d'attente lorsqu'elle élimine les domaine de services de catalogue intermédiaires en liant chacun des domaine de services de catalogue à chacun des autres domaine de services de catalogue. Toutefois, un domaine de services de catalogue doit effectuer la réplication par rapport à son nombre de liens. Pour les topologies de grande taille, le nombre de liens à définir peut entraîner une charge administrative.

  La vitesse à laquelle une modification est copiée vers les autres domaine de services de catalogue dépend de facteurs supplémentaires, tels que :

  – Bande passante du processeur et du réseau dans le domaine de services de catalogue source
  – Nombre de domaine de services de catalogue intermédiaire et de liens entre la source et la cible du domaine de services de catalogue source et cible
  – Ressources en processeur et en réseau disponibles pour les domaine de services de catalogue source, cible et intermédiaires

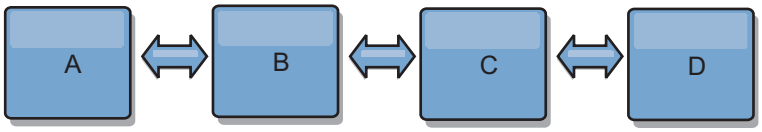
- **Tolérance aux pannes**

  La tolérance aux pannes est déterminée par le nombre de chemins existant entre deux domaine de services de catalogue pour la réplication des modifications.

  Si vous ne disposez que d'un seul lien entre une paire de domaine de services de catalogue, une défaillance de lien empêche la propagation des modifications.

De même, les modifications ne sont pas propagées entre les domaine de services de catalogue si un incident de liaison se produit sur les domaines intermédiaires. Votre topologie pourrait avoir un lien unique d'un domaine de services de catalogue vers un autre de sorte que le lien passe par des domaines intermédiaires. Dans ce cas, les modifications ne sont pas propagées si l'un des domaine de services de catalogue intermédiaires est défaillant.

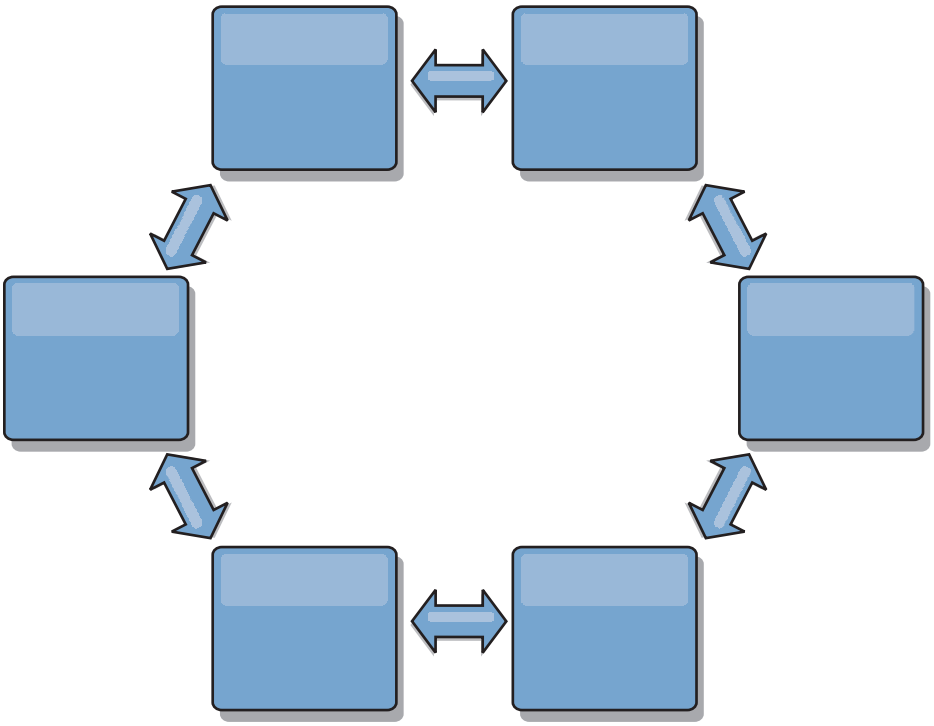
Supposons la topologie linéaire à quatre domaine de services de catalogue A, B, C et D :



Si l'une de ces conditions existe, le domaine D ne voit pas les modification de A :

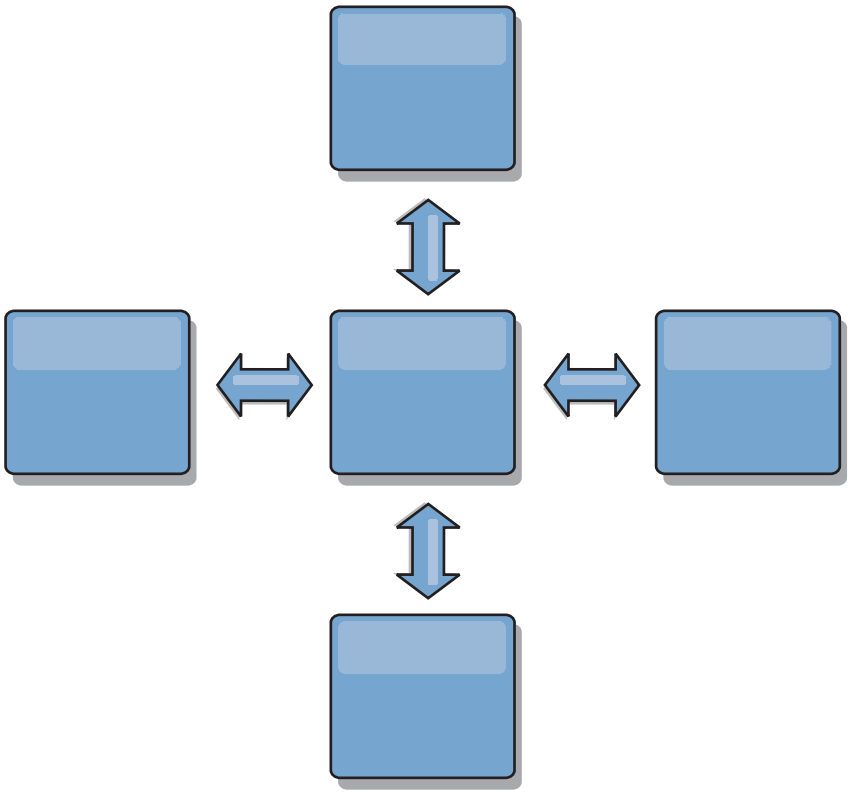
- Le domaine A est actif et B est arrêté.
- Les domaines A et B sont actifs et C est arrêté.
- Le lien entre A et B ne fonctionne pas.
- Le lien entre B et C ne fonctionne pas.
- Le lien entre C et D est arrêté.

En revanche, avec une topologie en anneau, chaque domaine de services de catalogue peut recevoir les modifications dans un sens ou dans l'autre.



Par exemple, si un service de catalogue donné de votre topologie en anneau est arrêté, les deux domaines contigus peuvent toujours extraire les modifications directement de l'autre.

Toutes les modifications sont propagées via le concentrateur. Par conséquent, contrairement aux topologies linéaires et en anneau, la conception en étoile peut tomber en panne si le concentrateur est défaillant.



Un domaine de services de catalogue unique est résilient à un certain degré de perte de service. Cependant, les incidents les plus importants, tels que les indisponibilités de réseau étendu ou les pertes de liaisons entre les centres de données physiques peuvent perturber les domaine de services de catalogue.

- **Liaison et performances**

  Le nombre de liaisons définies sur un domaine le service de catalogue affecte les performances. Un plus grand nombre de liaisons utilisent davantage de ressources et les performances de réplication peuvent baisser. La possibilité d'extraire les modifications pour un domaine A via d'autres domaines empêche le domaine A de répliquer ses transactions partout. La charge de la répartition des modifications dans un domaine est limitée par le nombre de liaisons qu'il utilise et non pas par le nombre de domaines dans la topologie. Cette propriété est synonyme d'évolutivité et les domaines de la topologie peuvent partager la charge de la répartition des modifications.

  Un domaine de services de catalogue peut extraire les modifications indirectement via d'autres domaine de services de catalogue. Supposons une topologie linéaire avec cinq domaine de services de catalogue.

  ```
  A <=> B <=> C <=> D <=> E
  ```

  – A extrait les modifications de B, C, D, et E via B
  – B extrait les modifications directement de A et de C et les modifications de D et de E via C
  – C extrait les modifications directement de B et de D et les modifications de A via B et de E via D
  – D extrait les modifications directement de C et de E et les modifications de A et de B via C
  – E extrait les modifications directement de D et les modifications de A, B et C via D

La charge de la répartition dans les domaine de services de catalogue A et E est la plus faible, car ils ont chacun une seule liaison à un domaine de services de catalogue unique. Les domaines B, C et D ont chacun une liaison avec deux domaines. Par conséquent, la charge de la répartition dans les domaines B, C, et D est le double de celle des domaines A et E. La charge de travail dépend du nombre de liaisons dans chaque domaine et non pas du nombre total de domaines dans la topologie. Par conséquent, la répartition de charge décrite demeurerait constante, même si la ligne contenait 1 000 domaines.

Considérations relatives aux performances de réplication multimaître

Tenez compte des limitations suivantes lorsque vous utilisez des topologies de réplication multimaître :

- **Optimisation de la répartition des modifications**, comme expliquée dans la section précédente.
- **Performances des liens de réplication** WebSphere eXtreme Scale crée un seul socket TCP/IP entre n'importe quelle paire de machines virtuelles Java. Tout le trafic entre les machines virtuelles Java passe par le socket unique, y compris le trafic de la réplication multimaître. Les domaine de services de catalogue sont hébergés dans au moins *n* machines virtuelles Java pour fournir au minimum *n* liaisons TCP aux domaine de services homologues. Ainsi, les domaine de services de catalogue avec un plus grand nombre de conteneurs offrent de meilleures performances de réplication. Un plus grand nombre de conteneurs requiert davantage de processeurs et de ressources réseau.
- **Le support de l'optimisation de la fenêtre dynamique TCP et RFC 1323** RFC 1323 à chaque extrémité d'une liaison renvoie plus de données pour un aller-retour. Ce support augmente le débit en développant la capacité de la fenêtre d'un facteur d'environ 16 000.

  Notez que les sockets TCP utilisent un mécanisme de fenêtre dynamique pour contrôler le flux des données en vrac. Ce mécanisme limite généralement le socket à 64 Ko pour un intervalle d'aller-retour. Si l'intervalle aller-retour est de 100 ms, la bande passante est limitée à 640 Ko/s sans optimisation supplémentaire. L'utilisation intégrale de la bande passante disponible sur un lien peut nécessiter une optimisation qui est spécifique au système d'exploitation. La plupart des systèmes d'exploitation comportent des paramètres d'optimisation, y compris des options RFC 1323, permettant d'améliorer le débit sur les liaisons à forte latence.

  Plusieurs facteurs peuvent affecter les performances de la réplication :

  – Vitesse d'extraction des modifications par eXtreme Scale.
  – Vitesse à laquelle eXtreme Scale peut traiter les demandes de réplication d'extraction.
  – Capacité de la fenêtre dynamique.
  – Avec l'optimisation de la mémoire tampon réseau aux deux extrémités d'une liaison, eXtreme Scale extrait les modification sur le socket de manière efficace.
- **Sérialisation des objets** Toutes les données doivent être sérialisables. Si un domaine de services de catalogue n'utilise pas `COPY_TO_BYTES`, il doit utiliser la sérialisation Java ou ObjectTransformers pour optimiser les performances de sérialisation.
- Par défaut **la compression** WebSphere eXtreme Scale compresse toutes les données envoyées entre les domaines de services de catalogue. Vous ne pouvez désactiver actuellement la compression.

- **Optimisation de la mémoire** L'utilisation de la mémoire pour une topologie de réplication multimaître est largement indépendante du nombre de domaine de services de catalogue dans la topologie.

  La réplication multimaître ajoute un temps de traitement fixe par entrée de mappe pour la gestion des versions. Chaque conteneur suit également une quantité fixe de données pour chaque domaine de services de catalogue dans la topologie. Une topologie à deux domaines de services de catalogue utilise approximativement la même quantité de mémoire qu'une topologie à 50 domaine de services de catalogue. WebSphere eXtreme Scale n'utilise pas de journaux de relecture ou de files d'attente similaires dans son implémentation. Ainsi, aucune structure de récupération n'est prête si la liaison de réplication n'est pas disponible pendant un certain temps et redémarre ensuite.

Tâches associées:

Configuration de plusieurs topologies de centres de données
Avec la réplication asynchrone multimaître, vous liez un ensemble de domaines de services de catalogue. Les domaines de services de catalogue connectés sont ensuite synchronisés en utilisant la réplication via les liaisons. Vous pouvez définir les liaisons à l'aide de fichiers de propriétés, lors de l'exécution avec des programmes JMX (Java Management Extensions) ou avec les utilitaires de ligne de commande. Le groupe de liaisons actuel d'un domaine est stocké dans le service de catalogue. Vous pouvez ajouter et supprimer des liens sans redémarrer le domaine de services de catalogue qui héberge la grille de données.

«Développement d'arbitres personnalisés pour la réplication multi-maître», à la page 303
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 multimaître, les domaines de services de catalogue détectent les collisions automatiquement. Lorsqu'un domaine de services de catalogue détecte une collision, il démarre un arbitre. En général, les collisions sont résolues avec l'arbitre de collision par défaut. Mais une application peut très bien fournir un arbitre personnalisé.

Planification pour développer des applications WebSphere eXtreme Scale

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

Présentation des API

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 réplique 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.

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.

Présentation des plug-ins

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-ins suivants sont disponibles pour une instance ObjectGrid. Si le plug-in est côté serveur uniquement, les plug-ins sont supprimés sur les instances ObjectGrid et BackingMap client. Ces instances sont des instances serveur uniquement.

- **TransactionCallback** : un plug-in TransactionCallback offre des événements de cycle de vie de transaction. Si le plug-in TransactionCallback est la classe d'implémentation intégrée JPATxCallback (com.ibm.websphere.objectgrid.jpa.JPATxCallback), le plug-in est côté serveur uniquement. Toutefois, les sous-classes de la classe JPATxCallback ne sont pas côté serveur uniquement.
- **ObjectGridEventListener** : un plug-in ObjectGridEventListener offre des événements de cycle de vie ObjectGrid pour l'ObjectGrid, les fragments et les transactions.
- **ObjectGridLifecycleListener** : un plug-in ObjectGridLifecycleListener offre des événements de cycle de vie ObjectGrid pour l'instance ObjectGrid. Le plug-in ObjectGridLifecycleListener peut être utilisé comme une interface mixte intégrée facultative pour tous les autres plug-ins ObjectGrid.
- **ObjectGridPlugin** : un plug-in ObjectGridPlugin est une interface intégrée mixte qui fournit des événements de gestion de cycle de vie étendue pour tous les autres plug-ins ObjectGrid.
- **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)

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.
- **BackingMapLifecycleListener** : un plug-in BackingMapLifecycleListener fournit des événements de cycle de vie de BackingMap pour l'instance BackingMap. Le plug-in BackingMapLifecycleListener peut être utilisé comme une interface mixte intégrée facultative pour tous les autres plug-ins BackingMap.
- **BackingMapPlugin** : un plug-in BackingMapPlugin est une interface intégrée mixte facultative qui fournit des événements de gestion de cycle de vie étendue pour tous les autres plug-ins ObjectGrid.
- **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** : un plug-in Loader dans une mappe ObjectGrid fait office de cache pour les données généralement conservées dans un magasin de persistance sur le même système ou un autre système. (côté serveur uniquement)

Présentation des services de données REST

Le service de données REST de WebSphere eXtreme Scale est un service HTTP Java qui est compatible avec Microsoft WCF Data Services (ex-ADO.NET Data Services) et qui implémente Open Data Protocol (OData). Microsoft WCF Data Services est compatible avec cette spécification lorsqu'on utilise Visual Studio 2008 SP1 et .NET Framework 3.5 SP1.

Compatibilités requises

Le service de données REST autorise n'importe quel client HTTP à accéder à une grille de données. 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. L'illustration montre l'interaction de WCF Data Services avec les clients et les bases de données.

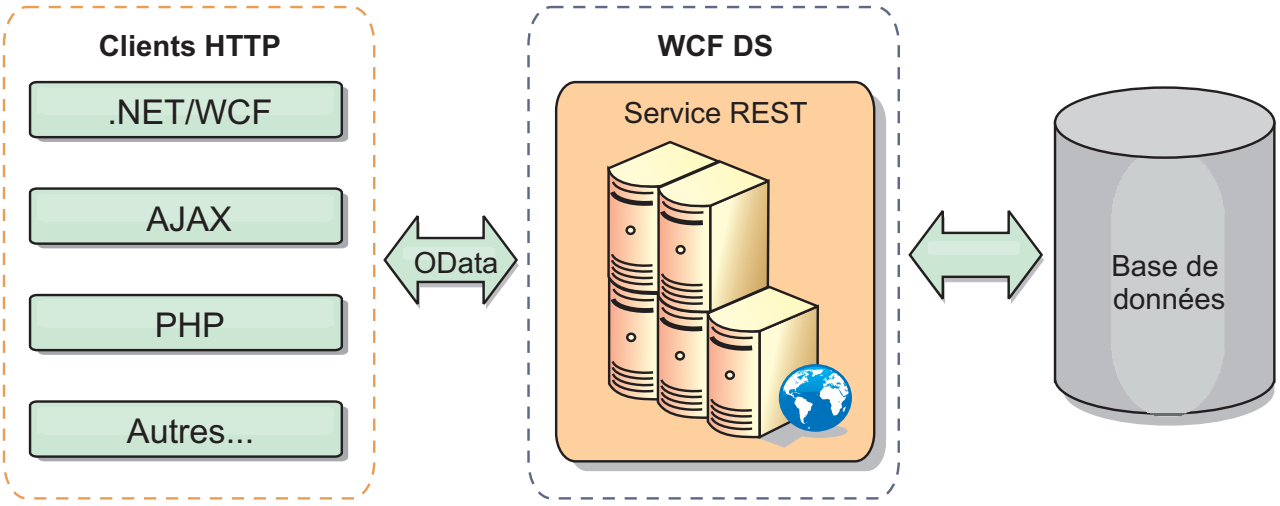


Figure 22. Microsoft WCF Data Services

WebSphere eXtreme Scale inclut un ensemble d'API riche en fonctionnalités pour les clients Java. Comme le montre l'illustration ci-dessous, le service de données REST est une passerelle entre les clients HTTP et la grille de données WebSphere eXtreme Scale, la communication avec la grille s'effectuant via un client WebSphere eXtreme Scale. Le service de données REST est un servlet Java qui permet des déploiements flexibles pour des plateformes Java Platform, Enterprise Edition (JEE) comme WebSphere Application Server. Le service de données REST communique avec la grille de données WebSphere eXtreme Scale en utilisant les API WebSphere eXtreme Scale Java. Il autorise le recours aux clients WCF Data Services ou à tout autre client capable de communiquer avec HTTP et XML.

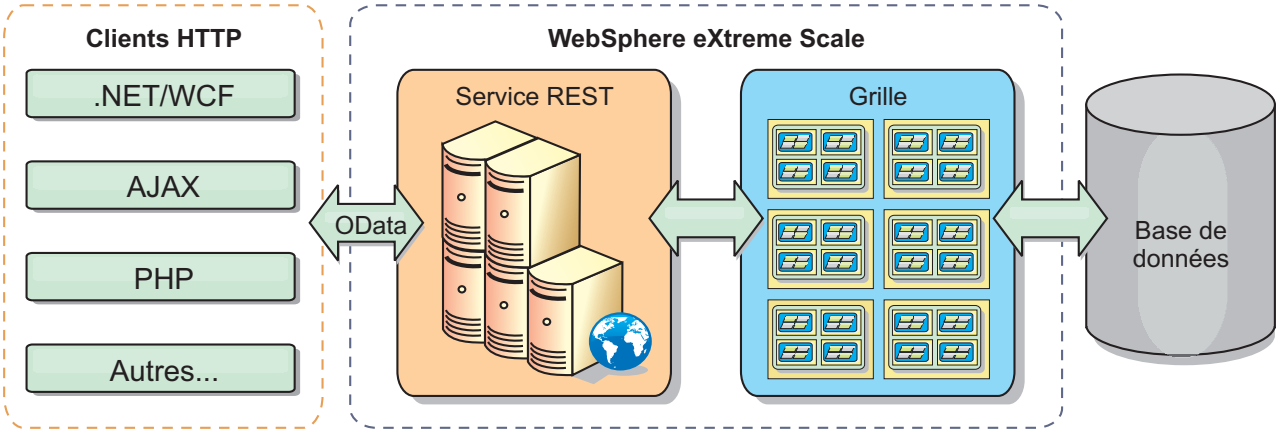


Figure 23. Service de données REST de WebSphere eXtreme Scale

Pour en savoir davantage sur WCF Data Services, reportez-vous à Configuration des services de données REST ou suivez les liens ci-dessous.

- Microsoft WCF Data Services Developer Center
- Présentation d'ADO.NET Data Services sur MSDN
- Livre blanc : Using ADO.NET Data Services
- Atom Publish Protocol : Data Services URI and Payload Extensions
- Conceptual Schema Definition File Format
- Entity Data Model for Data Services Packaging Format

- Open Data Protocol
- Open Data Protocol FAQ

Fonctionnalités

Cette version du service de données REST eXtreme Scale prend en charge les fonctionnalités suivantes :

- modélisation automatique des entités de l'API EntityManager d'eXtreme Scale en tant qu'entités WCF Data Services, ce qui inclut les prises en charge suivantes :
  - conversion du type de données Java en type Entity Data Model
  - prise en charge de l'association d'entités
  - prise en charge de la racine de schéma et de l'association de clés, obligatoire pour les grilles de données partitionnées

  Pour plus d'informations, voir Modèle d'entité.
- format XML Atom Publish Protocol (AtomPub ou APP) et format JavaScript Object Notation (JSON) de contenu des données
- opérations CRUD (Create, Read, Update et Delete) à l'aide des méthodes respectives de demandes HTTP : POST, GET, PUT et DELETE. En plus, l'extension Microsoft MERGE est prise en charge
- requêtes simples à l'aide de filtres
- extraction par lot et demandes d'ensembles de modifications
- support de grille de données partitionnées pour la haute disponibilité
- interopérabilité avec les clients API EntityManager d'eXtreme Scale
- prise en charge des serveurs Web JEE standard
- contrôle d'accès simultanés
- autorisation et authentification des utilisateurs entre le service de données REST et la grille de données eXtreme Scale

Problèmes connus et limitations

- Les demandes placées en tunnel ne sont pas prises en charge.

Tâches associées:

Configuration des services de données REST
Vous pouvez utiliser le service de données RESTWebSphere eXtreme Scale WebSphere Application Server, WebSphere Application Server Community Edition et Apache Tomcat.

«Accès aux données avec le service de données REST», à la page 268
Developpez des applications qui effectuent des opérations à l'aide des protocoles de services de données REST.

Référence associée:

«Concurrence optimiste dans le service de données REST», à la page 273
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.

«Protocoles de demande du service de données REST», à la page 274
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.

«Extraction de demandes avec le service de données REST», à la page 275
Une demande trieveEntity 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 sous la forme d'un document de flux Atom, qui est une relation to-many, ou un document Atom Entry qui est une relation to-one.

«Extraction d'éléments autres que des entités à l'aide des services de données REST», à la page 282
Le service de données REST permet d'extraire bien plus que des entités et notamment des collections d'entités, des propriétés, etc.

«Demandes d'insertion avec les services de données REST», à la page 288
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.

«Demandes de mise à jour avec les services de données REST», à la page 292
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.

«Suppression de demandes avec les services de données REST», à la page 296
Le service de données REST de WebSphere eXtreme Scale peut supprimer des entités, des valeurs de propriété et des liens.

Présentation de l'infrastructure Spring

Spring est une infrastructure de développement d'applications Java. WebSphere eXtreme Scale fournit le support permettant à Spring de gérer les transactions et de configurer les clients et serveurs constituant votre grille de données en mémoire déployée.

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 peut utiliser des implémentations différentes. WebSphere eXtreme Scale fournit une intégration du gestionnaire de transactions permettant à Spring de gérer les cycles de vie des transactions ObjectGrid. Voir les informations sur les transactions natives dans le *Guide de programmation* pour plus d'informations.

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.

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 pour les beans avec un ID ou des ID correspondant à cette définition de bean dans un même fragment amènent le conteneur Spring à retourner une instance de bean spécifique.

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

stocke son état de session dans une session HTTP par défaut. Si une application Web utilise eXtreme Scale pour la gestion de session, Spring automatiquement stocke l'état avec eXtreme Scale. e plus, la tolérance aux pannes est activée de la même manière que la session.

Consultez les informations de gestion de session HTTP dans le *Présentation du produit* pour plus d'informations.

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 Java EE qui s'exécute dans un WebSphere Application Server Network Deployment étendu WebSphere Extended Deployment, placez le fichier `spring.jar` et ses fichiers associés dans les modules EAR (Enterprise Archive). Vous devez également placer le fichier `ogspring.jar` au même emplacement.

Tâches associées:

«Développement d'applications avec l'infrastructure Spring», à la page 414
Apprenez comment intégrer vos applications eXtreme Scale à Spring.

«Démarrage d'un serveur de conteneur avec Spring», à la page 423
Vous pouvez démarrer un serveur de conteneur à l'aide de beans d'extension gérés Spring et la prise en charge d'espace de nom.

«Gestion des transactions avec Spring», à la page 416
est une infrastructure de développement d'applications Java communément utilisée. WebSphere eXtreme Scale permet à Spring de gérer les transactions eXtreme Scale et de configurer les clients et les serveurs eXtreme Scale.

Référence associée:

«Beans d'extension gérés par Spring», à la page 419
Vous pouvez déclarer des objets POJO (plain old Java objects) à utiliser comme point d'extension dans le fichier `objectgrid.xml`. 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. WebSphere eXtreme Scale peut désormais déléguer à Spring le rôle de fabrique de beans pour obtenir des instances de ces objets plug-in.

Fichier XML du descripteur de Spring
Utilisez un fichier XML de descripteur de Spring pour configurer et intégrer eXtreme Scale à Spring.

Fichier `objectgrid.xsd` de Spring
Utilisez le fichier `objectgrid.xsd` de Spring pour intégrer eXtreme Scale à Spring afin de gérer les transactions eXtreme Scale et configurer les clients et serveurs.

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.

Notamment, les processus client et conteneur eXtreme Scale doivent inclure les classes ou fichiers JAR dans le chemin d'accès aux classes lors du démarrage du processus. Lorsque vous concevez une application pour l'utiliser avec eXtreme Scale, séparez toute la logique métier des objets de données persistants.

Voir Chargement de classe dans le centre de documentation WebSphere Application Server pour plus d'informations.

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

Gestion des relations

Les langages orientés objet comme Java et les bases de données relationnelles prennent en charge les relations et les associations. Les relations diminuent la quantité d'espace de stockage utilisé grâce à l'utilisation de référence d'objet ou de clés externes.

Lorsque vous utilisez des relations dans une grille de données, les données doivent être organisées dans un arbre contraint. Un type de racine doit exister dans l'arborescence et tous les enfants ne doivent être associés qu'à une seule racine. Exemple : un département pourra avoir de nombreux salariés et un salarié pourra

avoir de nombreux projets. Mais un projet ne pourra avoir de nombreux salariés qui appartiennent à des départements différents. Une fois qu'une racine est définie, tous les accès à cet objet root et à ses descendants sont gérés via la racine. WebSphere eXtreme Scale utilise le code de hachage de la clé de l'objet racine pour choisir une partition. Par exemple :

```
partition = (hashCode MOD numPartitions).
```

Lorsque toutes les données d'une relation sont liées à une seule instance d'objet, l'instance, l'arborescence peut être située en totalité dans la même partition et il suffit d'une transaction pour y accéder de manière très efficace. Si les données embrassent plusieurs relations, plusieurs partitions seront nécessaires, ce qui implique des appels distants supplémentaires, avec le risque de goulots d'étranglement pour les performances.

Données de référence

Certaines relations incluent des données de recherche ou de référence, telles que `CountryName`. Avec des données de recherche ou de référence, les données doivent exister dans chaque partition. Les données sont accessibles à n'importe quelle clé racine et le même résultat est renvoyé. Les données de référence de ce type ne doivent être utilisées que dans les cas où les données sont relativement statiques. La mise à jour de ces données peut être coûteuse, car les données doivent être mis à jour dans chaque partition. L'API DataGrid est une technique usuellement employée pour conserver à jour les données de référence.

Coûts et avantages de la normalisation

La normalisation des données en utilisant des relations permet de réduire la quantité de mémoire utilisée par la grille de données, car la réplication des données diminue. Mais, en règle générale, plus l'on ajoute de données relationnelles et moindre est leur extensibilité. En effet, lorsque les données sont regroupées, la maintenance de leurs relations devient plus onéreuse et il devient difficile de leur conserver des tailles gérables. Comme la grille partitionne les données en fonction de la clé de la racine de l'arborescence, la taille de celle-ci n'est pas prise en compte. Par conséquent, si vous avez un grand nombre de relations pour une instance d'arborescence, la grille de données peuvent devenir déséquilibré, provoquant une partition détenant davantage de données que les autres.

Lorsque les données sont dénormalisées ou mises à plat, les données qui seraient normalement partagées entre deux objets sont dupliquées et chaque table peut être partitionnée indépendamment pour améliorer l'équilibre de la grille de données. Bien que cela augmente la quantité de mémoire utilisée, cela permet à l'application de se mettre à l'échelle puisqu'on est sûr que l'accès à une seule ligne de données donne accès à toutes les données nécessaires. C'est parfait pour les grilles qui sont essentiellement en lecture où la maintenance des données devient plus onéreuse.

Pour plus d'informations, voir Classifying XTP systems and scaling.

Gérer les relations à l'aide des API d'accès aux données

ObjectMap est l'API la plus rapide, la plus flexible et la plus granulaire de toutes les API d'accès aux données, car elle fournit une approche transactionnelle à base de sessions pour l'accès aux données présentes dans la grille des mappes. L'API

ObjectMap permet aux clients d'utiliser des opérations communes CRUD (create, read, update et delete) pour gérer des paires clé-valeur d'objets dans la grille de données répartie.

Lorsqu'on utilise l'API ObjectMap, les relations entre les objets doivent être exprimées par l'incorporation dans l'objet parent de la clé externe de toutes les relations.

Exemple :

```
public class Department {
 Collection<String> employeeIds;
}
```

L'API EntityManager simplifie la gestion des relations en extrayant les données persistantes des objets, y compris les clés externes. Lorsque l'objet est ultérieurement récupéré dans la grille de données, le graphique des relations est reconstruit, comme dans l'exemple suivant.

```
@Entity
public class Department {
 Collection<String> employees;
}
```

L'API EntityManager est très semblable aux autres technologies Java de persistance d'objet comme JPA et Hibernate, en ce qu'elle synchronise un graphe d'instances d'objets Java gérés avec le stockage de persistance. Dans ce cas, le magasin persistant est une grille de données eXtreme Scale, où chaque entité est représentée sous la forme d'une mappe et où la mappe contient les données de l'entité plutôt que les instances d'objets.

Clés de cache

WebSphere eXtreme Scale utilise des mappes de hachage pour stocker les données dans la grille. Un objet Java est alors utilisé en tant que clé.

Instructions

Lorsque vous choisissez une clé, prenez en compte les exigences suivantes :

- Les clés ne peuvent jamais être modifiées. Si une partie de la clé doit être modifiée, l'entrée de cache doit être supprimée, puis réinsérée.
- Les clés doivent être de petite taille. Les clés étant utilisées dans chaque opération d'accès aux données, il est judicieux de faire en sorte qu'elles gardent une taille modeste afin qu'elles puissent être sérialisées de manière efficace et qu'elles utilisent moins de mémoire.
- Implémentez un bon algorithme hash et equals. Les méthodes hashCode et equals(Object o) doivent toujours être remplacées pour chaque objet clé.
- Mettez en cache le code hashCode de la clé. Si possible, mettez en cache le code de hachage de l'instance de l'objet de clé pour accélérer les calculs hashCode(). La clé étant non modifiable, le code hashCode doit pouvoir être mis en cache.
- Evitez de dupliquer la clé dans la valeur. Lorsque vous utilisez l'API ObjectMap, il est commode de stocker la clé dans l'objet de la valeur. Une fois cette opération exécutée, les données de la clé sont dupliquées en mémoire.

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 d'heure sont créés en fonction du même fuseau horaire, surtout

en cas de déploiement sur des serveurs correspondant à des fuseaux horaires différents. En utilisant les mêmes objets de date et d'heure de fuseau horaire, l'application respecte les heures et les données peuvent être interrogées en fonction 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 d'heure, Java utilise le fuseau horaire local et peut entraîner des 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.

Configuration d'un environnement de développement autonome

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

Avant de commencer

Installez le produit WebSphere eXtreme Scale dans un répertoire nouveau ou vide et appliquez le dernier groupe de correctifs WebSphere eXtreme Scale. Vous pouvez également utiliser la version d'évaluation de WebSphere eXtreme Scale en décompressant le fichier zip. Pour plus d'informations sur l'installation, voir les informations relatives à l'installation de WebSphere eXtreme Scale autonome ou WebSphere eXtreme Scale Client dans *Guide d'administration*.

Procédure

- Configurez Eclipse pour générer et exécuter une application Java SE avec WebSphere eXtreme Scale.
  1. Définissez une bibliothèque utilisateur pour permettre à votre application de référencer des API WebSphere 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) Accédez au fichier `objectgrid.jar` ou `ogclient.jar` et sélectionnez-le dans le répertoire *wxs_root*`/lib`. Cliquez sur **OK**. Sélectionnez le fichier `ogclient.jar` si vous développez des applications client ou des caches en mémoire locaux. Si vous développez et testez des serveurs eXtreme Scale , utilisez le fichier `objectgrid.jar`.

2) Pour inclure la documentation Javadoc des API ObjectGrid, sélectionnez l'emplacement de la documentation du fichier `objectgrid.jar` ou `ogclient.jar` que vous avez ajouté précédemment. 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 génération 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. Voir Configuration d'un ORB personnalisé pour plus d'informations :

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

Que vous ayez une application client ou serveur, utilisez les mêmes étapes de base pour exécuter l'application dans Apache Tomcat dans Rational Application Developer. Dans le cas d'une application client, vous voulez 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 voulez activer une application Java EE dans l'interface Rational Application Developer avec une installation autonome de WebSphere eXtreme Scale. Suivez ces instructions pour configurer une application Java EE du projet pour l'utilisation de la bibliothèque client WebSphere eXtreme Scale.

Avant de commencer

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

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

Procédure

1. Ajoutez la bibliothèque d'exécution WebSphere eXtreme Scale au chemin de génération Java EE.

   Application client Dans ce scénario, vous voulez configurer et exécuter une application Web pour utiliser 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** et fermez 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 voulez configurer et exécuter une application Web pour exécuter un serveur WebSphere eXtreme Scale intégré dans Rational Application Developer.

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 sélectionnez *rép_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** et fermez 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 dans la perspective J2EE et cliquez sur l'onglet **Serveurs** dans la sous-fenêtre inférieure. 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...**. 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 vers le volet Configurées à droite sur le serveur, et cliquez sur **Terminer**.
3. Résolvez les éventuelles erreurs restantes pour le projet. Procédez comme suit pour éliminer les erreurs dans le volet Problèmes :
   a. Cliquez sur **Projet** > **Nettoyer** > *nom_projet*. Cliquez sur **OK**. Compilez le projet.
   b. Cliquez avec le bouton droit de la souris sur le projet Java EE et sélectionnez **Build Path** > **Configure Build Path**.
   c. Cliquez sur l'onglet **Bibliothèques**. Vérifiez que le chemin est correctement configuré :
      - **Pour les applications client :** vérifiez que Apache Tomcat, eXtremeScaleClient et Java 1,5 JRE se trouvent dans le chemin.

- **Pour les applications server :** vérifiez que Apache Tomcat, eXtremeScale et Java 1,5 JRE se trouvent dans le chemin.

4. Créez une configuration d'exécution pour exécuter l'application.
   a. Dans le menu **Run**, sélectionnez **Exécuter configurations**.
   b. Cliquez avec le bouton droit de la souris sur la catégorie Application Java et sélectionnez **New**.
   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. Voir Configuration d'un ORB personnalisé pour plus d'informations :

      ```
      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

Après avoir configuré et exécuté une application Web avec le client WebSphere eXtreme Scale dans Rational Application Developer, vous pouvez développer un servlet. Ce servlet utilise les API WebSphere eXtreme Scale pour stocker et extraire des données d'une grille de données distante.

Une fois que vous avez activé une application Java EE dans l'interface Rational Application Developer avec une installation autonome de WebSphere eXtreme Scale, vous pouvez développer un servlet qui utilise les API système WebSphere eXtreme Scale pour démarrer et arrêter les services de catalogue.

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

Configuration et exécution d'une application Java EE avec un client ou un serveur WebSphere eXtreme Scale avec l'environnement d'exécution WebSphere Application Server intégré à Rational Application Developer. Si vous configurez un serveur, le démarrage de WebSphere Application Server démarre automatiquement WebSphere eXtreme Scale.

Avant de commencer

Les étapes suivantes s'appliquent à WebSphere Application Server Version 7.0 avec Rational Application Developer Version 7.5. Elles peuvent varier si vous utilisez des versions différentes de ces produits.

Installez Rational Application Developer avec les extensions WebSphere Application Server Test Environment.

Installez le client ou le serveur WebSphere eXtreme Scale dans l'environnement de test WebSphere Application Server, Version 7.0 dans le répertoire *rad_home*\runtimes\base_v7. Pour plus d'informations, voir Installation de WebSphere eXtreme Scale ou de WebSphere eXtreme Scale Client avec WebSphere Application Server.

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. Cliquez avec le bouton droit sur **WebSphere Application Server v7.0** et sélectionnez **Démarrer**.

Chapitre 5. Développement d'applications

Développez des applications qui accèdent à la grille de données. Les tâches de développement d'applications sont les suivantes :

- Accès aux données
- Plug-in et API du système
- Intégration JPA
- Intégration Spring

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

Connexion aux instances ObjectGrid distribuées à l'aide d'un programme

Vous pouvez vous connecter à un ObjectGrid réparti avec un point de contact de connexion pour le domaine de services de catalogue. Vous devez connaître le nom d'hôte et le port d'écoute de chaque serveur de catalogue dans le domaine de services de catalogue auquel vous souhaitez vous connecter.

Avant de commencer

- Pour vous connecter à une grille de données répartie, vous devez avoir configuré votre environnement côté serveur avec un service de catalogue et des serveurs de conteneur.
- Vous devez disposer du port d'écoute de chaque service de catalogue. Pour plus d'informations, voir Planification des ports réseau.

Pourquoi et quand exécuter cette tâche

La méthode getObjectGrid(ClientClusterContext ccc, String objectGridName) se connecte au domaine de services de catalogue spécifié et renvoie une instance ObjectGrid client correspondant à une instance ObjectGrid côté serveur. Les étapes varient selon que vous utilisez une configuration autonome ou WebSphere Application Server.

Procédure

- Connectez-vous à une grille de données réparties autonome à l'aide de points de contact de service de catalogue explicite.

```
// Créer une instance ObjectGridManager.

    ObjectGridManager ogm = ObjectGridManagerFactory.getObjectGridManager();

    // Obtenir 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.
    // Les noeuds finaux de serveur de catalogue d'un domaine doit correspondre
                        // à une liste
    // d'éléments séparés par une virgule.

        String catalogServiceEndpoints = "host1:2809,host2:2809";
        ClientClusterContext ccc = ogm.connect(catalogServiceEndpoints, null, null);

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

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

- Connectez-vous à un domaine de services de catalogue depuis une application client hébergée dans WebSphere Application Server, où le domaine de services de catalogue a été configuré en utilisant la console d'administration ou la tâche admin. Les noeuds finaux de service de catalogue peuvent être extraits en utilisant les API de serveur embarqué :

```
...

// Extraire les noeuds finaux du service de catalogue depuis le
                    // singleton ServerPropeties
// qui est configuré dans la console d'administration WebSphere
// ou la tâche admin.

String catalogServiceEndpoints = ServerFactory.getServerProperties()
  .getCatalogServiceBootstrap();
ClientClusterContext ccc = ogm.connect(catalogServiceEndpoints,
  null, null);

...
```

  Si le domaine de services de catalogue dans WebSphere Application Server est hébergé par le gestionnaire de déploiement, les clients extérieurs à la cellule, y compris les clients Java Platform, Standard Edition, doivent se connecter au service de catalogue à l'aide du nom d'hôte du gestionnaire de déploiement et du port d'amorçage IIOP. Lorsque le service de catalogue s'exécute dans des cellules WebSphere Application Server alors que les clients sont exécutés hors des cellules, vous devez rechercher dans les pages de configuration des domaines eXtreme Scale, sur la console d'administration de WebSphere Application Server, les informations permettant de pointer un client sur le service de catalogue.

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.

Interaction avec un objet ObjectGrid en utilisant l'interface ObjectGridManager

La classe ObjectGridManagerFactory et l'interface ObjectGridManager fournissent un mécanisme permettant de créer, d'accès et ajouter des données à des instances ObjectGrid. La classe ObjectGridManagerFactory est une classe auxiliaire statique permettant d'accéder à l'interface ObjectGridManager, 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 qui peuvent être accessibles à plusieurs utilisateurs.

Création d'instances avec l'interface ObjectGrid 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 Répartition de 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. IBM Java Developer Kit (JDK) Version 5 prend en charge 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.

Méthodes createObjectGrid

```
.
/**
 * 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 an ObjectGrid instance
 * @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 an ObjectGrid instance
 * @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;
```

Tâches associées:

«Traitement des problèmes de connectivité», à la page 511
Il existe plusieurs problèmes courants propres aux clients et à la connectivité client que vous pouvez résoudre comme indiqué dans les sections suivantes.

Récupération de données mises en cache avec l'interface ObjectGridManager

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, voir «Connexion aux instances ObjectGrid distribuées à l'aide d'un programme», à la page 131.

Suppression des instances ObjectGrid avec l'interface ObjectGridManager

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. L'interface ObjectGridManager ne conserve pas de référence des instances qui sont supprimées. 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 avec l'interface ObjectGridManager

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 avec des index (API Index)

Utilisez l'indexation pour améliorer l'accès aux données.

Pourquoi et quand exécuter cette tâche

La classe HashIndex est l'implémentation de plug-in d'indexation intégré qui peut prendre en charge les deux interfaces d'indexation d'application intégrée MapIndex et MapRangeIndex. Vous pouvez également créer vos propres index. Vous pouvez ajouter HashIndex sous la forme d'un index statique ou dynamique dans la mappe de sauvegarde, obtenir soit un objet proxy d'index MapIndex ou MapRangeIndex et utiliser l'objet proxy d'index pour rechercher des objets mis en cache.

Si vous souhaitez exécuter une itération dans les clés d'une mappe locale, vous pouvez utiliser l'index par défaut. Cet index ne requiert pas de configuration, mais elle doit être utilisée sur le fragment en utilisant un agent ou une instance ObjectGrid extraits de la méthode ShardEvents.shardActivated(ObjectGrid shard).

Remarque : Dans un environnement réparti, si l'objet d'index est obtenu à partir d'un ObjectGrid client, l'index possède un objet de type client et toutes les opérations d'index sont exécutées dans un ObjectGrid de serveur. Si la mappe est partitionnée, les opérations d'indexation sont exécutées dans chaque partition à distance. Les résultats de chaque partition sont fusionnés avant de renvoyer les résultats à l'application. Les performances sont déterminées par le nombre de partitions et la taille des résultats renvoyés par chaque partition. Les performances peuvent être faibles si les deux facteurs sont élevés.

Procédure

1. Si vous souhaitez utiliser des index autres que l'index local par défaut, ajoutez des plug-ins d'index à la mappe de sauvegarde.
   - **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 intégrée HashIndex est utilisée comme plug-in d'indexation. La classe HashIndex prend en charge les propriétés que les utilisateurs peuvent configurer, telles que Name, RangeIndex et AttributeName dans l'exemple précédent.
     - La propriété **Name** a la valeur `CODE`, une chaîne qui identifie ce plug-in d'index. 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** a la valeur `true`, ce qui signifie que l'application peut transtyper l'objet d'index extrait vers l'interface MapRangeIndex. Si la propriété RangeIndex a la valeur `false`, l'application peut transtyper uniquement l'objet d'index extrait 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 repose sur la requête, la propriété **RangeIndex** doit avoir la valeur `true` dans les index à un seul attribut. Dans le cas d'un index de relations ou d'un index composite, la propriété RangeIndex doit être configurée comme `false`.

– La propriété **AttributeName** a la valeur `employeeCode`, ce qui implique que l'attribut **employeeCode** de l'objet en cache est utilisé pour créer un index à un seul attribut. Si une application doit rechercher des objets mis en cache avec plusieurs attributs, la propriété **AttributeName** peut être affectée d'une liste d'attributs séparés par une virgule pour produire un index composite.

- **Configuration à l'aide d'un programme :**

  L'interface BackingMap comporte deux méthodes que vous pouvez utiliser pour ajouter des plug-ins d'index statique : addMapIndexplugin et setMapIndexplugins. Pour plus d'informations, consultez la documentation d'API. L'exemple suivant crée la même configuration que l'exemple de configuration XML :

```
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);
```

2. Accédez aux clés de mappe et aux valeurs avec des index.

   - **Index local :**

     Pour effectuer une itération dans les clés d'une mappe locale, vous pouvez utiliser l'index par défaut. Cet index fonctionne uniquement avec le fragment, en utilisant un agent ou l'instance ObjectGrid extraits de la méthode ShardEvents.shardActivated(ObjectGrid shard). Reportez-vous à l'exemple suivant :

     ```
     MapIndex keyIndex = (MapIndex)
     objMap.getIndex(MapIndexPlugin.SYSTEM_KEY_INDEX_NAME);
     Iterator keyIterator = keyIndex.findAll();
     ```

   - **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.

     ```
     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);
         }
     ```

   - **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 des index dynamiques.

Voir la documentation de l'API BackingMap pour plus d'informations sur les méthodes createDynamicIndex et removeDynamicIndex.

```
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");
```

Que faire ensuite

Vous pouvez utiliser l'interface DynamicIndexCallback pour obtenir des notifications des événements d'indexation. Pour plus d'informations, voir «Interface DynamicIndexCallback», à la page 147.

Concepts associés:

«Plug-ins d'indexation des données», à la page 326
Le plug-in HashIndex intégré, la classe com.ibm.websphere.objectgrid.plugins.index.HashIndex, est un plug-in MapIndexPlugin que vous pouvez ajouter à un mappe de sauvegarde BackingMap pour générer des index statiques ou dynamiques. Cette classe prend en charge à la fois les interfaces MapIndex et MapRangeIndex. La définition et l'implémentation d'index peuvent considérablement améliorer les performances des requêtes.

«Plug-in d'indexation personnalisée des objets en cache», à la page 332
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.

«Utilisation d'un index composite», à la page 335
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.

«Indexation», à la page 97
Utilisez le plug-in MapIndexPlugin pour générer un ou plusieurs index dans une mappe BackingMap pour prendre en charge l'accès aux données ne correspondant pas à une clé.

Référence associée:

«Attributs du plug-in HashIndex», à la page 329
Vous pouvez utiliser les attributs suivants pour configurer le plug-in HashIndex Ces attributs définissent des propriétés comme si vous utilisiez un attribut ou un index HashIndex composite ou que l'indexation de plage était activée.

Interface DynamicIndexCallback

L'interface DynamicIndexCallback est destinée aux applications qui veulent recevoir des notifications pour les événements d'indexation ready, error ou destroy. DynamicIndexCallback est un paramètre facultatif pour la méthode createDynamicIndex de la mappe de sauvegarde. Avec une instance enregistrée DynamicIndexCallback, les applications peuvent exécuter la logique applicative à la réception de la notification d'un événement d'indexation.

Evénements 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. Voir l'API DynamicIndexCallback dans la documentation d'API pour plus d'informations.

Exemple : 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);
    }
}
```

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 d'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

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. L'appel de la méthode getSession plusieurs fois renvoie un nouvel objet Session chaque fois.

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 cartes sont sauvegardées par un Loader, qui fournit un stockage de persistance aux données dans 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 mémoire et elles ne sont pas validées dans le stockage de persistance fourni par le chargeur. 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 règle de placement de partition par conteneur, vous pouvez utiliser un objet SessionHandle. Un objet SessionHandle contient des informations de partition pour la session en cours et il peut être réutilisé pour une nouvelle session.

Un objet SessionHandle inclut des informations pour 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 disposez d'un objet SessionHandle, vous pouvez appliquer cet indicateur à une session avec la méthode setSessionHandle(SessionHandle target) en utilisant l'indicateur comme cible. Vous pouvez extraire l'objet SessionHandle avec la méthode Session.getSessionHandle.

Etant donné que cela s'applique uniquement dans un scénario de placement par conteneur, l'extraction de l'objet SessionHandle émet une exception IllegalStateException si une grille de données a plusieurs groupes de mappes par conteneur ou aucun groupe de mappes par conteneur. Si vous n'appelez pas la méthode setSessionHandle avant la méthode getSessionHandle, l'objet approprié SessionHandle est sélectionné en fonction de votre configuration des propriétés du client.

Vous pouvez également utiliser la classe auxiliaire SessionHandleTransformer pour convertir l'indicateur dans 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 d'un objet SessionHandle, voir la rubrique sur le routage en fonction de la zone.

Intégration SessionHandle

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

Objet SessionHandle du routage des demandes

Vous pouvez lier un objet SessionHandle à une session des manières suivantes :

Conseil : Dans chacun des appels de méthode suivants après qu'un objet SessionHandle est lié à une session, l'objet SessionHandle peut être obtenu à partir de la méthode Session.getSessionHandle.

- **Appelez la méthode Session.getSessionHandle :** lorsque cette méthode est appelée, si aucun objet SessionHandle n'est lié à la session, un objet SessionHandle est sélectionné de façon aléatoire et lié à la session.

- **Appelez les opérations de création, de lecture, de mise à jour, de suppression transactionnelles :** lorsque ces méthodes sont appelées ou lors de la validation, si aucun objet SessionHandle n'est lié à la session, un objet SessionHandle est sélectionné de manière aléatoire et lié à la session.
- **Appelez la méthode ObjectMap.getNextKey :** lorsque cette méthode est appelée, si aucun objetSessionHandle n'est lié à la session, la demande d'opération est acheminée de façon aléatoire vers les partitions individuelles jusqu'à ce qu'une clé soit obtenue. Si une clé est renvoyée par une partition, un objet SessionHandle correspondant à cette partition est lié à la session. Si aucune clé n'est trouvée, aucun objet SessionHandle n'est lié à la session.
- **Appelez la méthode QueryQueue.getNextEntity ou QueryQueue.getNextEntities :** lorsque cette méthode est appelée, si aucun objet SessionHandle n'est lié à la session, la demande d'opération est acheminée de façon aléatoire vers les partitions individuelles jusqu'à ce qu'un objet soit obtenu. Si un objet est renvoyé par une partition, un objet SessionHandle correspondant à cette partition est lié à la session. Si aucun objet n'est trouvé, aucun objet SessionHandle n'est lié à la session.
- **Définissez un objet SessionHandle avec la méthode Session.setSessionHandle(SessionHandle sh) :** si un objet SessionHandle est obtenu de la méthode Session.getSessionHandle, l'objet SessionHandle peut être lié à une session. La définition d'un objet SessionHandle a un impact sur le routage des demandes dans la portée de la session à laquelle il est lié.

La méthode Session.getSessionHandle renvoie toujours un objet SessionHandle. La méthode lie également automatiquement un objet SessionHandle dans la session si aucun objet SessionHandle n'est lié à la session. Si vous souhaitez vérifier si une session comporte un objet SessionHandle uniquement, appelez la méthode Session.isSessionHandleSet. Si cette méthode renvoie la valeur `false`, cela implique qu'aucun objet SessionHandle n'est actuellement lié à la session.

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

Vous trouverez ci-après le résumé du comportement de routage des principaux types d'opérations dans le scénario de placement de partition par conteneur concernant les objets SessionHandle .

- **Objet de session avec objet lié SessionHandle**
  - Index - MapIndex et API MapRangeIndex : SessionHandle
  - Query et ObjectQuery : SessionHandle
  - Agent - MapGridAgent et API ReduceGridAgent : SessionHandle
  - ObjectMap.Clear : SessionHandle
  - ObjectMap.getNextKey : SessionHandle
  - QueryQueue.getNextEntity, QueryQueue.getNextEntities : SessionHandle
  - Opérations de création, d'extraction, de mise à jour et de suppression transactionnelles (API ObjectMap et API EntityManager) : SessionHandle
- **Objet de session sans objet lié SessionHandle**
  - Index - MapIndex et API MapRangeIndex : toutes les partitions actives
  - Query et ObjectQuery : partition spécifiée avec 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 objet 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 objet SessionHandle.
– QueryQueue : spécifie une partition avec la méthode QueryQueue.setPartition. Si aucune partition n'est définie, la méthode aléatoire sélectionne une partition à renvoyer. Si un objet est renvoyé, la session actuelle est liée à l'objet SessionHandle qui est lié à la partition renvoyant l'objet. Si aucun objet n'est envoyé, la session n'est pas liée à un objet SessionHandle.
– Opérations de création, d'extraction, de mise à jour et de suppression transactionnelles (API ObjectMap et API EntityManager) : sélection aléatoire d'une partition.

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

- **Comportement de routage QueryQueue**

  Dans le scénario de placement 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 objet SessionHandle, les demandes sont acheminées vers la partition à laquelle l'objet SessionHandle est lié. Si la session n'est pas liée à un objet 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'a pas d'objet SessionHandle ou de partition liés, une partition sélectionnée de manière aléatoire est renvoyée. Si aucune partition n'est trouvée, le processus s'interrompt et aucun objet SessionHandle n'est lié à la session actuelle.

Le fragment de code suivant montre comment utiliser les objets 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 placement par conteneur, utilisez l'objet SessionHandle pour la plupart des opérations. L'objet SessionHandle contrôle le routage vers les partitions. Pour extraire des données, l'objet SessionHandle que vous liez à la session doit correspondre à l'objet SessionHandle d'une transaction de données d'insertion.

Lorsque vous voulez effectuer une opération sans objet SessionHandle défini dans la session, vous pouvez déconnecter un objet SessionHandle d'une session en exécutant un appel de méthode Session.setSessionHandle(null).

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

Mise en cache d'objets sans aucune relation 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 166.

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.

Tâches associées:

«Initiation au développement d'applications», à la page 70
Pour commencer à développer des applications WebSphere eXtreme Scale, configurez n environnement de développement dans Eclipse.

«Tutoriel : Stockage des informations de commande dans des entités», à la page 7
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.

Information associée:

«Leçon 2 du tutoriel d'initiation : Création d'une application client», à la page 63
Pour pouvoir insérer, supprimer, mettre à jour et extraire des données dans votre grille de données, vous devez écrire une application client. L'exemple d'initiation inclut une application client que vous pouvez utiliser pour en savoir plus sur la création de votre propre application client.

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

Avec les mappes dynamiques, vous pouvez créer des mappes après l'initialisation de la grille de données.

Dans les versions précédentes de WebSphere 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 avoir à définir toutes les mappes avant l'initialisation. En utilisant des modèles de mappes, vous pouvez créer des mappes après que ObjectGrid a été initialisé.

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 crée une mappe ObjectMap et une mappe BackingMap associée.

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

Définir un modèle de mappe en XML est aussi simple que définir un attribut booléen dans 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.

Remarque : Lorsque vous définissez des modèles de mappes, vérifiez que les noms des mappes sont suffisamment uniques pour que l'application puisse établir une correspondance avec un seul des modèles de mappes avec la méthode Session.getMap(String mapName). Si la méthode getMap() correspond à plusieurs modèles de mappes, une exception `IllegalArgumentException` se produit.

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.

Exemple

Vous devez configurer un modèle de mappe pour créer une mappe dynamique. Ajoutez le modèle Boolean à une mappe backingMap 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. Cet élément identifie le groupe de mappes mapSet dans lequel les mappes dynamiques sont définies.

Tenez compte de la modification du comportement de la méthode Session.getMap(String cacheName) lorsque vous utilisez des modèles de mappes. 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, voir les informations sur l'autorisation de client d'application dans *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">
    <backingMap name="objectgrid.session.metadata.dynamicmap.*" template="true"
        lockStrategy="PESSIMISTIC" ttlEvictorType="LAST_ACCESS_TIME">
    <backingMap name="objectgrid.session.attribute.dynamicmap.*"
        template="true" lockStrategy="OPTIMISTIC"/>
    <backingMap name="datagrid.session.global.ids.dynamicmap.*"
        lockStrategy="PESSIMISTIC"/>
 </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">
    <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 :

- L'élément 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.
- Lorsque vous définissez des modèles de mappes, vérifiez que les noms des mappes sont suffisamment uniques pour que l'application puisse établir une correspondance avec un seul des modèles de mappes avec la méthode

Session.getMap(String mapName). Si la méthode getMap() correspond à plusieurs modèles de mappes, une exception `IllegalArgumentException` se produit.

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 dans WebSphere Application Server, entre autres, appliquent cette approche. Les API fondées sur les mappes connaissent toutefois des limitations.

L'API EntityManager simplifie l'interaction avec la grille de données en facilitant la déclaration et l'interaction avec un graphique complexe d'objets associés.

Limitations de l'API fondée sur les mappes

Si vous utilisez une API basée sur les mappes, telle que le cache dynamique dans WebSphere Application Server ou l'API ObjectMap, tenez compte des points suivants :

- Les index et les requêtes doivent utiliser la réflexion pour les zones de requête et les propriétés dans les objets cache.
- Il est nécessaire de sérialiser les données pour améliorer les performances des objets complexes.
- Il est difficile de manipuler des graphes d'objets. Vous devez développer l'application pour stocker les références artificielles entre les objets et joindre manuellement les objets.

Avantages de l'API EntityManager

L'API EntityManager utilise l'infrastructure de mappe existante, mais elle convertit les objets entités en ou depuis des blocs de données 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.

Ce groupe d'API suit le type de programmation POJO (Plain Old Java) adopté par la plupart des infrastructures.

Tâches associées:

«Tutoriel : Stockage des informations de commande dans des entités», à la page 7
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.

Référence associée:

«Agent d'instrumentation des performances d'entité», à la page 463
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.

«Définition d'un schéma d'entité», à la page 170
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.

«Programmes d'écoute d'entité et méthodes de rappel», à la page 186
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.

«Exemple de programme d'écoute d'entité», à la page 190
Vous pouvez écrire des programmes d'écoute d'entité en fonction de vos exigences. Vous trouverez ci-après plusieurs exemples de script.

«Interface EntityTransaction», à la page 201
Vous pouvez utiliser l'interface EntityTransaction pour démarquer les transactions.

Gestion des relations

Les langages orientés objet comme Java et les bases de données relationnelles prennent en charge les relations et les associations. Les relations diminuent la quantité d'espace de stockage utilisé grâce à l'utilisation de référence d'objet ou de clés externes.

Lorsque vous utilisez des relations dans une grille de données, les données doivent être organisées dans un arbre contraint. Un type de racine doit exister dans l'arborescence et tous les enfants ne doivent être associés qu'à une seule racine. Exemple : un département pourra avoir de nombreux salariés et un salarié pourra avoir de nombreux projets. Mais un projet ne pourra avoir de nombreux salariés qui appartiennent à des départements différents. Une fois qu'une racine est définie, tous les accès à cet objet root et à ses descendants sont gérés via la racine. WebSphere eXtreme Scale utilise le code de hachage de la clé de l'objet racine pour choisir une partition. Par exemple :

```
partition = (hashCode MOD numPartitions).
```

Lorsque toutes les données d'une relation sont liées à une seule instance d'objet, l'instance, l'arborescence peut être située en totalité dans la même partition et il suffit d'une transaction pour y accéder de manière très efficace. Si les données embrassent plusieurs relations, plusieurs partitions seront nécessaires, ce qui implique des appels distants supplémentaires, avec le risque de goulots d'étranglement pour les performances.

Données de référence

Certaines relations incluent des données de recherche ou de référence, telles que `CountryName`. Avec des données de recherche ou de référence, les données doivent exister dans chaque partition. Les données sont accessibles à n'importe quelle clé racine et le même résultat est renvoyé. Les données de référence de ce type ne doivent être utilisées que dans les cas où les données sont relativement statiques. La mise à jour de ces données peut être coûteuse, car les données doivent être mis à jour dans chaque partition. L'API DataGrid est une technique usuellement employée pour conserver à jour les données de référence.

Coûts et avantages de la normalisation

La normalisation des données en utilisant des relations permet de réduire la quantité de mémoire utilisée par la grille de données, car la réplication des données diminue. Mais, en règle générale, plus l'on ajoute de données relationnelles et moindre est leur extensibilité. En effet, lorsque les données sont regroupées, la maintenance de leurs relations devient plus onéreuse et il devient difficile de leur conserver des tailles gérables. Comme la grille partitionne les données en fonction de la clé de la racine de l'arborescence, la taille de celle-ci n'est pas prise en compte. Par conséquent, si vous avez un grand nombre de relations pour une instance d'arborescence, la grille de données peuvent devenir déséquilibré, provoquant une partition détenant davantage de données que les autres.

Lorsque les données sont dénormalisées ou mises à plat, les données qui seraient normalement partagées entre deux objets sont dupliquées et chaque table peut être partitionnée indépendamment pour améliorer l'équilibre de la grille de données. Bien que cela augmente la quantité de mémoire utilisée, cela permet à l'application de se mettre à l'échelle puisqu'on est sûr que l'accès à une seule ligne de données donne accès à toutes les données nécessaires. C'est parfait pour les grilles qui sont essentiellement en lecture où la maintenance des données devient plus onéreuse.

Pour plus d'informations, voir Classifying XTP systems and scaling.

Gérer les relations à l'aide des API d'accès aux données

ObjectMap est l'API la plus rapide, la plus flexible et la plus granulaire de toutes les API d'accès aux données, car elle fournit une approche transactionnelle à base de sessions pour l'accès aux données présentes dans la grille des mappes. L'API ObjectMap permet aux clients d'utiliser des opérations communes CRUD (create, read, update et delete) pour gérer des paires clé-valeur d'objets dans la grille de données répartie.

Lorsqu'on utilise l'API ObjectMap, les relations entre les objets doivent être exprimées par l'incorporation dans l'objet parent de la clé externe de toutes les relations.

Exemple :

```
public class Department {
 Collection<String> employeeIds;
}
```

L'API EntityManager simplifie la gestion des relations en extrayant les données persistantes des objets, y compris les clés externes. Lorsque l'objet est

ultérieurement récupéré dans la grille de données, le graphique des relations est reconstruit, comme dans l'exemple suivant.

```
@Entity
public class Department {
 Collection<String> employees;
}
```

L'API EntityManager est très semblable aux autres technologies Java de persistance d'objet comme JPA et Hibernate, en ce qu'elle synchronise un graphe d'instances d'objets Java gérés avec le stockage de persistance. Dans ce cas, le magasin persistant est une grille de données eXtreme Scale, où chaque entité est représentée sous la forme d'une mappe et où la mappe contient les données de l'entité plutôt que les instances d'objets.

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 «Gestionnaire d'entités dans un environnement distribué», à la page 179.

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.

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.

Concepts associés:

«Optimisation des performances de l'interface EntityManager», à la page 462
L'interface EntityManager sépare les applications de l'état conservé dans son magasin de données de grille de données de serveurs.

«Mise en cache d'objets et de leurs relations (API EntityManager)», à la page 166
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 dans WebSphere Application Server, entre autres, appliquent cette approche. Les API fondées sur les mappes connaissent toutefois des limitations. L'API EntityManager simplifie l'interaction avec la grille de données en facilitant la déclaration et l'interaction avec un graphique complexe d'objets associés.

«Gestionnaire d'entités dans un environnement distribué»
Vous pouvez utiliser l'API EntityManager avec un ObjectGrid local ou dans un environnement distribué eXtreme Scale. 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.

«Interactions avec EntityManager», à la page 183
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.

«Stratégie FetchPlan de l'EntityManager», à la page 192
Un plan d'extraction (FetchPlan) est la stratégie utilisée par EntityManager pour extraire des objets associés si l'application doit accéder aux relations.

«Files d'attente des requêtes d'entité», à la page 196
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.

Tâches associées:

«Tutoriel : Stockage des informations de commande dans des entités», à la page 7
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.

Gestionnaire d'entités dans un environnement distribué

Vous pouvez utiliser l'API EntityManager avec un ObjectGrid local ou dans un environnement distribué eXtreme Scale. 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 de déploiement ou XML descripteur de la grille de données

Ces fichiers indiquent les entités et les mappes BackingMaps qu'un serveur héberge.

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 un ObjectGrid avec le nom `bookstore` et une mappe de sauvegarde associée nommée `order`. Le fichier de fragment `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>
```

Le fichier `objectgrid.xml` indique le fichier `entity.xml` avec l'attribut **`entityMetadataXMLFile`**. La valeur peut être un répertoire relatif ou un chemin absolu.

- **Pour un répertoire relatif :** indiquez l'emplacement relatif à l'emplacement du fichier `objectgrid.xml`.
- **Pour un chemin d'accès absolu :** indiquez l'emplacement à l'aide d'une URL, telle que `file://` ou `http://`.

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 la classe Order a les zones **`orderNumber`** et **`desc`** annotées de la même manière.

Un fichier sans classe équivalent `entity.xml` se présente comme suit :

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 plus d'informations sur le démarrage des serveurs, voir *Guide d'administration*. Vous utilisez les fichiers `deployment.xml` et `objectgrid.xml` pour démarrer le serveur de catalogue.

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");
   ```

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 dans le même chemin défini sur le serveur, le client utilise la configuration d'entité sur le 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.

Client et schéma 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 pas 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 remplacerait la grille de données par un fichier `entity.xml`, tel que :

```
<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.

Une grille de données locale n'a pas besoin d'un fichier XML. 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`.

Tâches associées:

«Tutoriel : Stockage des informations de commande dans des entités», à la page 7
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.

Référence associée:

«Agent d'instrumentation des performances d'entité», à la page 463
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.

«Définition d'un schéma d'entité», à la page 170
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.

«Programmes d'écoute d'entité et méthodes de rappel», à la page 186
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.

«Exemple de programme d'écoute d'entité», à la page 190
Vous pouvez écrire des programmes d'écoute d'entité en fonction de vos exigences. Vous trouverez ci-après plusieurs exemples de script.

«Interface EntityTransaction», à la page 201
Vous pouvez utiliser l'interface EntityTransaction pour démarquer les transactions.

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. Voir le «Tutoriel : Stockage des informations de commande dans des entités», à la page 7 tutoriel du gestionnaire d'entités dans *Présentation du produit* pour plus d'informations.

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 196.

Tâches associées:

«Tutoriel : Stockage des informations de commande dans des entités», à la page 7
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.

Référence associée:

«Agent d'instrumentation des performances d'entité», à la page 463
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.

«Définition d'un schéma d'entité», à la page 170
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.

«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.

«Exemple de programme d'écoute d'entité», à la page 190
Vous pouvez écrire des programmes d'écoute d'entité en fonction de vos exigences. Vous trouverez ci-après plusieurs exemples de script.

«Interface EntityTransaction», à la page 201
Vous pouvez utiliser l'interface EntityTransaction pour démarquer les transactions.

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.

Concepts associés:

«Optimisation des performances de l'interface EntityManager», à la page 462
L'interface EntityManager sépare les applications de l'état conservé dans son magasin de données de grille de données de serveurs.

«Mise en cache d'objets et de leurs relations (API EntityManager)», à la page 166
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 dans WebSphere Application Server, entre autres, appliquent cette approche. Les API fondées sur les mappes connaissent toutefois des limitations. L'API EntityManager simplifie l'interaction avec la grille de données en facilitant la déclaration et l'interaction avec un graphique complexe d'objets associés.

«Gestionnaire d'entités dans un environnement distribué», à la page 179
Vous pouvez utiliser l'API EntityManager avec un ObjectGrid local ou dans un environnement distribué eXtreme Scale. 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.

«Interactions avec EntityManager», à la page 183
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.

«Stratégie FetchPlan de l'EntityManager», à la page 192
Un plan d'extraction (FetchPlan) est la stratégie utilisée par EntityManager pour extraire des objets associés si l'application doit accéder aux relations.

«Files d'attente des requêtes d'entité», à la page 196
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.

Tâches associées:

«Tutoriel : Stockage des informations de commande dans des entités», à la page 7
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.

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.

Concepts associés:

«Optimisation des performances de l'interface EntityManager», à la page 462
L'interface EntityManager sépare les applications de l'état conservé dans son magasin de données de grille de données de serveurs.

«Mise en cache d'objets et de leurs relations (API EntityManager)», à la page 166
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 dans WebSphere Application Server, entre autres, appliquent cette approche. Les API fondées sur les mappes connaissent toutefois des limitations. L'API EntityManager simplifie l'interaction avec la grille de données en facilitant la déclaration et l'interaction avec un graphique complexe d'objets associés.

«Gestionnaire d'entités dans un environnement distribué», à la page 179
Vous pouvez utiliser l'API EntityManager avec un ObjectGrid local ou dans un environnement distribué eXtreme Scale. 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.

«Interactions avec EntityManager», à la page 183
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.

«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.

«Files d'attente des requêtes d'entité», à la page 196
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.

Tâches associées:

«Tutoriel : Stockage des informations de commande dans des entités», à la page 7
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.

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*.

  éDans 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.

Tâches associées:

«Tutoriel : Stockage des informations de commande dans des entités», à la page 7
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.

Référence associée:

«Agent d'instrumentation des performances d'entité», à la page 463
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.

«Définition d'un schéma d'entité», à la page 170
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.

«Programmes d'écoute d'entité et méthodes de rappel», à la page 186
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.

«Exemple de programme d'écoute d'entité», à la page 190
Vous pouvez écrire des programmes d'écoute d'entité en fonction de vos exigences. Vous trouverez ci-après plusieurs exemples de script.

«Interface EntityTransaction», à la page 201
Vous pouvez utiliser l'interface EntityTransaction pour démarquer les transactions.

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 213.

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.

Tâches associées:

«Tutoriel : Stockage des informations de commande dans des entités», à la page 7
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.

Référence associée:

«Agent d'instrumentation des performances d'entité», à la page 463
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.

«Définition d'un schéma d'entité», à la page 170
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.

«Programmes d'écoute d'entité et méthodes de rappel», à la page 186
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.

«Exemple de programme d'écoute d'entité», à la page 190
Vous pouvez écrire des programmes d'écoute d'entité en fonction de vos exigences. Vous trouverez ci-après plusieurs exemples de script.

«Interface EntityTransaction»
Vous pouvez utiliser l'interface EntityTransaction pour démarquer les transactions.

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.

Concepts associés:

«Optimisation des performances de l'interface EntityManager», à la page 462
L'interface EntityManager sépare les applications de l'état conservé dans son magasin de données de grille de données de serveurs.

«Mise en cache d'objets et de leurs relations (API EntityManager)», à la page 166
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 dans WebSphere Application Server, entre autres, appliquent cette approche. Les API fondées sur les mappes connaissent toutefois des limitations. L'API EntityManager simplifie l'interaction avec la grille de données en facilitant la déclaration et l'interaction avec un graphique complexe d'objets associés.

«Gestionnaire d'entités dans un environnement distribué», à la page 179
Vous pouvez utiliser l'API EntityManager avec un ObjectGrid local ou dans un environnement distribué eXtreme Scale. 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.

«Interactions avec EntityManager», à la page 183
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.

«Stratégie FetchPlan de l'EntityManager», à la page 192
Un plan d'extraction (FetchPlan) est la stratégie utilisée par EntityManager pour extraire des objets associés si l'application doit accéder aux relations.

«Files d'attente des requêtes d'entité», à la page 196
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.

Tâches associées:

«Tutoriel : Stockage des informations de commande dans des entités», à la page 7
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.

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.
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 225 syntaxe
- «Référence pour les requêtes eXtreme Scale», à la page 217

Les rubriques ci-dessous décrivent les modes d'utilisation de l'API de requête :

- «API de requête EntityManager», à la page 213
- «Utilisation de l'API ObjectQuery», à la page 208

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.

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 d'heure sont créés en fonction du même fuseau horaire, surtout en cas de déploiement sur des serveurs correspondant à des fuseaux horaires différents. En utilisant les mêmes objets de date et d'heure de fuseau horaire, l'application respecte les heures et les données peuvent être interrogées en fonction 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 d'heure, Java utilise le fuseau horaire local et peut entraîner des 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.

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.

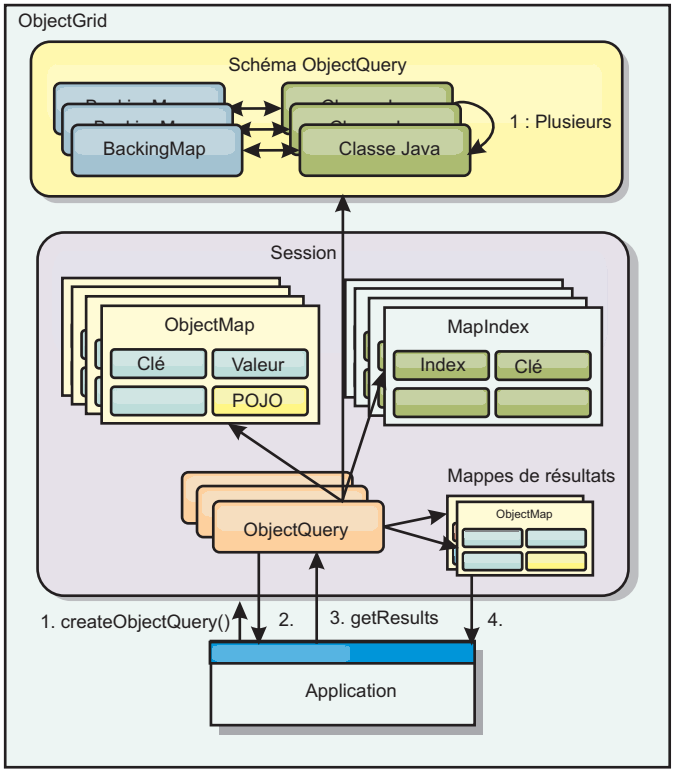


Figure 24. Interaction de la requête avec les mappes de l'objet ObjectGrid, définition d'un schéma pour les classes et association de celui-ci à une mappe ObjectGrid.

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 un exemple de création d'un schéma à l'aide d'un programme ou en utilisant un fichier descripteur XML d'ObjectGrid, voir «Tutoriel ObjectQuery - Etape 3», à la page 3le tutoriel sur ObjectQuery dans *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 des exemples d'utilisation de l'ObjectQuery, voir «Tutoriel : interrogation d'une grille de données en mémoire locale», à la page 1le tutoriel de l'API ObjectQuery dans *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. Voir le «Tutoriel ObjectQuery - Etape 2», à la page 2 tutoriel ObjectQuery dans *Présentation du produit* pour un exemple d'ajout d'un index à un schéma.

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.

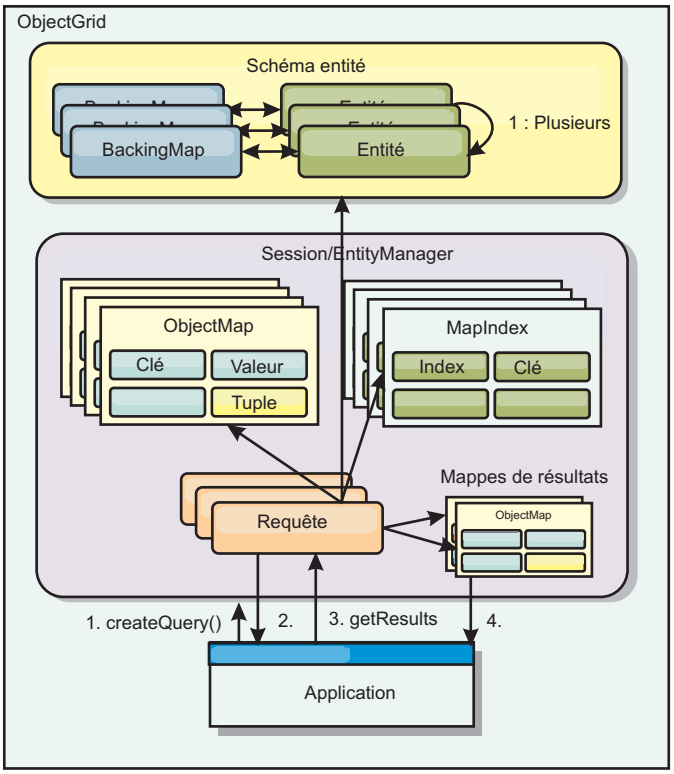


Figure 25. Interaction de la requête avec les mappes de l'objet ObjectGrid, définition du schéma d'entité et association de celui-ci à une mappe ObjectGrid.

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)
```

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.

Programmation 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

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.

Accès aux données et transactions :

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.

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.

.

Transactions :

Les transactions disposent de nombreux avantages pour le stockage et la manipulation de données. Vous pouvez utiliser des transactions pour protéger la grille de données 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 volumineuses 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. Si vous utilisez WebSphere Application Server, vous pouvez utiliser le plug-in WebSphereTransactionCallback.

Attribut CopyMode :

Vous pouvez ajuster le nombre de copies en définissant l'attribut CopyMode des objets BackingMap et ObjectMap dans le fichier descripteur XML d'ObjectGrid.

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_TO_BYTES_RAW

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 réplique 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"/>
```

Référence associée:

Fichier XML du descripteur d'ObjectGrid
Pour configurer WebSphere eXtreme Scale, utilisez un fichier XML de descripteur d'ObjectGrid et l'API ObjectGrid.

Gestionnaire de verrous :

Lorsque vous configurez une stratégie de verrouillage, un gestionnaire de verrous est créé pour la mappe de sauvegarde pour conserver la cohérence des entrées de cache.

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.

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 sur l'implémentation d'une approche de verrouillage optimiste, reportez-vous à la section OptimisticCallback dans les considérations relatives aux chargeurs 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 à la rubrique «Verrouillage pessimiste», à la page 236.

Répartition des transactions :

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 de client et la réplication entre homologues, sont des exemples de répartition de modifications transactionnelles JMS. Voir informations relatives à la configuration de la réplication entre homologues avec JMS dans *Guide d'administration* pour plus d'informations.

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 de transaction, voir les informations relatives à la répartition des modifications entre des machines JVM (Java Virtual Machines) homologues dans *Guide d'administration*.

Transactions à partition unique et cross-data-grid :

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 à prendre en compte sont les transactions à une seule partition et les transactions every-partition (inter-data-grid).

En règle générale, les interactions avec le cache peuvent être classées comme des transactions à partition unique ou des transactions cross-data-grid, comme décrit dans la section suivante.

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 que le serveur défaillant est remplacé et revenu dans la grille de données, le cache retrouve 100 % de sa capacité.

Transactions Cross-data-grid

En termes de performances, de disponibilité et d'évolutivité, les transactions cross-data-grid sont l'opposé des transactions à partition unique. Elles ont accès à toutes les partitions et donc à chaque ordinateur de la configuration. Chaque ordinateur de la grille de données est sollicité pour rechercher les données, puis renvoyer le résultat. La transaction n'est pas terminée tant que tous les ordinateurs n'ont pas répondu, et donc le traitement de l'ensemble de la grille de données est limité 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 cross-data-grid 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. Pendant ce temps, avant la fin du basculement, la grille de données ne peut 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 de données gérait 10 partitions, 10 des 99 ordinateurs restants reçoivent au moins une partition supplémentaire dans le cadre du processus de basculement. L'ajout d'une partition supplémentaire augmente la charge de travail de cet ordinateur d'au moins 10 %. Etant donné que la capacité de la grille de données est limitée à la capacité de traitement de l'ordinateur le plus lent dans une transaction cross-data-grid, en moyenne, le traitement est réduit de 10.

Il est préférable d'utiliser des transactions à une seule partition que des transactions cross-data-grid pour l'évolutivité avec un cache d'objet haute disponibilité réparti tel que WebSphere eXtreme Scale. L'optimisation des performances de ces systèmes requiert l'utilisation de techniques qui diffèrent des méthodologies relationnelles traditionnelles, mais vous pouvez transformer les transactions cross-data-grid en transactions évolutives à 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 évolueront de manière linéaire à mesure que des ordinateurs sont ajoutés à la grille de données.

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 votre entreprise prévoit que vous avez besoins de 10 % de capacité de plus pour l'année prochaine, augmentez le nombre d'ordinateurs de 10 % dans la grille de données. 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. Vous devez implémenter un modèle de données pouvant fonctionner dans la couche d'applications ou en tant que plug-in de la grille de données répartie pour garantir la cohérence des données.

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 de données n'évoluera pas si un enregistrement est utilisé un nombre de fois disproportionné par rapport aux autres enregistrements. Les performances de la grille de données seront limitées par celle de l'ordinateur qui contient 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 cross-data-grid, mais dans l'ensemble elle était déjà aussi inefficace qu'une opération cross-data-grid, car ses performances étaient liées à celles de l'ordinateur contenant 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 pour le scénario de connexion consiste à utiliser une transaction cross-data-grid 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 basées sur l'ID du client deviennent des transactions cross-data-grid. 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 de données contient 10 000 produits, seul 1/1O 000 de la grille doit être verrouillé lors de la mise à jour. Dans l'organisation B, la grille de données contient seulement 50 magasins, si bien qu'1/50 de la grille doit être verrouillé pour terminer 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 cross-data-grid, car chaque entité de magasin doit être accessible afin de déterminer si le produit est vendu dans ce magasin, ce qui donne un avantage de performance remarquable pour l'organisation A.

Evolutivité avec les données normalisées

L'évolution du traitement des données est l'une des utilisations légitimes des transactions cross-data-grid. Si une grille de données comporte 5 ordinateurs et qu'une transaction cross-data-grid est distribuée pour trier environ 100 000 enregistrements sur chaque ordinateur, cette transaction trie 500 000 enregistrements. Si l'ordinateur le plus lent dans la grille de données peut traiter 10 de ces transactions par seconde, la grille de données est capable de trier 5 000 000 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 l'ordinateur le plus lent à 5 transactions par seconde, ce qui réduit le traitement de la grille de données à 5 transactions par seconde. Cependant, la grille trie toujours les données de 5 000 000 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 la grille de données reste la même à 10 demandes par seconde, mais chaque transaction traite désormais 1 000 000 d'enregistrements, si bien que la grille a doublé sa capacité en terme de traitement des enregistrements pour atteindre 10 000 000 par seconde.

Pour les applications, telles qu'un moteur de recherche, qui ont besoin d'évoluer à la fois en termes de traitement des données pour s'adapter à la taille croissante de l'Internet et de capacité pour s'adapter à l'augmentation du nombre d'utilisateurs, vous devez créer plusieurs grilles de données avec un traitement circulaire des demandes entre les grilles. Si vous devez augmenter le débit, ajoutez des ordinateurs et ajoutez une grille de données aux demandes de service. Si vous devez augmenter le traitement des données, ajoutez des ordinateurs et ne modifiez pas le nombre de grilles.

Utilisation du verrouillage

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.

Les 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.

Verrous partagés, pouvant être mis à niveau et exclusifs

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.

Tâches associées:

«Traitement des problèmes d'interblocage», à la page 517
Les sections ci-dessous décrivent certains scénarios d'interblocage courants et des suggestions permettant de les éviter.

Implémentation de gestion des exceptions dans les scénarios de verrouillage :

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 intercepter les exceptions inattendues et appeler la méthode rollback lorsqu'un événement imprévu survient.

Procédure

1. Intercepter l'exception et affichez le message résultant.

   ```
   try {
   ...
   } catch (ObjectGridException oe) {
   System.out.println(oe);
   }
   ```

   L'exception suivante s'affiche :

   ```
   com.ibm.websphere.objectgrid.plugins.LockDeadlockException: Message
   ```

Ce message représente la chaîne transmise en tant que paramètre lorsque l'exception est créée et émise.

2. Annuler la transaction après une exception :

```
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 bloc `finally` dans le fragment de code garantit qu'une transaction est annulée 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.

Configuration d'une stratégie de verrouillage :

Vous pouvez définir une stratégie optimiste, pessimiste ou sans verrouillage sur chaque BackingMap de la configuration de WebSphere eXtreme Scale.

Pourquoi et quand exécuter cette tâche

Chaque instance de mappe de sauvegarde BackingMap peut être configurée pour utiliser l'une de ces stratégies de verrouillage :

1. mode de verrouillage optimiste
2. mode de verrouillage pessimiste
3. aucun

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, voir les «Stratégies de verrouillage», à la page 235informations relatives aux stratégies de verrouillage dans *Présentation du produit*.

Procédure

- **Configurez une stratégie de verrouillage optimiste**
  - Utilisation de la méthode setLockStrategy 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("optimisticMap");
bm.setLockStrategy( LockStrategy.OPTIMISTIC );
```

  - Utilisation de l'attribut lockStrategy dans :

```
<?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="optimisticMap"
                lockStrategy="OPTIMISTIC"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

- **Configurez une stratégie de verrouillage pessimiste**
  - Utilisation de la méthode setLockStrategy à l'aide d'un programme :

```
specify pessimistic strategy programmatically
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("pessimisticMap");
bm.setLockStrategy( LockStrategy.PESSIMISTIC);
```

  - Utilisation de l'attribut lockStrategy dans .

```
specify pessimistic strategy using 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="pessimisticMap"
```

```
                lockStrategy="PESSIMISTIC"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

- **Configurez une stratégie sans verrouillage**
  - Utilisation de la méthode setLockStrategy 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("noLockingMap");
bm.setLockStrategy( LockStrategy.NONE);
```

  - Utilisation de l'attribut lockStrategy dans :

```
<?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="noLockingMap"
                lockStrategy="NONE"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

Que faire ensuite

Pour éviter une exception java.lang.IllegalStateException, vous devez appeler la méthode setLockStrategy avant d'appeler les méthodes initialize ou getSession sur l'instance ObjectGrid.

Définition de la valeur de délai d'attente de verrouillage :

La valeur de délai d'attente de verrouillage sur une instance BackingMap évite à une application d'attendre indéfiniment l'octroi d'un mode de verrouillage suite à un interblocage généré par une erreur de l'application.

Avant de commencer

Pour définir la valeur du délai d'attente de verrouillage, la stratégie de verrouillage doit être OPTIMISTIC ou PESSIMISTIC. Pour plus d'informations, voir «Configuration d'une stratégie de verrouillage», à la page 251.

Pourquoi et quand exécuter cette tâche

Lorsqu'une exception LockTimeoutException se produit, l'application doit déterminer si le dépassement du délai est provoqué par un ralentissement de l'application ou par un 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.

Pour empêcher les interblocages, le gestionnaire de verrouillage possède un délai d'expiration par défaut de 15 secondes. Si le délai d'expiration est dépassé, une exception LockTimeoutException est générée. Si votre système est très chargé, la valeur de délai d'attente par défaut peut générer des exceptions LockTimeoutException alors qu'il n'existe aucun interblocage. Dans ce cas, vous pouvez augmenter le délai d'attente à l'aide d'un programme ou dans le fichier XML descripteur ObjectGrid.

Procédure

- Définissez une valeur de délai d'attente de verrouillage à l'aide d'un programme sur une instance BackingMap avec la méthode setLockTimeout.

  L'exemple suivant montre comment définir les 60 secondes comme valeur de délai d'attente de verrouillage pour la mappe de sauvegarde map1 :

  ```
  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.
- Configurez la valeur du délai d'attente de verrouillage à l'aide de l'attribut locktimeout dans le Fichier XML du descripteur d'ObjectGridfichier XML descripteur 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="test">
              <backingMap name="optimisticMap"
                  lockStrategy="OPTIMISTIC"
           lockTimeout="60"/>
          </objectGrid>
      </objectGrids>
  </objectGridConfig>
  ```
- Remplacez le délai d'attente de verrou d'une seule instance ObjectMap. Utilisez la méthode ObjectMap.setLockTimeout pour remplacer le délai d'attente de verrou d'une instance ObjectMap. 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.

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;
}
```

Exécution de la logique métier en parallèle sur la grille de données (API DataGrid)

L'API DataGrid fournit une interface de programmation simple pour exécuter la logique métier sur toute la grille de données ou sous-ensemble de la grille de données en parallèle avec l'emplacement des données.

API DataGrid et partitionnement :

Avec les API DataGrid, un client peut envoyer des demandes à une partition, un sous-ensemble de partitions ou à toutes les partitions d'une grille de données. Le client peut spécifier une liste de clés et WebSphere eXtreme Scale détermine l'ensemble des partitions qui hébergent ces clés. La demande est alors transmise en parallèle à toutes les partitions de l'ensemble et le client attend les résultats. Le client peut également envoyer des demandes sans spécifier de clés. Dans ce cas, les demandes sont expédiées à la totalité des partitions.

Les agents qui sont déployés dans la grille de données ne fonctionnent pas en mode client. Ces agents opèrent directement sur le fragment primaire. Ce fonctionnement direct se traduit par des performances maximales, permettant à des dizaines de milliers, voire plus, de transactions par seconde l'agent travaillant avec les données à pleine vitesse mémoire. L'utilisation directe du fragment primaire signifie également que l'agent ne voit que les données qui sont dans ce fragment. Cela offre un certain nombre de possibilités intéressantes impossibles à réaliser sur un client.

Un client eXtreme Scale normal doit pouvoir déterminer la partition à partir de la transaction car le client a besoin de router la demande. Si un agent est directement rattaché à un fragment, aucun routage n'est nécessaire. Toutes les demandes aboutissent à ce fragment. L'agent étant directement attaché à un fragment, les données qui se trouvent dans les autres mappes du fragment sont accessibles sans que l'on ait à se préoccuper des clés communes de partitionnement, etc., car il n'intervient aucun routage.

Agents DataGrid et mappes fondées sur des entités :

Une mappe contient des objets clés et des objets valeurs. Un objet clé est un bloc de données généré, de même qu'un objet valeur. Un agent est normalement fourni avec les objets clés définis dans l'application.

Un objet clé est un bloc de données généré, de même qu'un objet valeur. Un agent est normalement fourni avec les objets clés définis dans l'application. Ces objets sont les objets clés utilisés par l'application ou les blocs de données, dans le cas d'une mappe d'entités. Une application qui utilise des entités ne souhaite pas traiter directement avec les blocs de données, mais préfère utiliser les objets Java mappés vers l'entité.

Une classe d'agent peut donc implémenter l'interface EntityAgentMixin. La classe doit alors implémenter une méthode supplémentaire, getClassForEntity(). Celle-ci renvoie la classe d'entités à utiliser avec l'agent côté serveur. Les clés sont converties dans cette entité avant que les méthodes de traitement et de réduction soient appelées.

Il s'agit d'une sémantique différente de celle d'un agent non-EntityAgentMixin où ces méthodes sont simplement fournies avec les clés. Un agent qui implémente EntityAgentMixin reçoit l'objet Entity qui contient les clés et les valeurs dans un seul objet.

Remarque : Si l'entité n'existe pas sur le serveur, les clés sont au format de bloc de données brut à la place de l'entité gérée.

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.

Configuration des clients à l'aide d'un programme

Vous pouvez configurer un client WebSphere eXtreme Scale en fonction de vos besoins, tels que la nécessité de remplacer les paramètres.

Remplacement de plug-ins

Vous pouvez remplacer les plug-in suivants sur un client :

- **ObjectGrid**
  - TransactionCallback
  - ObjectGridEventListener
- **BackingMap**
  - Evictor
  - MapEventListener
  - attribut numberOfBuckets
  - attribut ttlEvictorType
  - attribut timeToLive

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.

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.

Activation de 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.

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.

Concepts associés:

«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.

«Présentation des services de données REST», à la page 117
Le service de données REST de WebSphere eXtreme Scale est un service HTTP Java qui est compatible avec Microsoft WCF Data Services (ex-ADO.NET Data Services) et qui implémente Open Data Protocol (OData). Microsoft WCF Data Services est compatible avec cette spécification lorsqu'on utilise Visual Studio 2008 SP1 et .NET Framework 3.5 SP1.

Référence associée:

«Concurrence optimiste dans le service de données REST», à la page 273
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.

«Protocoles de demande du service de données REST», à la page 274
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.

«Extraction de demandes avec le service de données REST», à la page 275
Une demande trieveEntity 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 sous la forme d'un document de flux Atom, qui est une relation to-many, ou un document Atom Entry qui est une relation to-one.

«Extraction d'éléments autres que des entités à l'aide des services de données REST», à la page 282
Le service de données REST permet d'extraire bien plus que des entités et notamment des collections d'entités, des propriétés, etc.

«Demandes d'insertion avec les services de données REST», à la page 288
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.

«Demandes de mise à jour avec les services de données REST», à la page 292
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.

«Suppression de demandes avec les services de données REST», à la page 296
Le service de données REST de WebSphere eXtreme Scale peut supprimer des entités, des valeurs de propriété et des liens.

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. Cette rubrique indiquent et décrit les fonctionnalités de la spécification qui sont prises en charge et leur mappage 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.

Tâches associées:

«Accès aux données avec le service de données REST», à la page 268
Developpez des applications qui effectuent des opérations à l'aide des protocoles de services de données REST.

Référence associée:

«Concurrence optimiste dans le service de données REST»
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.

«Protocoles de demande du service de données REST», à la page 274
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.

«Extraction de demandes avec le service de données REST», à la page 275
Une demande trieveEntity 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 sous la forme d'un document de flux Atom, qui est une relation to-many, ou un document Atom Entry qui est une relation to-one.

«Extraction d'éléments autres que des entités à l'aide des services de données REST», à la page 282
Le service de données REST permet d'extraire bien plus que des entités et notamment des collections d'entités, des propriétés, etc.

«Demandes d'insertion avec les services de données REST», à la page 288
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.

«Demandes de mise à jour avec les services de données REST», à la page 292
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.

«Suppression de demandes avec les services de données REST», à la page 296
Le service de données REST de WebSphere eXtreme Scale peut supprimer des entités, des valeurs de propriété et des liens.

Concurrence optimiste dans le service de données REST

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 d'informations sur l'accès concurrent optimiste, 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 170.

Concepts associés:

«Opérations avec le service de données REST», à la page 269
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.

«Présentation des services de données REST», à la page 117
Le service de données REST de WebSphere eXtreme Scale est un service HTTP Java qui est compatible avec Microsoft WCF Data Services (ex-ADO.NET Data Services) et qui implémente Open Data Protocol (OData). Microsoft WCF Data Services est compatible avec cette spécification lorsqu'on utilise Visual Studio 2008 SP1 et .NET Framework 3.5 SP1.

Tâches associées:

«Accès aux données avec le service de données REST», à la page 268
Developpez des applications qui effectuent des opérations à l'aide des protocoles de services de données REST.

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).

Concepts associés:

«Opérations avec le service de données REST», à la page 269
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.

«Présentation des services de données REST», à la page 117
Le service de données REST de WebSphere eXtreme Scale est un service HTTP Java qui est compatible avec Microsoft WCF Data Services (ex-ADO.NET Data Services) et qui implémente Open Data Protocol (OData). Microsoft WCF Data Services est compatible avec cette spécification lorsqu'on utilise Visual Studio 2008 SP1 et .NET Framework 3.5 SP1.

Tâches associées:

«Accès aux données avec le service de données REST», à la page 268
Developpez des applications qui effectuent des opérations à l'aide des protocoles de services de données REST.

Extraction de demandes avec le service de données REST

Une demande trieveEntity 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 sous la forme d'un document de flux Atom, qui est une relation to-many, ou un document Atom Entry qui est une 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 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

Pour obtenir la liste complète et la description des expressions qui sont disponibles dans Microsoft WCF Data Services, voir la section 2.2.3.6.1.1 : Synaxe des expressions communes.

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

Concepts associés:

«Opérations avec le service de données REST», à la page 269
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.

«Présentation des services de données REST», à la page 117
Le service de données REST de WebSphere eXtreme Scale est un service HTTP Java qui est compatible avec Microsoft WCF Data Services (ex-ADO.NET Data Services) et qui implémente Open Data Protocol (OData). Microsoft WCF Data Services est compatible avec cette spécification lorsqu'on utilise Visual Studio 2008 SP1 et .NET Framework 3.5 SP1.

Tâches associées:

«Accès aux données avec le service de données REST», à la page 268
Developpez des applications qui effectuent des opérations à l'aide des protocoles de services de données REST.

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.

Concepts associés:

«Opérations avec le service de données REST», à la page 269
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.

«Présentation des services de données REST», à la page 117
Le service de données REST de WebSphere eXtreme Scale est un service HTTP Java qui est compatible avec Microsoft WCF Data Services (ex-ADO.NET Data Services) et qui implémente Open Data Protocol (OData). Microsoft WCF Data Services est compatible avec cette spécification lorsqu'on utilise Visual Studio 2008 SP1 et .NET Framework 3.5 SP1.

Tâches associées:

«Accès aux données avec le service de données REST», à la page 268
Developpez des applications qui effectuent des opérations à l'aide des protocoles de services de données REST.

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

Concepts associés:

«Opérations avec le service de données REST», à la page 269
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.

«Présentation des services de données REST», à la page 117
Le service de données REST de WebSphere eXtreme Scale est un service HTTP Java qui est compatible avec Microsoft WCF Data Services (ex-ADO.NET Data Services) et qui implémente Open Data Protocol (OData). Microsoft WCF Data Services est compatible avec cette spécification lorsqu'on utilise Visual Studio 2008 SP1 et .NET Framework 3.5 SP1.

Tâches associées:

«Accès aux données avec le service de données REST», à la page 268
Developpez des applications qui effectuent des opérations à l'aide des protocoles de services de données REST.

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 l'entité, en plus d'être mise à jour, doit être automatiquement associée aux autres entités existantes du service de données, qui sont associées par des associations to-one à une seule valeur.

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. Le 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 «Extraction d'éléments autres que des entités à l'aide des services de données REST», à la page 282.

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'une liaison entre deux instances d'entité eXtreme Scale peut établir des associations ou en supprimer. Par exemple, si le client établit une association to-one entre une entité `Order(orderId=5000,customer_customerId='IBM')` 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 liaisons, le service de données REST n'analyse que la première. 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') vers 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 réponse : 204 No Content

JSON

- Méthode : PUT
- URI de 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

Concepts associés:

«Opérations avec le service de données REST», à la page 269
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.

«Présentation des services de données REST», à la page 117
Le service de données REST de WebSphere eXtreme Scale est un service HTTP Java qui est compatible avec Microsoft WCF Data Services (ex-ADO.NET Data Services) et qui implémente Open Data Protocol (OData). Microsoft WCF Data Services est compatible avec cette spécification lorsqu'on utilise Visual Studio 2008 SP1 et .NET Framework 3.5 SP1.

Tâches associées:

«Accès aux données avec le service de données REST», à la page 268
Developpez des applications qui effectuent des opérations à l'aide des protocoles de services de données REST.

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

Concepts associés:

«Opérations avec le service de données REST», à la page 269
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.

«Présentation des services de données REST», à la page 117
Le service de données REST de WebSphere eXtreme Scale est un service HTTP Java qui est compatible avec Microsoft WCF Data Services (ex-ADO.NET Data Services) et qui implémente Open Data Protocol (OData). Microsoft WCF Data Services est compatible avec cette spécification lorsqu'on utilise Visual Studio 2008 SP1 et .NET Framework 3.5 SP1.

Tâches associées:

«Accès aux données avec le service de données REST», à la page 268
Developpez des applications qui effectuent des opérations à l'aide des protocoles de services de données REST.

Plug-in et API du système

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.

Gestion des cycles de vie du plug-in

Vous pouvez gérer les cycles de vie de plug-in avec des méthodes spécialisées de chaque plug-in, qui peuvent être appelées à des points fonctionnels déterminés. Les deux méthodes initialize et destroy définissent le cycle de vie des plug-ins qui sont contrôlés par leurs objets *propriétaire*. 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.

Pourquoi et quand exécuter cette tâche

De même, tous les plug-ins peuvent implémenter les interfaces intégrées mixtes adaptées à leur objet propriétaire. Un plug-in ObjectGrid peut implémenter l'ObjectGridPlugin d'interface intégrée mixte facultatif. Un plug-in BackingMap peut implémenter le BackingMapPlugin d'interface intégrée mixte facultatif. Les interfaces intégrées mixtes nécessitent d'implémenter plusieurs méthodes additionnelles en plus des méthodes initialize() et destroy() pour les plug-ins de base. Pour plus d'informations sur ces interfaces, voir la documentation des API.

Lorsque les objets propriétaires sont initialisés, ils définissent des attributs dans le plug-in, puis appellent la méthode d'initialisation de leurs plug-ins détenus. Lors du cycle de destruction des objets propriétaires, la méthode de destruction de plug-ins est, par conséquent, également appelée. Pour plus de détails sur les spécificités des méthodes initialize et destroy et sur les 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'ObjectGrid côté client, et par conséquent, de ses instances de plug-in, dépend d'ObjectGrid côté serveur et des instances de plug-in.

Dans une topologie répartie de ce type, supposons que vous ayez la grille ObjectGrid, `myGrid`, définie dans le fichier `objectGrid.xml` et configurée avec le programme d'écoute ObjectGridEventListener, myObjectGridEventListener. Le fichier `objectGridDeployment.xml` définit la stratégie de déploiement de la grille `myGrid`. Les fichiers `objectGrid.xml` et `objectGridDeployment.xml` sont utilisés pour démarrer les serveurs de conteneur. Au cour du démarrage du serveur de conteneur, l'instance ObjectGrid `myGrid` côté serveur est initialisée. Entre temps, la méthode initialize de l'instance myObjectGridEventListener qui appartient à l'instance myObjectGrid est appelée. Une fois le serveur de conteneur démarré, votre application peut se connecter à l'instance ObjectGrid `myGrid` et obtenir une instance côté client.

Lors de l'obtention de l'instance ObjectGrid myGrid côté client, l'instance `myGrid` côté client passe par son propre cycle de vie d'initialisation et appelle la méthode initialize de sa propre instance myObjectGridEventListener côté client. Cette instance myObjectGridEventListener côté client dépend de l'instance myObjectGridEventListener. Son cycle de vie est contrôlé par son propriétaire qui est l'instance ObjectGrid `myGrid` côté client.

Si votre application se déconnecte ou détruit l'instance ObjectGrid `myGrid`, la méthode destroy qui appartient à l'instance myObjectGridEventListener côté client est appelée automatiquement. Toutefois, ce processus n'a aucun impact sur l'instance myObjectGridEventListener côté serveur. La méthode destroy de l'instance myObjectGridEventListener côté serveur peut uniquement être appelée au cours du cycle de vie destroy de l'instance Object `myGrid` lors de l'arrêt d'un serveur de conteneur. Notamment, lors de l'arrêt d'un serveur de conteneur, les instances ObjectGrid contenues sont détruites et la méthode destroy de tous leurs plug-ins 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 interface BackingMap. En outre, déterminez soigneusement vos configurations pour les plug-ins que vous pouvez écrire en fonction de ces informations de cycle de vie. Consultez les rubriques suivantes pour écrire des plug-ins qui fournissent des événements de gestion de cycle de vie étendu que vous pouvez utiliser pour configurer ou supprimer des ressources dans votre environnement :

Concepts associés:

«Présentation de l'infrastructure OSGi», à la page 37
OSGi définit un système de module dynamique pour Java. La plateforme de service OSGi présente une architecture à couches et elle est conçue pour s'exécuter dans plusieurs profils standard Java. Vous pouvez démarrer les serveurs et les clients WebSphere eXtreme Scale dans un conteneur OSGi.

Information associée:

documentation sur les API

Ecriture du plug-in ObjectGridPlugin

Un plug-in ObjectGridPlugin est une interface intégrée mixte que vous pouvez utiliser pour fournir des événements de gestion de cycle de vie étendu à tous les autres plug-ins ObjectGrid .

Pourquoi et quand exécuter cette tâche

Un plug-in ObjectGrid qui implémente le plug-in ObjectGridPlugin reçoit le jeu étendu des événements de cycle de vie, et peut fournir davantage de contrôle, que vous pouvez utiliser pour configurer ou supprimer des ressources. Dans un conteneur de grille de données partitionnées, il existera une instance ObjectGrid (le plug-in propriétaire) pour chaque partition gérée par le conteneur.Lorsque des partitions individuelles sont supprimées, les ressources utilisées par l'instance ObjectGrid doivent être supprimées également. Par conséquent, vous devrez peut-être fermer ou terminer une ressource, telle qu'un fichier de configuration ouvert ou une unité d'exécution qui est gérée par un plug-in, lorsque la partition propriétaire de cette ressource est supprimée.

L'interface ObjectGridPlugin fournit les méthodes pour définir ou modifier l'état du plug-in, ainsi que des méthodes pour analyser l'état actuel du plug-in. Toutes les méthodes doivent être correctement implémentées, et l'environnement d'exécution WebSphere eXtreme Scale vérifie le comportement de la méthode dans certains cas. Par exemple, après l'appel de la méthode initialize() , l'environnement d'exécution eXtreme Scale appelle la méthode isInitialized() pour s'assurer que la méthode a terminé l'initialisation appropriée.

Procédure

1. Implémentez l'interface ObjectGridPlugin afin que le plug-in ObjectGridPlugin reçoive des notifications sur les événements eXtreme Scale importants. Il existe trois catégories principales de méthodes :

Méthodes de propriété	Fonction
setObjectGrid()	Appelée pour indiquer l'instance ObjectGrid pour laquelle le plug-in est utilisé.
getObjectGrid()	Appelée pour obtenir ou confirmer l'instance ObjectGrid pour laquelle le plug-in est utilisé.

Méthodes d'initialisation	Fonction
initialize()	Appelée pour initialiser le plug-in ObjectGridPlugin.
isInitialized()	Appelée pour obtenir ou confirmer l'initialisation du plug-in.

Méthodes de destruction	Fonction
destroy()	Appelée pour détruire le plug-in ObjectGridPlugin.
isDestroyed()	Appelée pour obtenir ou confirmer la destruction du plug-in.

   Voir la doculentation d'API pour plus d'informations sur ces interfaces.
2. Configurez un plug-in ObjectGridPlugin avec XML. Utilisez la classe com.company.org.MyObjectGridPluginTxCallback qui implémente l'interface TransactionCallback et l'interface ObjectGridPlugin.

   Dans l'exemple de code suivant, le rappel de transaction personnalisée, qui finalement recevra des événements de cycle de vie étendu, est généré et ajouté à un plug-in ObjectGrid.

   Important : L'interface TransactionCallback dispose déjà d'une méthode d'initialisation, une nouvelle méthode initialisation est ajoutée, ainsi que de la méthode destroy et d'autres méthodes ObjectGridPlugin. Chaque méthode est utilisée, et les méthodes d'initialisation exécutent une seule initialisation. Le code XML suivant crée une configuration qui utilise l'interface TransactionCallback étendue.

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.company.org.MyObjectGridPluginTxCallback" />
            <backingMap name="Book"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

Notez que les déclarations de bean précèdent les déclarations backingMap.

3. Indiquez le fichier myGrid.xml au plug-in ObjectGridManager pour faciliter la création de cette configuration.

Tâches associées:

«Ecriture d'un plug-in BackingMapPlugin»
Un plug-in BackingMap implémente l'interface intégrée mixte que vous pouvez utiliser pour recevoir des fonctions étendues pour gérer son cycle de vie.

Information associée:

../com.ibm.websphere.extremescale.javadoc.doc/topics/com/ibm/websphere/objectgrid/management/package-summary.html

Ecriture d'un plug-in BackingMapPlugin

Un plug-in BackingMap implémente l'interface intégrée mixte que vous pouvez utiliser pour recevoir des fonctions étendues pour gérer son cycle de vie.

Pourquoi et quand exécuter cette tâche

Un plug-in BackingMap qui implémente également l'interface BackingMapPlugin recevra automatiquement le groupe étendu d'événement de cycle de vie au cours de sa construction et de son utilisation.

L'interface BackingMapPlugin fournit les méthodes pour définir ou modifier l'état du plug-in, ainsi que des méthodes pour analyser l'état actuel du plug-in.

Toutes les méthodes doivent être correctement implémentées et l'environnement d'exécution WebSphere eXtreme Scale vérifie le comportement de la méthode dans certains cas. Par exemple, après l'appel de la méthode initialize() , l'environnement d'exécution eXtreme Scale appelle la méthode isInitialized() pour s'assurer que la méthode a terminé l'initialisation appropriée.

Procédure

1. Implémentez l'interface BackingMapPlugin afin que le plug-in ObjectGridPlugin reçoive des notifications sur les événements eXtreme Scale importants. Il existe trois principales catégories de méthodes :

Méthodes de propriété	**Fonction**
setBackingMap()	Appelée pour indiquer l'instance BackingMap pour laquelle le plug-in est utilisé.
getBackingMap()	Appelée pour obtenir ou confirmer l'instance BackingMap pour laquelle le plug-in est utilisé.

Méthodes d'initialisation	**Fonction**
initialize()	Appelée pour initialiser le plug-in BackingMapPlugin.

Méthodes d'initialisation	Fonction
isInitialized()	Appelée pour obtenir ou confirmer l'état d'initialisation du plug-in.

Méthodes de destruction	Fonction
destroy()	Appelée pour détruire le BackingMapPlugin.
isDestroyed()	Appelée pour obtenir ou confirmer la destruction du plug-in.

Voir la documentation d'API pour plus d'informations sur ces interfaces.

2. Configurez un plug-in BackingMapPlugin avec XML. Supposons que le nom de classe d'un plug-in Loader eXtreme Scale est la classe com.company.org.MyBackingMapPluginLoader qui implémente l'interface Loader et l'interface BackingMapPlugin.

   Dans l'exemple de code suivant, le rappel de transaction personnalisée, qui recevra finalement des événements de cycle de vie étendu, est généré et ajouté à un plug-in BackingMap.

   Vous pouvez configurer un plug-in BackingMapPlugin en utilisant XML. 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">
            <backingMap name="Book" pluginCollectionRef="myPlugins" />
        </objectGrid>
    </objectGrids>
    <backingMapPluginCollections>
        <backingMapPluginCollection id="myPlugins">
            <bean id="Loader"
                  className="com.company.org.MyBackingMapPluginLoader" />
        </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

3. Fournissez le fichier `myGrid.xml` au plug-in ObjectGridManager pour faciliter la création de cette configuration.

Résultats

L'instance BackingMap qui est créée possède un chargeur qui reçoit les événements de cycle de vie BackingMapPlugin.

Tâches associées:

«Ecriture du plug-in ObjectGridPlugin», à la page 299
Un plug-in ObjectGridPlugin est une interface intégrée mixte que vous pouvez utiliser pour fournir des événements de gestion de cycle de vie étendu à tous les autres plug-ins ObjectGrid .

Information associée:

../com.ibm.websphere.extremescale.javadoc.doc/topics/com/ibm/websphere/objectgrid/management/package-summary.html

Plug-ins pour la réplication multimaître

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 multimaître, les domaines de services de catalogue détectent les collisions automatiquement. Lorsqu'un domaine de services de catalogue détecte une collision, il démarre un arbitre. En général, les collisions sont résolues avec l'arbitre de collision par défaut. Mais une application peut très bien fournir un arbitre personnalisé.

Avant de commencer

- Voir «Planification de plusieurs topologies de centre de données», à la page 99 pour plus d'informations sur la planification et la conception de la topologie de réplication multimaître.
- Voir Configuration de plusieurs topologies de centres de données pour plus d'informations sur la configuration des liaisons entre les domaine de services de catalogue.

Pourquoi et quand exécuter cette tâche

Si un domaine de services de catalogue reçoit une entrée répliquée qui entre en collision avec un enregistrement de collision, l'arbitre par défaut utilise les modifications dans le domaine de services de catalogue le nom vient en premier par ordre alphabétique. Supposons 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 pour correspondre à l'enregistrement dans le domaine A. Les noms de domaine sont convertis en majuscules pour 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 suivante :

   ```
   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 de services de catalogue fournit les informations suivantes pour un arbitre de collisions personnalisé :

   - Version existante de l'enregistrement
   - Version de collision de l'enregistrement
   - Objet Session qui doit servir à créer la version révisée de l'entrée en collision

   La méthode du plug-in retourne un objet qui indique sa décision. La méthode invoquée par le domaine pour appeler le plug-in doit retourner true ou false, false indiquant d'ignorer le conflit. Lorsque la collision est ignorée, la version

locale demeure inchangée et l'arbitre oublie qu'il a vu la version existante. 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`, indiquez 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>
```

Concepts associés:

«Planification de plusieurs topologies de centre de données», à la page 99
En utilisant la réplication asynchrone multimaître, au moins deux grilles de données peuvent devenir des copies exactes de l'une de l'autre. Chaque grille de données est hébergée dans un domaine de services de catalogue indépendant, avec ses propres de service de catalogue, serveurs de conteneur et un nom unique. Avec la réplication asynchrone multimaître, vous pouvez utiliser des liaisons pour connecter un ensemble de domaine de services de catalogue. Les domaine de services de catalogue sont ensuite synchronisés en utilisant la réplication via ces liaisons. Vous pouvez construire quasiment n'importe quelle topologie via la définition de liaisons entre les domaine de services de catalogue.

«Topologies pour la réplication multimaître», à la page 100
Vous disposez de plusieurs options pour choisir la topologie de votre déploiement qui intègre la réplication multimaître.

«Considérations de configuration pour les topologies multimaîtres», à la page 105
Tenez compte des points suivants lorsque vous déterminez l'opportunité et la manière d'utiliser des topologies de réplication multimaîtres.

«Considérations de conception pour la réplication multimaître», à la page 108
Lors de l'implémentation de la réplication multimaître, vous devez tenir compte de divers éléments dans votre conception, tels que l'arbitrage, les liaisons et les performances.

«Remarques sur les chargeurs dans une topologie multimaître», à la page 106
Lorsque vous utilisez des chargeurs dans une topologie multimaître, vous devez envisager les problèmes éventuels de collision et de maintenance des informations de révision. La grille de données conserve les informations de révision sur les éléments de façon à ce que les collisions puissent être détectées lorsque d'autres fragments primaires dans la configuration y écrivent des entrées. Lorsque des entrées sont ajoutées à partir d'un chargeur, ces informations de révision ne sont pas incluses et l'entrée prend une nouvelle révision. Etant donné que la révision de l'entrée semble être une nouvelle insertion, une fausse collision peut se produire si un autre fragment primaire modifie également cet état ou insère les mêmes informations à partir d'un chargeur.

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 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 );
```

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>
```

Plug-ins de sérialisation des objets mis en cache

WebSphere eXtreme Scale utilise plusieurs processus Java pour sérialiser les données en convertissant les instances d'objet Java en octets, puis de nouveau en objets, si nécessaire, pour transférer les données entre les processus client et serveur.

Pour sérialiser les données dans eXtreme Scale, vous pouvez utiliser la sérialisation Java, le pug-in ObjectTransformer ou les plug-ins DataSerializer.

 L'interface ObjectTransformer a été remplacée par les plug-ins DataSerializer que vous pouvez utiliser pour stocker des données arbitraires efficacement dans WebSphere eXtreme Scale afin que les API existantes du produit puissent interagir efficacement avec vos données.

Concepts associés:

Présentation de la sérialisation
Les données sont toujours exprimées, mais pas nécessairement stockées, comme les objets Java dans la grille de données. WebSphere eXtreme Scale utilise plusieurs processus Java pour sérialiser les données en convertissant les instances d'objet Java en octets, puis de nouveau en objets, si nécessaire, pour transférer les données entre les processus client et serveur.

Présentation de la programmation du sérialiseur

Vous pouvez utiliser les plug-ins DataSerializer pour écrire des sérialiseurs optimisés pour stocker les objets Java et d'autres données binaires dans la grille. Le plug-in fournit également des méthodes que vous pouvez utiliser pour rechercher des attributs dans les données binaires sans avoir à augmenter l'ensemble de l'objet données.

Les plug-ins DataSerializer incluent trois plug-ins principaux et plusieurs interfaces intégrées mixtes optionnelles. Le plug-in MapSerializerPlugin contient des métadonnées sur la relation entre une mappe et d'autres mappes. Il contient également un référence à un plug-in KeySerializerPlugin et à un plug-in ValueSerializerPlugin. Les plug-ins de sérialiseur de clé et de valeur contiennent des métadonnées et du code de sérialisation responsable de l'interaction avec les

données key et value d'une mappe. Un plug-in MapSerializerPlugin doit contenir l'un des sérialiseurs de clé et de valeur ou les deux sérialiseurs.

Le plug-in KeySerializerPlugin fournit des méthodes et des métadonnées pour la sérialisation, l'inflation et l'introspection des clés. Le plug-in ValueSerializer fournit des méthodes et des métadonnées pour la sérialisation, l'inflation et l'introspection des clés. Les deux interfaces ont des exigences différentes. Pour plus d'informations sur les méthodes disponibles dans les plug-ins DataSerializer, voir la documentation d'API du module com.ibm.websphere.objectgrid.plugins.io.

Plug-in MapSerializerPlugin
MapSerializerPlugin est le point de plug-in principal vers l'interface BackingMap et il contient deux plug-ins imbriqués : KeySerializerPlugin et ValueSerializerPlugin. Comme eXtreme Scale ne prend pas en charge les plug-ins imbriqués ou connectés, le plug-in accède à ces plug-ins imbriqués de manière artificielle. Lorsque vous utilisez ces plug-ins avec l'infrastructure OSGi, le seul proxy est le plug-in MapSerializerPlugin. Tous les plug-ins imbriqués ne doivent pas être mis en cache dans d'autres plug-ins dépendants, tels que des chargeurs, sauf si ces plug-ins écoutent également les événements de cycle de vie de BackingMap. Ce point est important lors de l'exécution dans une infrastructure OSGi, car les références à ces plug-ins peuvent continuer à être régénérées.

Plug-in KeySerializerPlugin
Le plug-in KeySerializerPlugin étend l'interface DataSerializer et inclut d'autres interfaces mixtes intégrées et les métadonnées qui décrivent la clé. Utilisez ce plug-in pour sérialiser et étendre les objets et les attributs de données de clés.

Plug-in ValueSerializerPlugin
Le plug-in ValueSerializerPlugin étend l'interface DataSerializer, mais n'expose pas d'autres méthodes. Utilisez ce plug-in pour sérialiser et étendre les objets et les attributs de données de valeur.

Interfaces facultatives et intégrées mixtes

Les interfaces facultatives et intégrées mixtes fournissent des fonctions supplémentaires, telles que :

Gestion optimiste des versions
L'interface versionnable permet au plug-in ValueSerializerPlugin de vérifier et de mettre à jour les versions lors de l'utilisation du verrouillage optimiste. Si la gestion optimiste des versions n'est pas incluse et que le verrouillage optimiste est activé, la version est la forme sérialisée complète de la valeur d'objet données.

Routage non-hashCode-based
L'interface partitionnable permet aux implémentations KeySerializerPlugin de router les demandes vers des partitions explicites. Elle est équivalente à l'interface PartitionableKey lorsqu'elle est utilisée avec l'API ObjectMap dans plug-in KeySerializerPlugin. Sans cette fonction, la clé est routée vers la partition enfonction du code de hachage résultant.

Interface UserReadable (toString)
Cette interface permet à toutes les implémentations DataSerializer de fournit une méthode alternative pour afficher les données dans les fichiers journaux et les débogueurs. Avec cette fonction, vous pouvez masquer les données confidentielles, telles que les mots de passe. Si les implémentations DataSerializer n'implémentent pas cette interface,

l'environnement d'exécution pourrait appeler toString() directement sur l'objet ou inclure des représentations alternatives, si nécessaire.

Support d'évolution
L'interface Mergeable peut être implémentée dans les implémentations de plug-in ValueSerializerPlugin pour permettre l'interopérabilité entre plusieurs versions des objets lorsque plusieurs versions DataSerializer différentes mettent à jour les données dans la grille pendant sa durée de vie. Les méthodes Mergable permettent au sérialiseur DataSerializer de conserver des données qu'il peut autrement ne pas comprendre.

Tâches associées:

«Eviter l'extension d'objet pour extraire des attributs depuis des données sérialisées»
Vous pouvez utiliser le plug-in DataSerializer pour éviter l'extension d'objet automatique et extraire manuellement des attributs depuis des données déjà sérialisées.

«Programmation de l'utilisation de l'infrastructure OSGi», à la page 391
Vous pouvez démarrer les serveurs et les clients eXtreme Scale dans un conteneur OSGi qui permet d'ajouter et de mettre à jour dynamiquement les plug-ins eXtreme Scale dans l'environnement d'exécution.

Information associée:

Documentation de l'API DataSerializer

Eviter l'extension d'objet pour extraire des attributs depuis des données sérialisées

Vous pouvez utiliser le plug-in DataSerializer pour éviter l'extension d'objet automatique et extraire manuellement des attributs depuis des données déjà sérialisées.

Pourquoi et quand exécuter cette tâche

Cette rubrique explique comment éviter d'étendre l'ensemble de l'objet pour extraire les attributs. Toutefois, vous pouvez étendre l'ensemble de l'objet en étendant les objets Java à une représentation de type POJO des données. Pour étendre l'objet dans son ensemble, remplacez la dernière ligne de l'exemple de cette rubrique par la ligne de code suivante :

```
Order order = (Order) sa.getMapSerializerPlugin().getValueSerializerPlugin().
inflateDataObject(serValue.getContext(), bufValue);
```

Cette tâche utilise le mode de copie COPY_TO_BYTES_RAW avec les plug-ins MapSerializerPlugin et ValueSerializerPlugin. MapSerializer est le point de plug-in vers l'interface BackingMap. Il contient deux plug-ins imbriqués, KeyDataSerializer et ValueDataSerializer. Comme le produit ne prend pas en charge les plug-ins imbriqués ou connectés, BaseMapSerializer prend en charge les plug-ins imbriqués ou connectés artificiellement. Par conséquent, lorsque vous utilisez ces API dans le conteneur OSGi, MapSerializer est le seul proxy. Tous les plug-ins imbriqués ne doivent pas être mis en cache dans d'autres plug-ins dépendants, tels qu'un chargeur, sauf s'il est également à l'écoute des événements de cycle de vie de BackingMap, de sorte qu'il puisse régénérer ses références sous-jacentes.

Procédure

1. Extrayez l'instance ObjectMap.
2. Définissez le mode de copie COPY_TO_BYTES_RAW.
3. Utilisez la méthode get pour extraire l'objet SerializedValue.

4. Utilisez la méthode SerializedValue.getByteBuffers() pour extraire le format sérialisé de la valeur.
5. Appelez le plug-in ValueSerializerPlugin pour étendre les attributs individuels à partir de la mémoire tampon d'octets.

Exemple

Utiliser l'exemple de code suivant pour extraire des attributs à partir des données sérialisées sans étendre tout l'objet Java.

```
//  BackingMap est configuré avec COPY_TO_BYTES et un MapSerializerPlugin avec un  ValueSerializerPlugin

Session session = objectGrid.getSession();
ObjectMap orderMap = session.getMap("OrderMap");

// Extension automatique à une commande normal de type
POJO order = (Order) orderMap.get(1234);

// Remplacement du mode CopyMode pour extraire les octets.  Ce processus affecte tous les méthodes API à
// partir de ce point
// pendant toute la vie de la session.
// Remarque :  le tableau d'octets a un en-tête eXtreme Scale.
orderMap.setCopyMode(CopyMode.COPY_TO_BYTES_RAW);
SerializedValue serValue = (SerializedValue) orderMap.get(1234);

// Obtention des mémoire tampons d'octets
XsByteBuffer[] bufValue= serValue.getByteBuffers();

// Conversion/obtention du tableau d'octets
Byte[] bytesValue = ByteBufferUtils.asByteArray(bufValue);

// Extraction d'un seul attribut depuis la mémoire tampon d'octets.
String name = (String) sa.getMapSerializerPlugin().getValueSerializerPlugin().inflateDataObjectAttributes
(serValue.getContext(),
 bufValue, new String[]{"name"};
```

Concepts associés:

«Présentation de la programmation du sérialiseur», à la page 309
Vous pouvez utiliser les plug-ins DataSerializer pour écrire des sérialiseurs optimisés pour stocker les objets Java et d'autres données binaires dans la grille. Le plug-in fournit également des méthodes que vous pouvez utiliser pour rechercher des attributs dans les données binaires sans avoir à augmenter l'ensemble de l'objet données.

Présentation de la sérialisation
Les données sont toujours exprimées, mais pas nécessairement stockées, comme les objets Java dans la grille de données. WebSphere eXtreme Scale utilise plusieurs processus Java pour sérialiser les données en convertissant les instances d'objet Java en octets, puis de nouveau en objets, si nécessaire, pour transférer les données entre les processus client et serveur.

Information associée:

Documentation de l'API DataSerializer

Plug-in ObjectTransformer

Le plug-in ObjectTransformer permet de sérialiser, désérialiser et copier des objets du cache afin d'améliorer les performances.

L'interface ObjectTransformer a été remplacée par les plug-ins DataSerializer que vous pouvez utiliser pour stocker efficacement les données arbitraires dans WebSphere eXtreme Scale pour que les API de produit existantes puissent interagir efficacement avec vos données.

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 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);
```

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>
```

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);
```

Plug-in de programme d'écoute d'événement

Vous pouvez utiliser les plug-ins ObjectGridEventListener, MapEventListener, ObjectGridLifecycleListener et BackingMapLifecycleListener afin de configurer des notifications pour divers événements dans le cache 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 319. 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 en utilisant les méthodes addEventListener et removeEventListener dans 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 319.

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 318. 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 et removeMapEventListener dans 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», à la page 318.

Plug-in BackingMapLifecycleListener

Un plug-in BackingMapLifecycleListener fournit les notifications de modifications d'état de cycle de vie qui se produisent dans une BackingMap. L'instance BackingMap passe par un ensemble d'états prédéfinis pendant sa durée de vie.

Ajout et suppression d'instances BackingMapLifecycleListener

Un serveur eXtreme Scale peut posséder plusieurs programmes d'écoute BackingMapLifecycleListener. Ajoutez et supprimez des programmes d'écoute avec les méthodes addMapEventListener et removeMapEventListener dans l'interface BackingMap. Tous les plug-ins BackingMap qui implémentent l'interface BackingMapLifecycleListener sont également automatiquement ajoutés en tant que BackingMapLifecycleListener pour l'instance ObjectGrid dans laquelle ils sont enregistrés. Vous pouvez également enregistrer de manière déclarative les programmes d'écoute BackingMapLifecycleListener avec le fichier descripteur d'ObjectGrid. Pour des exemples, voir Plug-in BackingMapLifecycleListener.

Plug-in ObjectGridLifecycleListener

Un plug-in ObjectGridLifecycleListener fournit les notifications de modifications d'état de cycle de vie qui se produisent dans une instance ObjectGrid. L'instance ObjectGrid passe par un ensemble d'états prédéfinis pendant sa durée de vie.

Ajout et suppression d'instances ObjectGridLifecycleListener

eXtreme Scale peut avoir plusieurs programmes d'écoute ObjectGridLifecycleListener. Ajoutez et supprimez les programmes d'écoute en utilisant les méthodes addEventListener et removeEventListener dans l'interface ObjectGrid. Tous les plug-ins ObjectGrid qui implémentent l'interface ObjectGridLifecycleListener sont également automatiquement ajoutés en tant que ObjectGridLifecycleListener pour l'instance ObjectGrid dans laquelle ils sont enregistrés. Vous pouvez également enregistrer de manière déclarative les programmes d'écoute ObjectGridLifecycleListener avec le fichier descripteur de déploiement ObjectGrid. Pour des exemples, voir Plug-in ObjectGridLifecycleListener.

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 «Présentation des plug-ins», à la page 115. Pour les autres types de plug-in de programme d'écoute, voir «Plug-in de programme d'écoute d'événement», à la page 317.

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 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);
```

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);
```

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 réplique 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.

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 BackingMapLifecycleListener

Un plug-in BackingMapLifecycleListener est averti des événements de modification de l'état de cycle de vie WebSphere eXtreme Scale.

Le plug-in BackingMapLifecycleListener reçoit un événement contenant un objet ObjectGridLifecycleListener.State pour chaque changement d'état de la mappe de sauvegarde. Un plug-in BackingMap qui implémente également l'interface BackingMapLifecycleListener est ajouté automatiquement comme programme d'écoute pour l'instance BackingMap où le plug-in est enregistré.

Présentation

Un plug-in BackingMapLifecycleListener est utile lorsqu'un plug-in BackingMap existant doit effectuer des activités par rapport aux activités d'un plug-in associé. Par exemple, un plug-in de chargeur peut devoir récupérer la configuration depuis un plug-in MapIndexPlugin ou DataSerializer qui coopère.

En implémentant l'interface BackingMapLifecycleListener et en détectant l'événement BackingMapLifecycleListener.State.INITIALIZED, le plug-in TransactionCallback peut détecter l'état des autres plug-ins dans l'instance BackingMap. Le chargeur peut extraire des informations en toute sécurité à partir du plug-in MapIndexPlugin ou DataSerializer coopérant, car BackingMap a l'état INITIALIZED, ce qui signifie que la méthode initialize() de l'autre plug-in a été appelée.

Un BackingMapLifecycleListener peut être ajouté ou supprimé à tout moment avant ou après l'ObjectGrid et ses BackingMaps sont initialisés.

Ecrire un plug-in BackingMapLifecycleListener

Un plug-in BackingMapLifecycleListener doit implémenter l'interface ObjectGridLifecycleListener pour recevoir des notifications sur les événements importants eXtreme Scale. Tout plug-in ObjectGrid peut implémenter l'interface BackingMapLifecycleListener et être automatiquement ajouté en tant que programme d'écoute lorsqu'il est également ajouté à la mappe de sauvegarde.

Pour plus d'informations sur ces interfaces, voir la documentation d'API.

Evénement de cycle de vie et relations des plug-ins

BackingMapLifecycleListener extrait l'état de cycle de vie à partir de l'événement dans la méthode backingMapStateChanged ; par exemple :

```
public void backingMapStateChanged(BackingMap map,
                                   LifecycleEvent event)
throws LifecycleFailedException {
  switch(event.getState()) {
    case INITIALIZED: //Tous les autres plug-ins sont initialisé.
      // Récupérer la référence au plug-in X pour une utilisation à
      // partir de la grille.
      break;
    case DESTROYING:  // Phase de destruction démarrée
      // Supprimer la référence au plug-in X. Elle peut être détruite
      // avant ce plug-in
      break;
  }
}
```

Le tableau suivant décrit les relations entre les événements de cycle de vie envoyés au plug-in BackingMapLifecycleListener et les états d'ObjectGrid et d'autres objets de plug-ins.

BackingMapLifecycleListener.State value	Description
INITIALIZING	La phase d'initialisation d'ObjectGrid démarre. Les plug-ins The BackingMap et BackingMap sont sur le point d'être initialisés.
INITIALIZED	La phase d'initialisation de BackingMap est terminée. Tous les plug-ins BackingMap sont initialisés. L'état INITIALIZED peut réapparaître lorsque les activités de placement de fragment (promotion ou rétrogradation) se produisent.
STARTING	L'instance BackingMap est activée pour l'utiliser comme une instance locale, instance client ou comme instance dans un fragment primaire ou de réplique sur le serveur. Tous les plug-ins ObjectGrid dans l'instance ObjectGrid propriétaire de cette instance BackingMap ont été initialisés L'état STARTING peut réapparaître lorsque les activités de placement de fragment (promotion ou rétrogradation) se produisent.
PRELOAD	L'instance BackingMap est paramétrée sur l'état PRELOAD par l'API StateManager pour le préchargement ou le chargeur configuré précharge les données dans la mappe de sauvegarde.
ONLINE	L'instance BackingMap est prête à fonctionner comme instance locale, instance client ou comme instance dans un fragment primaire ou de réplique sur le serveur. Tous les plug-ins ObjectGrid dans l'instance ObjectGrid propriétaire de cette instance BackingMap ont été initialisés. Cet état stable est l'état standard de BackingMap. L'état ONLINE peut réapparaître lorsque les activités de placement de fragment (promotion ou rétrogradation) se produisent.
QUIESCE	Le travail est arrêté sur BackingMap du fait de l'API StateManager ou d'un autre événement. Aucun nouveau travail n'est autorisé. Les plug-ins arrêtent le travail en cours dès que possible.
OFFLINE	Tout le travail est arrêté sur BackingMap du fait de l'API StateManager ou d'un autre événement. Aucun nouveau travail n'est autorisé.
DESTROYING	L'instance BackingMap démarre la phase de destruction. Les plug-ins BackingMap de l'instance sont sur le point d'être détruits.
DESTROYED	L'instance BackingMap et tous les plug-ins BackingMap ont été détruits.

Configuration d'un plug-in BackingMapLifecycleListener avec XML

Supposons que le programme d'écoute est la classe com.company.org.MyBackingMapLifecycleListener et que cette classe implémente l'interface BackingMapLifecycleListener.

Vous pouvez configurer un plug-in BackingMapLifecycleListener en utilisant XML. Le texte suivant doit se trouver dans le fichier XML de grille d'objet :

```
<?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="BackingMapLifecycleListener"
                  className="com.company.org.MyBackingMapLifecycleListener" />
        </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

Fournissez ce fichier au plug-in ObjectGridManager pour faciliter la création de cette configuration. L'instance BackingMap qui est créée possède un programme d'écoute BackingMapLifecycleListener défini sur myGrid ObjectGrid.

A l'instar de BackingMapLifecycleListener, les autres plug-ins BackingMap, tels que Loader ou MapIndexPlugin, que vous définissez en utilisant XML qui implémente aussi l'interface BackingMapLifecycleListener, seront ajoutés automatiquement comme programmes d'écoute de cycle de vie.

Référence associée:

«Plug-in ObjectGridLifecycleListener»
Un plug-in ObjectGridLifecycleListener reçoit une notification de cycle de vie WebSphere eXtreme Scale, les événements de changement d'état pour la grille de données.

Plug-in ObjectGridLifecycleListener

Un plug-in ObjectGridLifecycleListener reçoit une notification de cycle de vie WebSphere eXtreme Scale, les événements de changement d'état pour la grille de données.

Le plug-in ObjectGridLifecycleListener reçoit un événement contenant un objet ObjectGridLifecycleListener.State pour chaque changement d'état de d'ObjectGrid. Un plug-in qui implémente également l'interface ObjectGridLifecycleListener est ajouté automatiquement comme programme d'écoute pour l'instance ObjectGrid où le plug-in est enregistré.

Présentation

Un plug-in ObjectGridLifecycleListener est utile lorsqu'un plug-in ObjectGrid existant doit effectuer des activités par rapport aux activités dans un plug-in associé. Un plug-in TransactionCallback peut devoir récupérer la configuration depuis un plug-in ObjectGridEventListener ou ShardListener qui coopère.

En implémentant l'interface ObjectGridLifecycleListener et en détectant l'événement ObjectGridLifecycleListener.State.INITIALIZED, le plug-in TransactionCallback peut détecter l'état des autres plug-ins dans l'instance ObjectGrid. Le plug-in TransactionCallback peut extraire des informations en toute sécurité à partir du plug-in ObjectGridEventListener ou ShardListener coopérant, car ObjectGrid a l'état INITIALIZED, ce qui signifie que la méthode initialize() de l'autre plug-in a été appelée.

Vous pouvez ajouter un plug-in ObjectGridLifecycleListener à tout moment, avant ou après l'initialisation d'ObjectGrid.

Ecrire un plug-in ObjectGridLifecycleListener

Un plug-in ObjectGridLifecycleListener doit implémenter l'interface ObjectGridLifecycleListener pour recevoir des notifications sur les événements importants eXtreme Scale. Tout plug-in ObjectGrid peut implémenter l'interface ObjectGridLifecycleListener et être automatiquement ajouté en tant que programme d'écoute lorsqu'il est également ajouté à l'ObjectGrid.

Pour plus d'informations sur ces interfaces, voir la documentation d'API.

Evénement de cycle de vie et relations des plug-ins

ObjectGridLifecycleListener extrait l'état de cycle de vie à partir de l'événement dans la méthode objectgridStateChanged ; par exemple :

```
public void objectGridStateChanged(ObjectGrid grid,
                                   LifecycleEvent event)
throws LifecycleFailedException {
  switch(event.getState()) {
    case INITIALIZED: // Tous les autres plug-ins sont initialisés.
      // Récupérer la référence au plug-in X pour une utilisation à
      // partir de la grille.
      break;
    case DESTROYING:  // Phase de destruction démarrée
      // Supprimer la référence au plug-in X. Elle peut être détruite
      // avant ce plug-in
      break;
  }
```

Le tableau suivant décrit les relations entre les événements de cycle de vie envoyés à un ObjectGridLifecycleListener et les états de l'ObjectGrid et d'autres objets de plug-ins.

ObjectGridLifecycleListener.State value	Description
INITIALIZING	La phase d'initialisation d'ObjectGrid démarre. L'ObjectGrid et les plug-ins ObjectGrid sont sur le point d'être initialisés.
INITIALIZED	La phase d'initialisation d'ObjectGrid est terminée. Tous les plug-ins ObjectGrid sont initialisés. L'état INITIALIZED peut réapparaître lorsque les activités de placement de fragment (promotion ou rétrogradation) se produisent. Tous les plug-in BackingMap dans les instances de BackingMap appartenant à cette instance ObjectGrid ont été initialisés.
STARTING	L'instance ObjectGrid est activée pour l'utiliser comme une instance locale, instance client ou instance dans un fragment primaire ou de réplique sur le serveur. L'état STARTING peut réapparaître lorsque les activités de placement de fragment (promotion ou rétrogradation) se produisent.
PRELOAD	L'instance ObjectGrid est paramétrée sur l'état PRELOAD par l'API ou une autre configuration StateManager.
ONLINE	L'instance ObjectGrid est prête à fonctionner comme instance locale, instance client ou instance dans un fragment primaire ou de réplique sur le serveur. Cet état stable est l'état standard d'ObjectGrid. L'état ONLINE peut réapparaître lorsque les activités de placement de fragment (promotion ou rétrogradation) se produisent.
QUIESCE	Le travail est arrêté sur ObjectGridWork du fait de l'API StateManager ou d'un autre événement. Aucun nouveau travail n'est autorisé. Arrêtez tout travail existant dès que possible.
OFFLINE	Le travail est arrêté sur ObjectGridWork du fait de l'API StateManager ou d'un autre événement. Aucun nouveau travail n'est autorisé.
DESTROYING	L'instance ObjectGrid démarre la phase de destruction. Les plug-ins ObjectGrid de l'instance sont sur le point d'être détruits. Au cours de la phase de suppression, toutes les instances de BackingMap appartenant à cette instance ObjectGrid sont également supprimées.
DESTROYED	L'instance ObjectGrid, ses instances de BackingMap et tous les plug-ins ObjectGrid ont été détruits.

Configuration d'un plug-in ObjectGridLifecycleListener avec XML

Supposons que le nom de la classe de l'écouteur d'événement eXtreme Scale soit la classe com.company.org.MyObjectGridLifecycleListener. Cette classe implémente l'interface ObjectGridLifecycleListener.

Vous pouvez configurer un plug-in ObjectGridPlugin avec XML. Le code XML suivant crée une configuration en utilisant le plug-in ObjectGridLifecycleListener. Le texte suivant doit se trouver dans le fichier XML de grille d'objet :

```
<?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="ObjectGridLifecycleListener"
                  className="com.company.org.MyObjectGridLifecycleListener" />
            <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.

Comme le plug-in ObjectGridLifecycleListener enregistré dans l'exemple précédent, d'autres plug-ins ObjectGrid, tels que CollisionArbiter ou TransactionCallback, que vous pouvez définir en utilisant XML, qui implémentent l'interface ObjectGridLifecycleListener, seront ajoutés automatiquement comme écouteurs de cycle de vie.

Référence associée:

«Plug-in BackingMapLifecycleListener», à la page 322
Un plug-in BackingMapLifecycleListener est averti des événements de modification de l'état de cycle de vie WebSphere eXtreme Scale.

Plug-ins d'indexation des données

Le plug-in HashIndex intégré, la classe com.ibm.websphere.objectgrid.plugins.index.HashIndex, est un plug-in MapIndexPlugin que vous pouvez ajouter à un mappe de sauvegarde BackingMap pour générer des index statiques ou dynamiques. Cette classe prend en charge à la fois les interfaces MapIndex et MapRangeIndex. La définition et l'implémentation d'index peuvent considérablement améliorer les performances des requêtes.

Tâches associées:

«Configuration du plug-in HashIndex»
Vous pouvez configurer le plug-in HashIndex intégré, la classe com.ibm.websphere.objectgrid.plugins.index.HashIndex avec un fichier XML à l'aide d'un programme ou d'une annotation d'entité dans une mappe d'entité.

«Accès aux données avec des index (API Index)», à la page 144
Utilisez l'indexation pour améliorer l'accès aux données.

Référence associée:

«Attributs du plug-in HashIndex», à la page 329
Vous pouvez utiliser les attributs suivants pour configurer le plug-in HashIndex Ces attributs définissent des propriétés comme si vous utilisiez un attribut ou un index HashIndex composite ou que l'indexation de plage était activée.

Configuration du plug-in HashIndex

Vous pouvez configurer le plug-in HashIndex intégré, la classe com.ibm.websphere.objectgrid.plugins.index.HashIndex avec un fichier XML à l'aide d'un programme ou d'une annotation d'entité dans une mappe d'entité.

Pourquoi et quand exécuter cette tâche

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, la valeur de la propriété **nomAttribut** est une liste d'attributs séparés par une virgule. Par exemple, la classe de valeur Address a trois attributs : city, state et zipcode. Un index composite peut être défini avec la valeur de la 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.

Procédure

- Configurez un index composite dans le fichier XML de descripteur d'ObjectGrid.

  Utilisez l'élément backingMapPluginCollections pour définir le plug-in :

  ```
  <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>
  ```

- Configurez un index composite à l'aide d'un programme.

  L'exemple de code suivant crée le même index composite :

  ```
  HashIndex mapIndex = new HashIndex();
   mapIndex.setName("Address.CityStateZip");
   mapIndex.setAttributeName(("city,state,zipcode"));
   mapIndex.setRangeIndex(true);

   BackingMap bm = objectGrid.defineMap("mymap");
   bm.addMapIndexPlugin(mapIndex);
  ```

- Configurez un index composite avec des notations d'entité.

  Si vous utilisez des mappes d'entité, vous pouvez utiliser une approche d'annotation pour définir un index composite. Vous pouvez définir une liste de CompositeIndex dans l'annotation CompositeIndex au niveau de la classe d'entité. CompositeIndex a un nom et une propriété **attributeNames**. Chaque CompositeIndex est associé à une instance HashIndex appliquées à la mappe de sauvegarde associées à 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 la mappe d'entité et de sauvegarde. Si aucun nom n'est indiqué, un nom généré est utilisé. La propriété **nomAttribut** est utilisée pour remplir l'attribut HashIndex avec la liste d'attributs séparés par une virgule. 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 getter de la propriété est getStreet.

Exemple : Ajout de HashIndex à BackingMap

Dans l'exemple suivant, vous configurez le plug-in HashIndex en ajoutant des plug-ins d'index au fichier 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 intégrée HashIndex est utilisée comme plug-in d'indexation. HashIndex prend en charge les propriétés que les utilisateurs peuvent configurer, telles que Name, RangeIndex et AttributeName.

- La propriété **Name** a la valeur `CODE`, une chaîne qui identifie ce plug-in d'index. La valeur de la propriété **Name** doit être unique dans la portée de la mappe de sauvegarde. Le nom peut être utilisé pour extraire l'objet d'index par nom à partir de l'instance ObjectMap pour la mappe de sauvegarde.
- La propriété **RangeIndex** a la valeur `true`, ce qui signifie que l'application peut transtyper l'objet d'index extrait vers l'interface MapRangeIndex. Si la propriété RangeIndex a la valeur `false`, l'application peut transtyper uniquement l'objet d'index extrait 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 repose sur la requête, la propriété **RangeIndex** doit avoir la valeur `true` dans les index à un seul attribut. Dans le cas d'un index de relations ou d'un index composite, la propriété **RangeIndex** doit avoir la valeur `false`.
- La propriété **AttributeName** a la valeur `employeeCode`, ce qui implique que l'attribut employeeCode de l'objet en cache est utilisé pour créer un index à un seul attribut. Si une application doit rechercher des objets mis en cache avec plusieurs attributs, la propriété **AttributeName** peut être affectée d'une liste d'attributs séparés par une virgule pour produire un index composite.

Pour résumer, l'exemple précédent définit un HashIndex de plage à attribut unique. Il s'agit d'un index HashIndex à un seul attribut, car la valeur de la propriété **AttributeName** est `employeeCode` qui inclut un seul nom d'attribut. Il s'agit également d'une plage HashIndex.

Concepts associés:

«Plug-ins d'indexation des données», à la page 326
Le plug-in HashIndex intégré, la classe com.ibm.websphere.objectgrid.plugins.index.HashIndex, est un plug-in MapIndexPlugin que vous pouvez ajouter à un mappe de sauvegarde BackingMap pour générer des index statiques ou dynamiques. Cette classe prend en charge à la fois les interfaces MapIndex et MapRangeIndex. La définition et l'implémentation d'index peuvent considérablement améliorer les performances des requêtes.

«Plug-in d'indexation personnalisée des objets en cache», à la page 332
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.

«Utilisation d'un index composite», à la page 335
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.

«Indexation», à la page 97
Utilisez le plug-in MapIndexPlugin pour générer un ou plusieurs index dans une mappe BackingMap pour prendre en charge l'accès aux données ne correspondant pas à une clé.

Référence associée:

«Attributs du plug-in HashIndex»
Vous pouvez utiliser les attributs suivants pour configurer le plug-in HashIndex Ces attributs définissent des propriétés comme si vous utilisiez un attribut ou un index HashIndex composite ou que l'indexation de plage était activée.

Attributs du plug-in HashIndex :

Vous pouvez utiliser les attributs suivants pour configurer le plug-in HashIndex Ces attributs définissent des propriétés comme si vous utilisiez un attribut ou un index HashIndex composite ou que l'indexation de plage était activée.

Attributs

Name Spécifie le nom de l'index. Le nom doit être unique pour chaque mappe. Ce nom est utilisé pour extraire l'objet d'index de l'instance de mappe d'objet pour la mappe de sauvegarde.

AttributeName
Spécifie à l'index les noms des attributs, séparés par des virgules. Pour les index d'accès par champ, les noms d'attributs sont équivalents aux noms des champs. Pour les index d'accès par propriété, les noms d'attributs sont les noms de propriétés compatibles JavaBean. Si un seul nom d'attribut existe, HashIndex est un index d'attribut unique. Si cet attribut est une relation, il est également un index de relation. Si les noms d'attributs recouvrent plusieurs attributs, l'index HashIndex sera un index composite.

FieldAccessAttribute
Utilisé pour les mappes de non-entités. Si la valeur est égale à `true`, l'accès à l'objet s'effectue par les champs. S'il n'est pas défini ou a la valeur `false`, la méthode getter de l'attribut est utilisée pour accéder aux données.

POJOKeyIndex
Utilisé pour les mappes de non-entités. Si `true`, l'index introspecte l'objet dans la partie clé de la mappe. Ce paramètre est utile lorsque la clé est une clé composite et que la valeur ne contient pas la clé intégrée. Si l'attribut n'est pas défini ou que sa valeur est `false`, l'index introspecte l'objet dans la partie clé de la mappe.

RangeIndex
Si `true`, l'indexation de plage est activée et l'application peut transtyper l'objet d'index extrait vers l'interface MapRangeIndex. Si la propriété **`RangeIndex`** a la valeur `false`, l'application peut transtyper uniquement l'objet d'index extrait vers l'interface MapIndex.

HashIndex à attribut unique ou HashIndex composite ?

Lorsque la propriété **`AttributeName`** de l'index HashIndex contient plusieurs noms d'attributs, l'index HashIndex est un index composite. Dans le cas contraire, si l'index inclut un seul nom d'attribut, il s'agit d'un index à attribut unique. Par exemple, la valeur de la propriété AttributeName d'un HashIndex composite pourra être `city,state,zipcode`. L'index contient trois attributs séparés par des virgules. Si la valeur de la propriété **`AttributeName`** est uniquement `zipcode` qui n'a qu'un seul attribut, il s'agit d'un index HashIndex à attribut unique.

Les index HashIndex composites constituent un mode efficace de consultation des objets mis en cache lorsque les critères de recherche impliquent plusieurs attributs. Toutefois, ils ne prennent pas en charge les index de plage et la propriété RangeIndex doit avoir la valeur `false`.

Reportez-vous à la rubrique relative aux index composites dans le *guide d'administration*.

HashIndex de relation

Si l'attribut indexé d'un HashIndex à attribut unique est une relation, que ce soit à valeur unique ou à valeurs multiples, l'index HashIndex est un HashIndex de relation. Pour l'index HashIndex de relation, la propriété RangeIndex de l'index HashIndex doit être définie sur false.

Les HashIndex de relation peuvent accélérer l'exécution des requêtes qui exploitent des références cycliques ou qui utilisent les filtres de requête IS NULL, IS EMPTY, SIZE et MEMBER OF. Pour plus d'informations, voir «Optimisation des requêtes à l'aide d'index», à la page 454les informations sur l'optimisation de requête avec des index dans *Guide de programmation*.

HashIndex de clés

Dans le cas de mappes de non-entité, lorsque la propriété **`POJOKeyIndex`** de HashIndex a la valeur `true`, l'index HashIndex est un index HashIndex de clés et la partie clé de l'entrée est utilisée pour l'indexation. Lorsque la propriété AttributeName du HashIndex n'est pas spécifiée, la totalité de la clé est indexée. Sinon, le HashIndex de clés ne peut être qu'un HashIndex à attribut unique.

Par exemple, l'ajout de la propriété suivante dans l'exemple précédent transforme le HashIndex en HashIndex de clés car la propriété POJOKeyIndex a la valeur `true`.

```
<property name="POJOKeyIndex" type="boolean" value="true"
description="indicates if POJO key HashIndex" />
```

Dans l'exemple d'index de clé précédent, comme la propriété **AttributeName** a la valeur employeeCode, l'attribut indexé est la zone **employeeCode** de la partie clé de l'entrée de mappe. Si vous souhaitez générer un index de clé sur la totalité de la partie clé de l'entrée de mappe, supprimez la propriété **AttributeName**.

HashIndex de plage

Lorsque la propriété RangeIndex de HashIndex a la valeur true, l'index HashIndex est un index de plage et il peut prendre en charge l'interface MapRangeIndex. Une implémentation 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). Pour un index à attribut unique, la propriété **RangeIndex** ne peut avoir la valeur true que si l'attribut indexé est de type Comparable. Si l'index à attribut unique est utilisé par la requête, la propriété RangeIndex doit avoir la valeur true et l'attribut indexé doit être de type Comparable. Pour l'index HashIndex de relation et l'index HashIndex composite, la propriété RangeIndex doit avoir la valeur false.

L'exemple qui précède est un index HashIndex de plage car la propriété RangeIndex a la valeur true.

Le tableau qui suit récapitule l'utilisation de l'index de plage.

Tableau 5. Prise en charge de l'index de plage. Indique si les types de HashIndex prennent ou non en charge l'index de plage.

Type de HashIndex	Prise en charge de l'index de plage
HashIndex à attribut unique : la clé ou l'attribut indexé est de type Comparable	Oui
HashIndex à attribut unique : la clé ou l'attribut indexé n'est pas de type Comparable	Non
HashIndex composite	Non
HashIndex de relation	Non

Optimisation des requêtes avec des plug-in HashIndex

La définition d'index peut améliorer sensiblement les performances des requêtes. Les requêtes WebSphere eXtreme Scale peuvent utiliser des plug-ins HashIndex pour améliorer les performances des requêtes. Même si l'utilisation des index peut considérablement améliorer les performances des requêtes, elle n'est pas sans impact sur les performances des opérations transactionnelles des mappes.

Concepts associés:

«Plug-ins d'indexation des données», à la page 326
Le plug-in HashIndex intégré, la classe com.ibm.websphere.objectgrid.plugins.index.HashIndex, est un plug-in MapIndexPlugin que vous pouvez ajouter à un mappe de sauvegarde BackingMap pour générer des index statiques ou dynamiques. Cette classe prend en charge à la fois les interfaces MapIndex et MapRangeIndex. La définition et l'implémentation d'index peuvent considérablement améliorer les performances des requêtes.

«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.

«Utilisation d'un index composite», à la page 335
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.

«Indexation», à la page 97
Utilisez le plug-in MapIndexPlugin pour générer un ou plusieurs index dans une mappe BackingMap pour prendre en charge l'accès aux données ne correspondant pas à une clé.

Tâches associées:

«Configuration du plug-in HashIndex», à la page 327
Vous pouvez configurer le plug-in HashIndex intégré, la classe com.ibm.websphere.objectgrid.plugins.index.HashIndex avec un fichier XML à l'aide d'un programme ou d'une annotation d'entité dans une mappe d'entité.

«Accès aux données avec des index (API Index)», à la page 144
Utilisez l'indexation pour améliorer l'accès aux données.

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.

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);
        }
    }
```

Tâches associées:

«Configuration du plug-in HashIndex», à la page 327
Vous pouvez configurer le plug-in HashIndex intégré, la classe com.ibm.websphere.objectgrid.plugins.index.HashIndex avec un fichier XML à l'aide d'un programme ou d'une annotation d'entité dans une mappe d'entité.

«Accès aux données avec des index (API Index)», à la page 144
Utilisez l'indexation pour améliorer l'accès aux données.

Référence associée:

«Attributs du plug-in HashIndex», à la page 329
Vous pouvez utiliser les attributs suivants pour configurer le plug-in HashIndex Ces attributs définissent des propriétés comme si vous utilisiez un attribut ou un index HashIndex composite ou que l'indexation de plage était activée.

Utilisation d'un index 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'entité uniquement pour les mappes d'entités.

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.

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>
```

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.

Tâches associées:

«Configuration du plug-in HashIndex», à la page 327
Vous pouvez configurer le plug-in HashIndex intégré, la classe com.ibm.websphere.objectgrid.plugins.index.HashIndex avec un fichier XML à l'aide d'un programme ou d'une annotation d'entité dans une mappe d'entité.

«Accès aux données avec des index (API Index)», à la page 144
Utilisez l'indexation pour améliorer l'accès aux données.

Référence associée:

«Attributs du plug-in HashIndex», à la page 329
Vous pouvez utiliser les attributs suivants pour configurer le plug-in HashIndex Ces attributs définissent des propriétés comme si vous utilisiez un attribut ou un index HashIndex composite ou que l'indexation de plage était activée.

Plug-ins pour communiquer avec les bases de données

Avec un plug-in Loader, une mappe ObjectGrid peut se comporter comme un cache pour les données généralement conservées dans un magasin persistant 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).

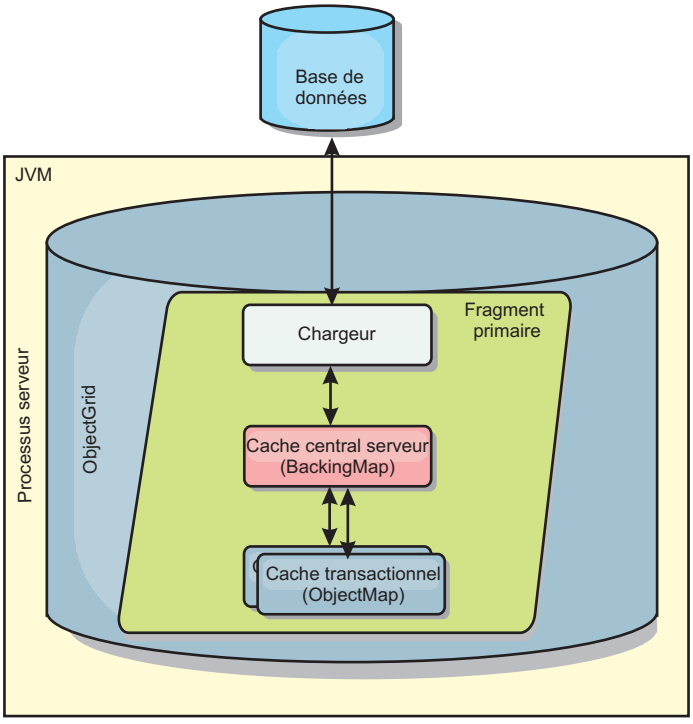


Figure 26. Chargeur

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.

Chargeurs dans les configurations multimaîtres

Pour des remarques sur l'utilisation de chargeurs dans les configurations multimaîtres, voir «Remarques sur les chargeurs dans une topologie multimaître», à la page 106.

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 les mêmes propriétés Loader de niveau d'isolement. Vous devez spécifier le nom de classe pour le chargeur, le nom de la base de données et les détails de connexion ainsi que les propriétés de niveau d'isolement. Vous pouvez utiliser la même structure XML si vous utilisez uniquement un préchargeur en spécifiant le nom de classe du préchargeur et non pas un nom de classe de chargeur complet :

```
<?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>
```

Référence associée:

«Remarques sur la programmation de chargeurs JPA», à la page 365
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.

Configuration des chargeurs de base de données

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).

Remarques sur le préchargement

Les chargeurs 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 une présentation de la manière dont eXtreme Scale interagit avec un chargeur, voir «Cache en ligne», à la page 83.

Chaque mappe de sauvegarde dispose d'un attribut booléen preloadMode qui est défini pour indiquer si le préchargement d'une mappe s'exécute de manière asynchrone. Par défaut, l'attribut preloadMode est défini sur false, ce qui signifie que l'initialisation de la mappe de sauvegarde ne s'effectue que si le préchargement de la mappe est terminé. Par exemple, l'initialisation de la mappe de sauvegarde n'est pas complète tant que la méthode preloadMap s'exécute. Si la méthode preloadMap reconnaît une grande quantité de données en provenance de son

programme d'arrière plan et les charge dans la mappe, son exécution peut prendre un certain temps. Dans ce cas, vous pouvez configurer l'utilisation du préchargement asynchrone pour une mappe de sauvegarde donnée en définissant l'attribut preloadMode sur true. Ce paramètre amène le code d'initialisation de la mappe de sauvegarde à démarrer une unité d'exécution qui appelle la méthode preloadMap pour permettre d'initialiser une mappe de sauvegarde lorsque le préchargement de la mappe est en cours.

Dans un scénario eXtreme Scale réparti, l'un des motifs de préchargement est le préchargement client. Dans le modèle de préchargement client, un client eXtreme Scale est responsable de l'extraction des données à partir du programme d'arrière-plan et de l'insertion des données dans le serveur de conteneur réparti à l'aide d'agents DataGrid. En outre, le préchargement client peut être exécuté dans la méthode Loader.preloadMap dans une seule et unique partition spécifique. Dans ce cas, le chargement asynchrone des données dans la grille devient crucial. Si le préchargement client était exécuté dans la même unité d'exécution, la mappe de sauvegarde ne serait jamais initialisée, de sorte que la partition dans laquelle elle réside ne passerait jamais au statut ONLINE. Par conséquent, le client eXtreme Scale ne pourrait pas envoyer la demande à la partition, ce qui provoquerait une exception au final.

Si un client eXtreme Scale est utilisé dans la méthode preloadMap, vous devez affecter la valeur true à l'attribut **preloadMode**. Il est également possible de lancer une unité d'exécution dans le code de préchargement du client.

Le fragment de code ci-dessous illustre comment l'attribut preloadMode est défini pour activer le préchargement asynchrone :

```
BackingMap bm = og.defineMap( "map1" );
bm.setPreloadMode( true );
```

L'attribut preloadMode peut également être défini à l'aide d'un fichier XML tel qu'illustré dans l'exemple suivant :

```
<backingMap name="map1" preloadMode="true" pluginCollectionRef="map1"
 lockStrategy="OPTIMISTIC" />
```

Objet TxID et utilisation de l'interface TransactionCallback

Les méthodes get et batchUpdate sur l'interface Loader reçoivent un objet TxID qui représente la transaction Session nécessitant d'exécuter l'opération get ou batchUpdate. Les méthodes get et batchUpdate peuvent être appelées plusieurs fois par transaction. Par conséquent, les objets inclus dans l'étendue de la transaction nécessaires au chargeur sont généralement conservés dans un emplacement de l'objet TxID. Un chargeur JDBC (Java Database Connectivity) est utilisé pour illustrer comment un chargeur utilise l'objet TxID et les interfaces TransactionCallback.

Plusieurs mappes ObjectGrid peuvent être stockées dans la même base de données. Chaque mappe possède son propre chargeur, et chaque chargeur peut devoir se connecter à la même base de données. Lorsque les chargeurs se connectent à la base de données, ils doivent utiliser la même connexion JDBC pour valider les modifications dans chaque table de la même transaction de base de données. En règle générale, la personne qui écrit l'implémentation Loader écrit également l'implémentation de TransactionCallback. La meilleure méthodes est lorsque l'interface TransactionCallback est étendue pour ajouter des méthodes requises par le chargeur pour l'obtention d'une connexion à la base de données et pour mettre

en cache les instructions préparées. Cette méthode devient une évidence lorsque vous constatez comment les interfaces TransactionCallback et TxID sont utilisées par le chargeur.

Par exemple, le chargeur peut nécessiter l'extension de l'interfaceTransactionCallback de la manière suivante :

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
public interface MyTransactionCallback extends TransactionCallback
{
    Connection getAutoCommitConnection(TxID tx, String databaseName) throws SQLException;
    Connection getConnection(TxID tx, String databaseName, int isolationLevel ) throws SQLException;
    PreparedStatement getPreparedStatement(TxID tx, Connection conn, String tableName, String sql)
   throws SQLException;
    Collection getPreparedStatementCollection( TxID tx, Connection conn,  String tableName );
}
```

A l'aide de ces nouvelles méthodes, les méthodes Loader get et batchUpdate peuvent obtenir une connexion comme suit :

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
private Connection getConnection(TxID tx, int isolationLevel)
{
    Connection conn = ivTcb.getConnection(tx, databaseName, isolationLevel );
    return conn;
}
```

Dans l'exemple précédent et dans les exemples qui suivent, ivTcb et ivOcb sont des variables d'instance du chargeur qui ont été initialisées tel que décrit dans la section Remarques sur le préchargement. La variable ivTcb est une référence à l'instance MyTransactionCallback et la variable ivOcb est une référence à l'instance MyOptimisticCallback. La variable databaseName est une variable d'instance du chargeur qui a été définie en tant que propriété du chargeur lors de l'initialisation de la mappe de sauvegarde. L'argument isolationLevel est l'une des constantes de la connexion JDBC définies pour les différents niveaux d'isolement pris en charge par JDBC. Si le chargeur utilise une implémentation optimiste, la méthode get utilise généralement une connexion JDBC de validation automatique pour extraire les données de la base de données. Dans ce cas, il se peut que le chargeur possède une méthode getAutoCommitConnection implémentée de la façon suivante :

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
private Connection getAutoCommitConnection(TxID tx)
{
    Connection conn = ivTcb.getAutoCommitConnection(tx, databaseName);
    return conn;
}
```

Rappelez-vous que la méthode batchUpdate comporte l'instruction switch suivante :

```
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;
}
```

Chacun méthode buildBatchSQL utilise l'interface MyTransactionCallback pour obtenir une instruction préparée. Le fragment de code ci-dessous présente la méthode buildBatchSQLUpdate créant une instruction SQL update pour mettre à jour une entrée EmployeeRecord et l'ajouter pour la mise à jour par lots :

```
private void buildBatchSQLUpdate( TxID tx, Object key, Object value,
 Connection conn )
throws SQLException, LoaderException
{
    String sql = "update EMPLOYEE set LASTNAME = ?, FIRSTNAME = ?, DEPTNO = ?,
    SEQNO = ?, MGRNO = ? where EMPNO = ?";
    PreparedStatement sqlUpdate = ivTcb.getPreparedStatement( tx, conn,
   "employee", sql );
    EmployeeRecord emp = (EmployeeRecord) value;
    sqlUpdate.setString(1, emp.getLastName());
    sqlUpdate.setString(2, emp.getFirstName());
    sqlUpdate.setString(3, emp.getDepartmentName());
    sqlUpdate.setLong(4, emp.getSequenceNumber());
    sqlUpdate.setInt(5, emp.getManagerNumber());
    sqlUpdate.setInt(6, key);
    sqlUpdate.addBatch();
}
```

Une fois que la boucle batchUpdate a créé toutes les instructions préparées, elle appelle la méthode getPreparedStatementCollection. Cette dernière est implémentée comme suit :

```
private Collection getPreparedStatementCollection( TxID tx, Connection conn )
{
    return ( ivTcb.getPreparedStatementCollection( tx, conn, "employee" ) );
}
```

Lorsque l'application appelle la méthode commit sur la session, le code de session appelle la méthode commit sur la méthode TransactionCallback après avoir envoyé toutes les modifications apportées par la transaction vers le chargeur pour chaque mappe qui a été modifiée par la transaction. Etant donné que tous les chargeurs ont utilisé la méthode MyTransactionCallback pour établir la connexion et les instructions préparées requises, la méthode TransactionCallback identifie la connexion à utiliser pour demander la validation des modifications par la programme d'arrière plan. L'extension de l'interface TransactionCallback avec des méthodes requises par chaque chargeur présente donc les avantages suivants :

- L'objet TransactionCallback incorpore l'utilisation des emplacements TxID pour les données incluses dans l'étendue de la transaction et le chargeur n'a pas besoin des informations sur les emplacements TxID. Le chargeur doit uniquement reconnaître les méthodes ajoutées à l'objet TransactionCallback utilisant l'interface MyTransactionCallback pour les fonctions de prise en charge requises par le chargeur.
- L'objet TransactionCallback peut permettre le partage de la connexion entre chaque chargeur qui établit la connexion avec le même programme d'arrière plan pour éviter un protocole commit à deux phases.
- L'objet TransactionCallback peut faire en sorte que la connexion au programme d'arrière plan est terminée via une instruction commit ou rollback appelée lors de la connexion si nécessaire.
- TransactionCallback garantit le nettoyage des ressources de la base de données à la fin d'une transaction.

- TransactionCallback se masque s'il établit une connexion gérée à partir d'un environnement géré tel que WebSphere Application Server ou autre serveur d'applications compatible Java 2 Platform, Enterprise Edition (J2EE). Cet avantage permet l'utilisation du même code de chargeur dans un environnement géré et dans un environnement non géré. Seul le plug-in TransactionCallback doit être modifié.
- Pour plus de détails sur l'utilisation des emplacements TxID par l'implémentation TransactionCallback pour les données incluses dans l'étendue de la transaction, reportez-vous à la rubrique Plug-in TransactionCallback.

OptimisticCallback

Comme mentionné précédemment, le chargeur peut utiliser une approche optimiste pour le contrôle des accès simultanés. Dans ce cas, l'exemple de la méthode buildBatchSQLUpdate doit être légèrement modifié pour l'implémentation d'une approche optimiste. Il existe différentes façons pour mettre en oeuvre une approche optimiste. La façon la plus courante consiste à disposer d'une colonne d'horodatage ou d'une colonne de compteur de numéros de séquence pour la gestion des versions de mise à jour de la ligne. Supposons que la table employee comporte une colonne de numéros de séquence qui s'incrémentent à chaque mise à jour de la ligne. Vous modifiez ensuite la signature de la méthode buildBatchSQLUpdate pour qu'elle soit transmise à l'objet LogElement et non à la paire clé-valeur. Elle doit également utiliser l'objet OptimisticCallback relié à la mappe de sauvegarde pour l'obtention de l'objet de version initiale et la mise à jour de l'objet de version. L'exemple suivant est celui d'une méthode buildBatchSQLUpdate modifiée qui utilise la variable d'instance ivOcb initialisée tel que décrit dans la section preloadMap :

```
modified batch-update method code example
private void buildBatchSQLUpdate( TxID tx, LogElement le, Connection conn )
 throws SQLException, LoaderException
{
    // Obtenir l'objet de version initiale lors de la dernière lecture ou mise
    // à jour de cette entrée
    // de mappe dans la base de données.
    Employee emp = (Employee) le.getCurrentValue();
    long initialVersion = ((Long) le.getVersionedValue()).longValue();
    // Obtenir l'objet de version de l'objet Employee mis à jour pour
    // l'opération SQL
    //update.
    Long currentVersion = (Long)ivOcb.getVersionedObjectForValue( emp );
    long nextVersion = currentVersion.longValue();
    // Créer maintenant l'instruction SQL update qui inclut l'objet de
    // version dans la clause where
    // pour la vérification optimiste.
    String sql = "update EMPLOYEE set LASTNAME = ?, FIRSTNAME = ?,
    DEPTNO = ?,SEQNO = ?, MGRNO = ? where EMPNO = ? and SEQNO = ?";
    PreparedStatement sqlUpdate = ivTcb.getPreparedStatement( tx, conn,
   "employee", sql );
    sqlUpdate.setString(1, emp.getLastName());
    sqlUpdate.setString(2, emp.getFirstName());
    sqlUpdate.setString(3, emp.getDepartmentName());
    sqlUpdate.setLong(4, nextVersion );
    sqlUpdate.setInt(5, emp.getManagerNumber());
    sqlUpdate.setInt(6, key);
    sqlUpdate.setLong(7, initialVersion);
    sqlUpdate.addBatch();
}
```

L'exemple montre que l'objet LogElement est utilisé pour obtenir la valeur de la version initiale. Lorsque la transaction accède pour la première fois à l'entrée de

mappe, un objet LogElement est créé avec l'objet Employee initial obtenu à partir de la mappe. L'objet Employee initial est également transmis à la méthode getVersionedObjectForValue dans l'interface OptimisticCallback et le résultat est enregistré dans l'objet LogElement. Ce traitement a lieu avant qu'une application soit référencée à l'objet Employee initial et qu'elle appelle une méthode modifiant l'état de l'objet Employee initial.

L'exemple montre que le chargeur utilise la méthode getVersiondObjectForValue pour obtenir l'objet de version pour l'objet Employee mis à jour. Avant d'appeler la méthode batchUpdate dans l'interface Loader, eXtreme Scale appelle la méthode updateVersionedObjectForValue dans l'interface OptimisticCallback pour provoquer la génération d'un nouvel objet de version pour l'objet Employee mis à jour. Une fois que la méthode batchUpdate est renvoyée à l'ObjectGrid, l'objet LogElement est mis à jour avec l'objet de version actuelle et devient le nouvel objet de version initiale. Cette étape est nécessaire parce qu'il se peut que l'application ait appelé la méthode flush sur la mappe et non la méthode commit sur la session. Il est possible que le chargeur soit appelé plusieurs fois par une seule transaction pour la même clé. C'est pourquoi, eXtreme Scale fait en sorte que l'objet LogElement soit mis à jour avec l'objet de nouvelle version à chaque mise à jour de la ligne dans la table employee.

Désormais que le chargeur détient l'objet de version initiale et l'objet de prochaine version, il peut exécuter une instruction SQL update qui définit la colonne SEQNO sur la valeur de l'objet de prochaine version et utilise cette dernière dans la clause where. Cette approche est parfois désignée comme étant une instruction update sur-qualifiée. L'utilisation d'une instruction update sur-qualifiée permet à la base de données relationnelle de vérifier que la ligne n'a pas été modifiée par une autre transaction entre la lecture des données de la base de données par cette transaction et la mise à jour de la base de données par cette transaction. Si une autre transaction a modifié la ligne, le tableau de comptage renvoyé par la mise à jour par lots indique qu'aucune ligne n'a été mise à jour pour cette clé. Le chargeur est chargé de vérifier que l'opération SQL update n'a pas mis à jour la ligne. Si c'est le cas, il affiche une exception com.ibm.websphere.objectgrid.plugins.OptimisticCollisionException pour informer la session que la méthode batchUpdate a échoué en raison de la tentative de plusieurs transactions simultanées de mettre à jour la même ligne dans la table de base de données. Cette exception provoque l'annulation de la session et l'application doit relancer la transaction entière. La justification réside dans le fait que la nouvelle tentative aboutira, ce qui explique pourquoi cette approche est appelée optimiste. L'approche optimiste est plus performante si les données sont peu modifiées ou si les transactions simultanées tentent rarement de mettre à jour la même ligne.

Le chargeur doit impérativement utiliser le paramètre de clé du constructeur OptimisticCollisionException pour identifier quelle clé ou quel ensemble de clés a provoqué l'échec de la méthode optimiste batchUpdate. Le paramètre de clé peut être l'objet clé proprement dit ou un tableau d'objets clé si plusieurs clés ont abouti à l'échec de la mise à jour optimiste. eXtreme Scale utilise la méthode getKey du constructeur OptimisticCollisionException pour déterminer quelles entrées de mappe contiennent les données périmées et ont provoqué l'exception. Un aspect du processus d'annulation consiste à expulser les entrées de mappe périmées de la mappe. Cette expulsion s'avère nécessaire pour que toutes les transactions suivantes accédant aux mêmes clés déclenchent la méthode get de l'interface Loader appelée pour actualiser les entrées de mappe avec les données actuelles de la base de données.

Autres modes d'implémentation d'une approche optimiste par un chargeur :

- Il n'existe aucune colonne d'horodatage ou de numéro de séquence. Dans ce cas, la méthode getVersionObjectForValue de l'interface OptimisticCallback renvoie simplement l'objet de valeur en tant que version. Avec cette approche, le chargeur doit créer une clause WHERE comprenant tous les champs l'objet version initial. Cette approche n'est pas efficace et tous les types de colonnes ne sont pas admissibles pour une utilisation dans la clause WHERE d'une instruction SQL update sur-qualifiée. Cette approche n'est généralement pas utilisée.
- Il n'existe aucune colonne d'horodatage ou de numéro de séquence. Cependant, contrairement à l'approche précédente, la clause WHERE contient uniquement les champs de valeur modifiées par la transaction. L'une des méthodes permettant de détecter les champs qui ont été modifiés consiste à donner au mode de copie de la mappe de sauvegarde la valeur CopyMode.COPY_ON_WRITE. Ce mode de copie exige la transmission d'une interface de valeur à la méthode setCopyMode dans l'interface de la mappe de sauvegarde. La mappe de sauvegarde crée des objets proxy dynamiques qui implémentent l'interface de valeur fournie. Avec ce mode de copie, le chargeur peut transtyper chaque valeur en objet com.ibm.websphere.objectgrid.plugins.ValueProxyInfo. L'interface ValueProxyInfo comporte une méthode qui permet au chargeur d'obtenir la liste des noms d'attributs modifiés par la transaction. Cette méthode permet au chargeur d'appeler les méthodes get dans l'interface de valeur pour les noms d'attributs afin de consulter les données modifiées et créer une instruction SQL update qui définit uniquement les attributs modifiés. La clause where peut désormais être créée en incluent la colonne de clé primaire plus chacune des colonnes d'attributs modifiés. Cette approche est plus efficace que la précédente mais elle exige l'écriture de plus de code dans le chargeur et un cache de l'instruction préparée potentiellement plus important pour le traitement des différentes permutations. Toutefois, si les transactions modifient généralement que quelques attributs, cette limitation peut ne pas être problématique.
- Certaines bases de données relationnelle peuvent comporter une API pour la maintenance automatique des données de colonne particulièrement utiles pour la gestion optimiste des versions. Pour plus d'informations sur l'existence de ces possibilités, consultez la documentation de votre base de données.

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 «Chargeurs», à la page 90.

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 381.

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. Voir «Développement des chargeurs JPA basés sur le client», à la page 398 pour plus d'informations.

```
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 Réplication à des fins de disponibilitéles informations sur la réplication dans *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 371.

Référence associée:

«Remarques sur la programmation de chargeurs JPA», à la page 365
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.

Préchargement de mappe

Les mappes peuvent être associées à des loaders. Un loader sert à aller chercher des objets quand ils sont introuvables dans la mappe (absence du cache) et il sert également à écrire les modifications à un dorsal lors de la validation des transactions. Les loaders peuvent également servir à précharger des données dans une mappe. La méthode preloadMap de l'interface Loader est appelée sur chacune des mappes lorsque la partition qui correspond à la mappe dans le MapSet devient un fragment primaire. La méthode preloadMap n'est pas appelée sur des fragments réplique. Cette méthode tente de charger dans la mappe à partir du dorsal toutes les données de référence concernées à l'aide de la session fournie. La mappe pertinente est identifiée par l'argument BackingMap qui est passée à la méthode preloadMap.

```
void preloadMap(Session session, BackingMap backingMap) throws LoaderException;
```

Préchargement dans un MapSet partitionné

Les mappes peuvent être partitionnées en N partitions. Elles peuvent donc s'étendre sur plusieurs serveurs, chaque entrée étant identifiée par une clé qui n'est stockée que sur l'un de ces serveurs. Les mappes de très grande taille peuvent être détenues sur un eXtreme Scale car l'application n'est plus limitée par la taille du segment d'une seule JVM pour contenir toutes les entrées d'une mappe. Les applications qui veulent effectuer un préchargement avec la méthode preloadMap de l'interface Loader doivent identifier le sous-ensemble des données à précharger. Les partitions existent toujours en nombre fixe. Il est possible de déterminer ce nombre à l'aide de cet exemple de code :

```
int numPartitions = backingMap.getPartitionManager().getNumOfPartitions();
int myPartition = backingMap.getPartitionId();
```

Cet exemple de code montre comment une application peut identifier le sous-ensemble de données qu'elle doit précharge à partir de la base de données. Les applications doivent toujours utiliser ces méthodes même si la mappe n'est pas initialement partitionnée. Ces méthodes permettent la flexibilité : si la mappe est ultérieurement partitionnée par les administrateurs, le loader continuera à opérer correctement.

L'application doit émettre des requêtes pour extraire du dorsal le sous-ensemble *myPartition*. Si une base de données est utilisée, il peut être plus facile d'avoir une colonne avec l'identificateur de partition d'un enregistrement donné à moins qu'il n'y ait une requête naturelle permettant aux données de la table de se partitionner facilement.

Performances

L'implémentation du préchargement copie dans la mappe les données à partir du dorsal en stockant plusieurs objets dans la mappe en une seule transaction. Le

nombre optimal d'enregistrements à stocker par transaction dépend de plusieurs facteurs, notamment la complexité et la taille. Ainsi, après que la transaction comprend des blocs de plus de 100 entrées, les avantages en termes de performances diminuent au fur et à mesure que l'on augmente le nombre des entrées. Pour déterminer le nombre optimal, commencez par 100 entrées, puis augmentez le nombre jusqu'à ce que les performances deviennent nulles. Les transactions de grande taille donnent de meilleures performances de réplication. N'oubliez pas que seul le fragment primaire exécute le code de préchargement. Les données préchargées sont répliquées depuis le fragment primaire vers les fragments réplique qui sont en ligne.

Préchargement de MapSets

Si l'application utilise un MapSet comprenant plusieurs mappes, chacune de ces mappes a son propre loader. Chaque loader a une méthode preload. Chaque mappe est chargée en série par eXtreme Scale. Ce sera plus efficace de précharger toutes les mappes en désignant une mappe comme la mappe de préchargement. Ce processus est une convention d'application. On pourrait, par exemple, avoir deux mappes, Department et Employee, qui utilisent le loader Department pour précharger les deux mappes. Cette procédure garantit que, de manière transactionnelle, si une application veut un département, les salariés de ce département seront dans le cache. Lorsque le loader Department précharge un département depuis le dorsal, il récupère également les salariés de ce département. L'objet Department et les objets Employee qui lui sont associés sont ajoutés à la mappe à l'aide d'une seule transaction.

Préchargement récupérable

Il arrive que les clients aient des ensembles de données de très grosse taille et qui nécessitent d'être mis en cache. Précharger ces données peut prendre énormément de temps. Parfois, le préchargement doit être terminé pour que l'application puisse aller en ligne. C'est là que rendre récupérable le préchargement devient intéressant. Supposons qu'il y ait un million d'enregistrements à précharger. Le fragment primaire les télécharge et échoue au 800 000e enregistrement. En principe, le fragment réplique choisi pour être le nouveau fragment primaire efface tout état répliqué et repart du début. eXtreme Scale peut utiliser une interface ReplicaPreloadController. Le loader de l'application aura également besoin d'implémenter cette interface. Notre exemple ajoute une seule méthode au loader : `Status checkPreloadStatus(Session session, BackingMap bmap);`. Cette méthode est appelée par l'environnement d'exécution eXtreme Scale avant la méthode preload de l'interface Loader. eXtreme Scale teste le résultat de cette méthode (Status) pour déterminer son comportement au cas où un fragment réplique passe au statut de fragment primaire.

Tableau 6. Valeur du statut et réponse

Valeur de statut retournée	Réponse d'eXtreme Scale
Status.PRELOADED_ALREADY	eXtreme Scale n'appelle pas du tout la méthode preload car ce statut indique que la mappe est complètement préchargée.
Status.FULL_PRELOAD_NEEDED	eXtreme Scale efface la mappe et appelle de manière normale la méthode preload.
Status.PARTIAL_PRELOAD_NEEDED	eXtreme Scale laisse la mappe comme elle est et appelle la méthode preload. Cette stratégie permet au loader de l'application de continuer le préchargement à partir du point où il en était resté.

On le voit clairement : lorsqu'un fragment primaire précharge la mappe, il doit laisser dans une mappe du MapSet en cours de réplication une indication de l'état

pour que le fragment réplique détermine quel statut retourner. Vous pouvez utiliser une mappe supplémentaire appelée, par exemple, RecoveryMap. Cette RecoveryMap doit faire partie du même MapSet que celui qui est préchargé afin de garantir que la mappe est répliquée de manière cohérente avec les données en cours de préchargement. Nous allons suggérer une implémentation.

Lorsque le préchargement valide chaque bloc d'enregistrements, dans le cadre de cette transaction, le processus actualise également un compteur ou une valeur dans la RecoveryMap. Les données préchargées et celles de la RecoveryMap sont répliquées de manière atomique vers les fragments réplique. Lorsque le fragment réplique passe au statut de fragment primaire, il est à présent en mesure de vérifier la RecoveryMap pour voir ce qui s'est passé.

La RecoveryMap peut contenir une seule entrée avec la clé d'état. S'il n'existe aucun objet pour cette clé, vous aurez besoin d'un préchargement complet (checkPreloadStatus retourne alors FULL_PRELOAD_NEEDED). S'il existe un objet pour cette clé et que la valeur est COMPLETE, le préchargement s'effectue et la méthode checkPreloadStatus retourne PRELOADED_ALREADY. Sinon, l'objet value indique à quel endroit le préchargement redémarre et la méthode checkPreloadStatus retourne PARTIAL_PRELOAD_NEEDED. Le loader peut stocker le point de récupération dans une variable d'instance du loader de manière à ce que, lorsque le préchargement est appelé, le loader sache d'où partir. La RecoveryMap peut également détenir une entrée par mappe si chacune des mappes est préchargée de manière indépendante.

Gérer la récupération en mode de réplication synchrone avec un Loader

L'environnement d'exécution d'eXtreme Scale est conçu pour ne pas perdre de données validées en cas de défaillance du fragment primaire. Nous allons voir quels algorithmes sont utilisés à cet effet. Ces algorithmes ne s'appliquent que lorsqu'un groupe de réplication utilise la réplication synchrone. L'usage d'un loader n'est pas obligatoire.

Il est possible de configurer l'environnement d'exécution d'eXtreme Scale pour répliquer de manière synchrone vers les fragments réplique toutes les modifications d'un fragment primaire. Lorsqu'il est placé, un fragment réplique synchrone reçoit une copie des données existant dans le fragment primaire. Pendant ce temps, le fragment primaire continue de recevoir des transactions qu'il copie de manière asynchrone vers le fragment réplique. A ce moment-là, le fragment réplique n'est pas encore considéré comme étant en ligne.

Une fois que le fragment réplique a rattrapé le fragment primaire, il passe en mode homologue et la réplication synchrone peut commencer. Toute transaction validée sur le fragment primaire est envoyée aux fragments réplique synchrones et le fragment primaire attend la réponse de chacun de ces fragments. Une séquence de validation synchrone avec un loader dans le fragment primaire ressemble à la procédure suivante :

Tableau 7. Séquence de validation dans le fragment primaire

Avec loader	**Sans loader**
Obtention des verrous pour les entrées	Identique
Vidage des modifications vers le loader	"No operation"
Enregistrement des modifications dans le cache	Identique

Tableau 7. Séquence de validation dans le fragment primaire (suite)

Avec loader	Sans loader
Envoi des modifications vers les fragments réplique et attente d'accusé de réception	Identique
Validation vers le loader via le plug-in TransactionCallback	Appel de la validation via le plug-in, mais sans effet
Libération des verrous pour les entrées	Identique

Vous remarquerez que les modifications sont envoyées aux fragments réplique avant d'être validées vers le loader. Pour déterminer lorsque les modifications sont validées dans le fragment réplique, modifiez cette séquence : lors de l'initialisation, initialisez de la manière suivante les listes tx dans le fragment primaire.

```
CommitedTx = {}, RolledBackTx = {}
```

Pendant le traitement synchrone des validations, utilisez la séquence suivante :

Tableau 8. Traitement synchrone des validations

Avec loader	Sans loader
Obtention des verrous pour les entrées	Identique
Vidage des modifications vers le loader	"No operation"
Enregistrement des modifications dans le cache	Identique
Envoi des modifications avec une transaction validée, annulation de la transaction vers le fragment réplique et attente d'accusé de réception	Identique
Effacement de la liste des transactions validées et des transactions annulées	Identique
Validation vers le loader via le plug-in TransactionCallBack	La validation via le plug-in TransactionCallBack est toujours appelée, mais en principe sans effet
Si la validation réussit, ajout de la transaction aux transactions validées, sinon, ajout aux transactions annulées	"No operation"
Libération des verrous pour les entrées	Identique

Pour le traitement des fragments réplique, utilisez la séquence suivante :

1. Réception des modifications
2. Validation de toutes les transactions reçues dans la liste des transactions validées
3. Annulation de toutes les transactions reçues dans la liste des transactions annulées
4. Démarrage d'une transaction ou d'une session
5. Application des modifications à la transaction ou à la session
6. Enregistrement de la transaction ou de la session dans la liste en attente
7. Renvoi de la réponse

Vous remarquerez que, dans le fragment réplique, il ne se passe aucune interaction avec le loader tant que le fragment réplique est en mode réplique. C'est au fragment primaire d'impulser toutes les modifications via le loader. Le fragment réplique n'effectue aucune modification. Un effet collatéral de cet algorithme est

que le fragment réplique dispose toujours des transactions, mais celles-ci ne sont validées qu'après que la transaction primaire suivante a envoyé le statut de validation de ces transactions. Les transactions sont alors validées ou annulées dans le fragment réplique. Jusque-là, les transactions ne sont pas validées. Il est possible d'ajouter dans le fragment primaire un minuteur qui envoie le résultat de la transaction après un bref délai (quelques secondes). Ce minuteur limite, sans l'éliminer tout à fait, le décalage de ce créneau. Ce décalage n'est un problème qu'en mode de lecture de réplique. Sinon, il n'a aucun impact sur l'application.

Lorsque le fragment primaire échoue, c'est vraisemblablement que quelques transactions ont été validées ou annulées dans ce fragment primaire, mais que le message n'a jamais été transmis au fragment réplique avec ces résultats. Lorsqu'un fragment réplique passe au rang de nouveau fragment primaire, l'une de ses premières actions est de gérer cette situation. Chaque transaction en attente est traitée à nouveau par rapport à l'ensemble de mappes du nouveau fragment primaire. S'il y a un loader, chaque transaction est remise à ce dernier. Ces transactions sont appliquées dans un ordre FIFO strict. Si des transactions échouent, cet ordre est ignoré. Supposons que nous ayons trois transactions en attente, A, B et C et que A soit validée, B annulée ainsi que C. Aucune de ces trois transactions n'a d'impact sur les autres. Supposons qu'elles soient indépendantes.

Lorsqu'il se trouve en mode de reprise par basculement, un loader pourra vouloir utiliser une logique légèrement différente de celle utilisée en mode normal. L'implémentation de l'interface ReplicaPreloadController permet au loader de savoir facilement lorsqu'il est en mode de reprise par basculement. La méthode checkPreloadStatus n'est appelée que lorsque la reprise par basculement est terminée. Par conséquent, si la méthode apply de l'interface Loader est appelée avant la méthode checkPreloadStatus, il y a une transaction de récupération. Après l'appel de la méthode checkPreloadStatus, la récupération est terminée.

Configuration de la prise en charge d'un loader en écriture différée

Vous pouvez activer l'écriture différée en utilisant le fichier XML de descripteur d'ObjectGrid ou par programmation avec l'interface BackingMap.

Pour activer l'écriture différée, vous pouvez utiliser le fichier XML de descripteur d'ObjectGrid ou passer pa la programmation avec l'interface BackingMap.

Fichier XML de descripteur d'ObjectGrid

Lors de la configuration d'un ObjectGrid à l'aide d'un fichier XML de descripteur d'ObjectGrid, le chargeur en écriture différée est activé en définissant l'attribut write-behind dans la balise backingMap. Voici un exemple :

```
<objectGrid name="library" >
     <backingMap name="book" writeBehind="T300;C900" pluginCollectionRef="bookPlugins"/>
```

Dans l'exemple ci-dessus, l'écriture différée par la mappe de sauvegarde book est activée avec le paramètre T300;C900. L'attribut write-behind spécifie le temps de mise à jour maximal et/ou le nombre de mises à jour de clés maximal. Le format du paramètre write-behind est le suivant :

```
write-behind attribute  ::= <defaults> | <update time> | <update key count> | <update time> ";" <update key count>
update time  ::= "T" <positive integer>
update key count  ::= "C" <positive integer>
defaults  ::= "" {table}
```

Les mises à jour du chargeur ont lieu lorsque l'un des événements suivants se produit :

1. Le temps de mise à jour maximal, en secondes, s'est écoulé depuis la dernière mise à jour.
2. Le nombre de clés mises à jour dans la mappe de files d'attente a atteint le nombre de clés à mettre à jour.

Ces paramètres sont uniquement des indications. Le temps et le nombre réels de mises à jour seront compris dans une plage de paramètres proche. Toutefois, nous ne garantissons pas que le temps et le nombre réels de mises à jour soient identiques à ceux définis dans les paramètres. De plus, la première mise à jour suivante est susceptible d'avoir lieu dans un délai supérieur au double de celui de la mise à jour. Cela est dû au fait qu'ObjectGrid répartit au hasard l'heure de début des mises à jour de manière à ce que toutes les partitions n'accèdent pas à la base de données en même temps.

Dans l'exemple précédent T300;C900, le chargeur écrit les données dans le système dorsal lorsque 300 secondes se sont écoulées depuis la dernière mise à jour ou lorsque 900 clés sont en attente de mise à jour. Par défaut, le temps de mise à jour est de 300 secondes et le nombre de clés à mettre à jour est 1 000.

Tableau 9. Quelques options d'écriture différée

Valeur d'attribut	Temps
T100	Le temps de mise à jour est de 100 secondes et le nombre de clés à mettre à jour est de 1 000 (valeur par défaut)
C2000	Le temps de mise à jour est de 300 secondes (valeur par défaut) et le nombre de clés à mettre à jour est de 2 000.
T300;C900	Le temps de mise à jour est de 300 secondes et le nombre de clés à mettre à jour est de 900.
""	Le temps de mise à jour est de 300 secondes (valeur par défaut) et le nombre de clés à mettre à jour est de 1 000 (valeur par défaut) **Remarque :** Si vous configurez le chargeur en écriture différée en tant que chaîne vide : `writeBehind=""`, il est activé avec les valeurs par défaut. Par conséquent, ne spécifiez pas l'attribut d'écriture différée si vous ne souhaitez pas que l'écriture différée soit activée.

Activation par programmation de l'écriture différée

Lorsque vous créez une mappe de sauvegarde à l'aide d'un programme pour un eXtreme Scale en mémoire locale, vous pouvez utiliser la méthode suivante dans l'interface BackingMap pour activer ou désactiver l'écriture différée.

```
public void setWriteBehind(String writeBehindParam);
```

Pour plus de détails sur l'utilisation de la méthode setWriteBehind, voir les informations concernant l'interface BackingMap dans *Guide de programmation*.

Référence associée:

«Exemple : écriture d'une classe dumper en écriture différée», à la page 362
Cet exemple de code source montre comment écrire un programme de surveillance (dumper) pour gérer les mises à jour d'écriture différée ayant échoué.

Mise en cache en écriture différée :

Vous pouvez utiliser la mise en cache en écriture différée pour réduire le temps système supplémentaire nécessaire lors de la mise à jour d'une base de données utilisée en tant que base de données dorsale.

Présentation de la mise en cache en écriture différée

La mise en cache en écriture différée met en file d'attente de manière asynchrone les mises à jour du plug-in Loader. Vous pouvez améliorer les performances en déconnectant les mises à jour, les insertions et les suppressions au sein d'une mappe, le temps système pour la mise à jour de la base de données dorsale. La

mise à jour asynchrone est effectuée après un retard (de cinq minutes, par exemple) ou après un certain nombre d'entrées (1 000 entrées).

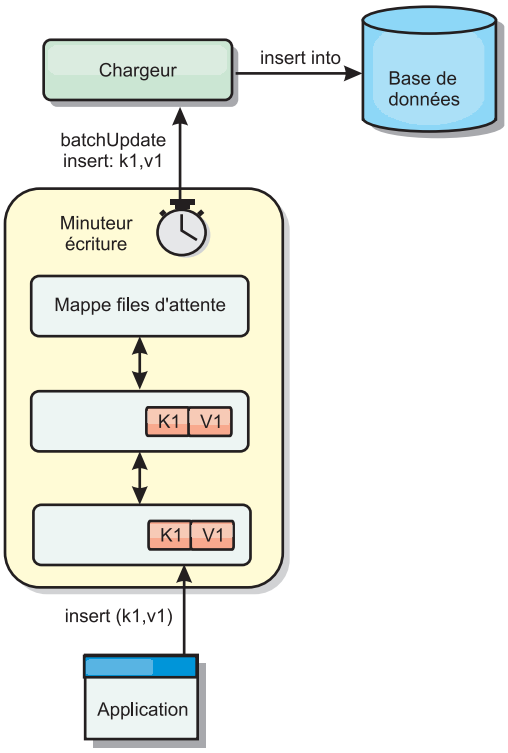


Figure 27. Mise en cache en écriture différée

La configuration à écriture différée sur une mappe de sauvegarde crée une unité d'exécution entre le Loader et la mappe. Le Loader délègue alors les demandes de données via l'unité d'exécution en fonction des paramètres de configuration de la méthode BackingMap.setWriteBehind. Lorsqu'une transaction eXtreme Scale insère, met à jour ou supprime une entrée dans une mappe, un objet LogElement est créé pour chacun de ces enregistrements. Ces éléments sont envoyés au Loader à écriture différée et mis en file d'attente dans une ObjectMap spéciale appelée mappe de files d'attente. Chaque mappe de sauvegarde pour laquelle le paramètre d'écriture différée est activé a ses propres mappes de files d'attente. L'unité d'exécution à écriture différée supprime périodiquement les données mises en file d'attente des mappes correspondantes et les insère dans le Loader dorsal.

Le chargeur à écriture différée envoie uniquement les types insertion, mise à jour et suppression des objets LogElement au chargeur réel. Tous les autres types, par exemple le type EVICT, sont ignorés.

La prise en charge de l'écriture différée *est* une extension du plug-in Loader, qui vous permet d'intégrer eXtreme Scale à la base de données. A ce sujet, vous pouvez consulter avec profit les explications Configuration des chargeurs JPA sur la configuration d 'un chargeur JPA.

Avantages

L'activation de l'écriture différée présente les avantages suivants :

- **Isolement en cas d'arrêt anormal de la base de données dorsale :** la mise en cache à écriture différée propose une couche d'isolement en cas d'arrêt anormal

de la base de données dorsale. Les mises à jour sont alors placées dans la mappe de files d'attente. Les applications peuvent continuer à envoyer des transactions vers eXtreme Scale. Lors de la reprise du système dorsal, les données contenues dans la mappe de files d'attente sont insérées dans celui-ci.

- **Réduction de la charge du système dorsal :** le chargeur à écriture différée fusionne les mises à jour en fonction des clés de façon qu'une seule mise à jour fusionnée par clé existe dans la mappe de files d'attente. Cette fusion diminue le nombre de mises à jour dans la base de données dorsale.
- **Amélioration des performances de la transaction :** la durée de chaque transaction eXtreme Scale est réduite car la transaction n'a plus à attendre que les données soient synchronisées avec le système dorsal.

Considérations liées à la conception d'applications

L'activation de l'écriture différée est une opération simple, mais la création d'une application devant utiliser l'écriture différée requiert une attention particulière. Sans écriture différée, la transaction ObjectGrid encadre la transaction dorsale. La transaction ObjectGrid démarre avant la transaction dorsale et se termine après celle-ci.

Lorsque la prise en charge de l'écriture différée est activée, la transaction ObjectGrid se termine avant le début de la transaction dorsale. La transaction ObjectGrid et la transaction dorsale sont dissociées.

Contraintes d'intégrité référentielle

Chaque mappe de sauvegarde configurée avec écriture différée dispose de sa propre unité d'exécution d'écriture différée pour envoyer les données vers le système dorsal. Les données mises à jour dans les différentes mappes d'une transaction ObjectGrid sont mises à jour dans le système dorsal via différentes transactions dorsales. Par exemple, la transaction T1 met à jour la clé key1 dans la mappe Map1 et la clé key2 dans la mappe Map2. La clé key1 mise à jour dans la mappe Map1 est actualisée vers le système dorsal dans une transaction expéditrice et la clé key2 actualisée dans la mappe Map2 l'est dans une autre transaction expéditrice ; cette mise à jour vers le dorsal est effectuée dans deux unités d'exécution différentes, toutes deux en écriture différée. Si les données stockées dans Map1 et Map2 ont des liens, tels que des contraintes de clé externe dans le système dorsal, les mises à jour sont susceptibles d'échouer.

Lors de la conception de contraintes d'intégrité référentielle dans votre base de données dorsale, vérifiez que les mises à jour désordonnées sont autorisées.

Verrouillage d'une mappe de files d'attente

Le comportement des transactions en matière de verrouillage constitue une autre différence notable. La grille d'objets prend en charge trois stratégies de verrouillage différentes : PESSIMISTIC, OPTIMISITIC et NONE. Les mappes de files d'attente à écriture différée utilisent la stratégie de verrouillage pessimiste, quelle que soit la stratégie de verrouillage configurée pour leur mappe de sauvegarde. Deux types différents d'opérations permettant d'acquérir un verrou sur la mappe de files d'attente existent :

- Lorsqu'une transaction ObjectGrid est validée ou qu'un vidage (de mappe ou de session) se produit, la transaction lit la clé de la mappe de files d'attente et place un verrou S sur la clé.

- Lorsqu'une transaction ObjectGrid est validée, elle tente de mettre à niveau le verrou S vers un verrou X sur la clé.

Ce comportement de mappe de files d'attente supplémentaire vous permet de voir quelques différences dans le comportement de verrouillage.

- Si la mappe de l'utilisateur est configurée comme stratégie de verrouillage pessimiste, la différence dans le comportement de verrouillage n'est pas grande. Chaque fois qu'un vidage ou qu'une validation est appelée, un verrou S est placé sur la même clé dans la mappe de files d'attente. Au moment de la validation, un verrou X est acquis pour la clé dans la mappe de l'utilisateur, mais également pour la clé dans la mappe de files d'attente.
- Si la mappe de l'utilisateur est configurée comme stratégie de verrouillage optimiste ou inexistante, la transaction utilisateur suit le modèle de la stratégie pessimiste. Chaque fois qu'un vidage ou qu'une validation est appelée, un verrou S est acquis sur la même clé dans la mappe de files d'attente. Au moment de la validation, un verrou X est acquis pour la clé dans la mappe de files d'attente utilisant la même transaction.

Nouvelles tentatives de transaction du chargeur

ObjectGrid ne prend pas en charge les transactions à 2 phases ou transactions XA. L'unité d'exécution à écriture différée supprime les enregistrements de la mappe de files d'attente et met à jour les enregistrements dans le système dorsal. En cas d'échec du serveur au milieu de la transaction, certaines mises à jour dorsales risquent d'être perdues.

Le chargeur à écriture différée tente automatiquement d'écrire à nouveau les transactions ayant échoué et envoie une LogSequence en attente de validation au système dorsal pour éviter toute perte de données. Pour que l'exécution de cette action soit possible, le chargeur doit être idempotent, ce qui signifie que lorsque Loader.batchUpdate(TxId, LogSequence) est appelé deux fois avec la même valeur, le résultat est le même que s'il était appelé une fois. Les implémentations du chargeur doivent implémenter l'interface RetryableLoader pour activer cette fonction. Pour plus de détails, consultez la documentation relative à l'API.

Echecs du chargeur

Le plug-in du chargeur (Loader) risque d'échouer lorsqu'il ne parvient pas à communiquer avec le dorsal de base de données. Cela peut se produire si le serveur de base de données ou la connexion réseau est arrêté(e). Le chargeur en écriture différée met en file d'attente les mises à jour et tente d'envoyer régulièrement les données modifiées au chargeur. Ce dernier doit signaler le problème de connectivité à l'environnement d'exécution ObjectGrid en générant une exception LoaderNotAvailableException.

L'implémentation du chargeur doit donc pouvoir distinguer un échec lié aux données d'une défaillance physique du chargeur. En cas d'échec lié aux données, une exception LoaderException ou OptimisticCollisionException doit être générée, alors qu'en cas de défaillance physique du chargeur, une exception LoaderNotAvailableException doit être générée. ObjectGrid gère ces deux exceptions de manière différente :

- Si une exception LoaderException est interceptée par le chargeur en écriture différée, celui-ci considère que l'échec est dû à une défaillance de données, telle qu'une erreur de clé en double. Le chargeur dégroupe la mise à jour et tente de ne mettre à jour qu'un enregistrement à la fois, afin d'isoler la défaillance de

données. Si une exception {{LoaderException}} est à nouveau détectée lors de la mise à jour de l'enregistrement concerné, un enregistrement d'échec de la mise à jour est créé et consigné dans la mappe des mises à jour ayant échoué.
- Si une exception LoaderNotAvailableException est interceptée par le chargeur en écriture différée, celui-ci considère que l'échec est dû à l'impossibilité de se connecter à la base de données, par exemple, lorsque la base de données dorsale est inactive, lorsque la connexion à une base de données est indisponible ou lorsque le réseau est inactif. Le chargeur attend 15 secondes, puis tente à nouveau la mise à jour par lots de la base de données.

L'erreur courante est d'émettre une exception LoaderException à la place d'une exception LoaderNotAvailableException. Tous les enregistrements du chargeur à écriture différée deviennent alors des enregistrements d'échec de la mise à jour, ce qui réduit à néant l'objectif de l'isolement en cas d'arrêt anormal du système dorsal.

Remarques sur les performances

En supprimant de la transaction la mise à jour du chargeur, la mise en cache en écriture différée augmente le temps de réponse. Elle augmente également la capacité de traitement de la base de données, car les mises à jour de base de données sont combinées. Il est important de comprendre le temps système supplémentaire généré par l'unité d'exécution à écriture différée, qui permet de retirer les données de la mappe de files d'attente et de les insérer dans le chargeur.

Le temps de mise à jour maximal et le nombre de mises à jour maximal doivent être ajustés en fonction de l'environnement et des types d'utilisation prévus. Si la valeur du temps ou du nombre de mises à jour maximal est trop petite, le temps système de l'unité d'exécution d'écriture différée peut dépasser les avantages tirés. Définir une valeur élevée pour ces deux paramètres peut également augmenter l'utilisation de la mémoire pour la mise en file d'attente des données et retarder le moment de péremption des enregistrements de bases de données.

Pour des performances optimales, réglez les paramètres d'écriture différée en fonction des facteurs suivants :
- ratio des transactions de lecture et d'écriture
- fréquence de mise à jour d'enregistrements identiques
- temps d'attente pour la mise à jour de la base de données

Référence associée:

«Exemple : écriture d'une classe dumper en écriture différée», à la page 362
Cet exemple de code source montre comment écrire un programme de surveillance (dumper) pour gérer les mises à jour d'écriture différée ayant échoué.

Traitement des échecs de mises à jour en écriture différée :

Etant donné que la transaction WebSphere eXtreme Scale se termine avant que la transaction expéditrice démarre, la transaction peut avoir un faux succès.

Si vous tentez d'insérer une entrée dans une transaction eXtreme Scale qui n'existe pas dans la mappe de sauvegarde, mais existe dans la base de données dorsale, entraînant ainsi une clé en double, la transaction eXtreme Scale aboutit. Cependant, la transaction dans laquelle l'unité d'exécution d'écriture différée insère l'objet dans la base de données du système dorsal échoue avec une exception de clé en double.

Traitement des échecs de mise à jour d'écriture différée : côté client

Ce type de mise à jour ou tout autre mise à jour dorsale ayant échoué est une mise à jour en écriture différée échouée. Les échecs de mises à jour en écriture différée sont stockés dans une mappe d'échecs de mises à jour en écriture différée. Cette mappe sert de file d'attente d'événements pour les mises à jour échouées. La clé de la mise à jour est un objet Integer unique et la valeur correspond à une instance de FailedUpdateElement. La mappe de mise à jour en écriture différée qui a échoué est configurée avec un expulseur qui expulse les enregistrements une heure après qu'il a été inséré. Ainsi, les enregistrements de mise à jour ayant échoué sont perdus s'ils ne sont pas récupérés sous une heure.

L'API ObjectMap peut être utilisée pour récupérer les entrées de mappe des échecs de mises à jour en écriture différée. Le nom de mappe de mise à jour eb écriture différée est : IBM_WB_FAILED_UPDATES_<map name>. Pour connaître le nom de préfixe de chacune des mappes système en écriture différée, reportez-vous à la documentation de l'API WriteBehindLoaderConstants. Voici un exemple.

échec de processus - exemple de code

```
ObjectMap failedMap = session.getMap(
    WriteBehindLoaderConstants.WRITE_BEHIND_FAILED_UPDATES_MAP_PREFIX + "Employee");
Object key = null;

session.begin();
while(key = failedMap.getNextKey(ObjectMap.QUEUE_TIMEOUT_NONE)) {
    FailedUpdateElement element = (FailedUpdateElement) failedMap.get(key);
    Throwable throwable = element.getThrowable();
    Object failedKey = element.getKey();
    Object failedValue = element.getAfterImage();
    failedMap.remove(key);
    // Faites quelque chose d'intéressant avec la clé, la valeur ou l'exception.

}
session.commit();
```

Un appel de la méthode getNextKey fonctionne avec une partition spécifique pour chaque transaction eXtreme Scale. Dans un environnement réparti, pour obtenir les clés de toutes les partitions, vous devez démarrer plusieurs transactions, comme le montre l'exemple suivant :

obtention des clés de toutes les partitions - exemple de code

```
ObjectMap failedMap = session.getMap(
    WriteBehindLoaderConstants.WRITE_BEHIND_FAILED_UPDATES_MAP_PREFIX + "Employee");
while (true) {
    session.begin();
    Object key = null;
    while(( key = failedMap.getNextKey(5000) )!= null ) {
        FailedUpdateElement element = (FailedUpdateElement) failedMap.get(key);
    Throwable throwable = element.getThrowable();
        Object failedKey = element.getKey();
        Object failedValue = element.getAfterImage();
        failedMap.remove(key);
        // Faites quelque chose d'intéressant avec la clé, la valeur ou l'exception.

    }
    Session.commit();
}
```

Remarque : La mappe de mise à jour ayant échoué fournit un moyen de surveiller l'état de l'application. Si un système génère plusieurs enregistrements dans la mappe de mise à jour ayant échoué, cela indique que vous devez modifier

l'application ou l'architecture pour utiliser l'écriture différée. Vous pouvez utiliser la commande **xscmd -showMapSizes** pour afficher la taille des entrées de mappe de mise à jour ayant échoué.

Traitement des échecs de mise à jour d'écriture différée : écouteur de fragments

La détection et la consignation des échecs de transactions en écriture différée sont importantes. Toutes les applications en écriture différée doivent implémenter un observateur pour traiter les échecs des mises à jour en écriture différée. Cette méthode permet d'éviter d'être à court de mémoire car les enregistrements d'une mappe d'échecs de mises à jour, censés être traités par l'application, ne sont pas expulsés.

Le code suivant montre comment se connecter à un observateur ou "videur," qui doit être ajouté au XML descripteur d'ObjectGrid comme dans le fragment.

```
<objectGrid name="Grid">
 <bean id="ObjectGridEventListener" className="utils.WriteBehindDumper"/>
```

Vous pouvez voir le bean ObjectGridEventListener qui a été ajouté et qui correspond à l'observateur d'écriture différé dont nous avons parlé plus haut. L'observateur interagit avec les mappes pour tous les fragments primaires d'une JVM à la recherche de ceux pour lesquels l'écriture différée a été activée. S'il trouve un fragment, il tente de consigner jusqu'à 100 échecs de mises à jour. Il observe en continu un fragment primaire jusqu'à ce que ce dernier soit déplacé vers une autre JVM. Toutes les applications utilisant l'écriture différée doivent utiliser un observateur similaire à celui-ci. Si ce n'est pas le cas, la mémoire de la machines virtuelles Java devient insuffisante car les enregistrements de cette mappe d'erreurs ne sont jamais expulsés.

Pour plus d'informations, voir «Exemple : écriture d'une classe dumper en écriture différée».

Référence associée:

«Exemple : écriture d'une classe dumper en écriture différée»
Cet exemple de code source montre comment écrire un programme de surveillance (dumper) pour gérer les mises à jour d'écriture différée ayant échoué.

Exemple : écriture d'une classe dumper en écriture différée :

Cet exemple de code source montre comment écrire un programme de surveillance (dumper) pour gérer les mises à jour d'écriture différée ayant échoué.

```
//
//Cet exemple de programme est fourni TEL QUEL et peut être utilisé, exécuté, copié et
//modifiés sans acquittement de droits par le client (a) pour sa propre formation et
//analyse, (b) pour développer des applications destinées à s'exécuter avec un produit IBM
//WebSphere pour une utilisation interne par le client ou pour redistribution
//par le client, dans le cadre d'une telle application, dans les produits du client. "
//
//5724-J34 (C) COPYRIGHT International Business Machines Corp. 2009
//  Tous droits réservés *  Eléments sous licence - Propriété d'IBM
//
package utils;

import java.util.Collection;
import java.util.Iterator;
import java.util.concurrent.Callable;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.ScheduledThreadPoolExecutor;
import java.util.concurrent.TimeUnit;
import java.util.logging.Logger;

import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.ObjectGrid;
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.UndefinedMapException;
import com.ibm.websphere.objectgrid.plugins.ObjectGridEventGroup;
import com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener;
import com.ibm.websphere.objectgrid.writebehind.FailedUpdateElement;
import com.ibm.websphere.objectgrid.writebehind.WriteBehindLoaderConstants;

/**
 * L'écriture différée s'attend à ce que les transactions vers le chargeur aboutissent. Si une
 * transaction échoue pour une clé, elle insère une entrée dans une mappe appelée PREFIXE + nomMappe.
 * L'application doit rechercher dans cette mappe les entrées de vidage des incidents de
 * transaction d'écriture différée. L'application est chargée d'analyser, puis de supprimer
 * ces entrées. Ces dernières peuvent être assez importantes car elles incluent la clé, les
 * images avant et après de la valeur et l'exception elle-même. Les exceptions peuvent facilement
 * atteindre 20k.
 *
 * La classe est enregistrée avec la grille et une instance est créée pour
 * chaque fragment primaire d'une machine virtuelle Java. Une unique unité
 * d'exécution est créée et cette dernière recherche le fragment dans chaque
 * mappe d'erreurs à écriture différée, imprime le problème, puis supprime
 * l'entrée.
 *
 * Cela signifie qu'il existera une unité d'exécution par fragment. Si le
 * fragment est transféré sur une autre machine virtuelle Java, la méthode
 * deactivate arrête l'unité d'exécution.
 * @author bnewport
 *
 */
public class WriteBehindDumper implements ObjectGridEventListener, ObjectGridEventGroup.ShardEvents,
 Callable<Boolean>
{
 static Logger logger = Logger.getLogger(WriteBehindDumper.class.getName());

 ObjectGrid grid;

 /**
  * Pool d'unités d'exécution chargé de gérer les vérificateurs de table. Si l'application possède
  * son propre pool, modifiez la valeur pour réutiliser le pool existant
  */
 static ScheduledExecutorService pool = new ScheduledThreadPoolExecutor(2); // deux unités d'exécution
      pour vider les enregistrements

 // futur de ce fragment
 ScheduledFuture<Boolean> future;

 // true si ce fragment est actif
 volatile boolean isShardActive;

 /**
  * Durée normale entre les recherches dans les mappes d'erreurs d'écriture différée
  */
 final long BLOCKTIME_SECS = 20L;

 /**
  * Une session allouée pour ce fragment. Il est inutile de les allouer encore et encore
  */
 Session session;
 /**
  * Si un fragment primaire est activé, planifiez les vérifications de sorte à vérifier périodiquement
  * les mappes d'erreurs d'écriture différée et imprimer les éventuels problèmes
  */
 public void shardActivated(ObjectGrid grid)
 {
  try
  {
   this.grid = grid;
   session = grid.getSession();

   isShardActive = true;
   future = pool.schedule(this, BLOCKTIME_SECS, TimeUnit.SECONDS); // vérification initiale toutes
                 les BLOCKTIME_SECS secondes
  }
  catch(ObjectGridException e)
  {
   throw new ObjectGridRuntimeException("Exception activating write dumper", e);
  }
 }

 /**
  * Marquez le fragment comme inactif, puis annulez le vérificateur
  */
 public void shardDeactivate(ObjectGrid arg0)
 {
  isShardActive = false;
  // s'il est annulé, l'annulation renvoie la valeur true
  if(future.cancel(false) == false)
  {
   // sinon, la tâche est bloquée jusqu'à ce que le vérificateur ait terminé son exécution
   while(future.isDone() == false) // attendez que la tâche se termine d'une manière ou d'une autre
   {
```

```
   try
   {
    Thread.sleep(1000L); // vérifiez à chaque seconde
   }
   catch (InterruptedException e)
   {
   }
  }
 }
}

/**
 * Test simple permettant de vérifier si l'écriture différée est activée
 * pour la mappe ; si c'est le cas, renvoie le nom de la mappe d'erreurs
 * associée.
 * @param mapName Mappe à tester
 * @return Nom de la mappe d'erreurs à écriture différée si elle existe ;
 * sinon null
 */
static public String getWriteBehindNameIfPossible(ObjectGrid grid, String mapName)
{
 BackingMap map = grid.getMap(mapName);
 if(map != null && map.getWriteBehind() != null)
 {
  return WriteBehindLoaderConstants.WRITE_BEHIND_FAILED_UPDATES_MAP_PREFIX + mapName;
 }
 else
  return null;
}

/**
 * Exécuté pour chaque fragment. Vérifie si l'écriture différée est activée pour chaque mappe et
 * si c'est le cas, imprime les éventuelles erreurs de transaction d'écriture différée
 * avant de supprimer l'enregistrement.
 */
public Boolean call()
{
 logger.fine("Called for " + grid.toString());
 try
 {
  // tant que le fragment primaire est présent sur cette machine virtuelle Java,
  // seules les mappes définies par l'utilisateur sont renvoyées ici ; aucune mappe système,
  // telle que les mappes d'écriture différée, ne figure dans cette liste.
  Iterator<String> iter = grid.getListOfMapNames().iterator();
  boolean foundErrors = false;
  // effectuez une itération sur toutes les mappes actuelles
  while(iter.hasNext() && isShardActive)
  {
   String origName = iter.next();

   // s'il s'agit d'une mappe d'erreurs à écriture différée
   String name = getWriteBehindNameIfPossible(grid, origName);
   if (name != null)
   {
    // essayez de supprimer des blocs de N erreurs à la fois
    ObjectMap errorMap = null;
    try
    {
     errorMap = session.getMap(name);
    }
    catch(UndefinedMapException e)
    {
     // au démarrage les mappes d'erreurs risquent de ne pas encore exister ; patience...
     continue;
    }
    // essayez de vider jusqu'à N enregistrements à la fois
    session.begin();
    for(int counter = 0; counter < 100; ++counter)
    {
     Integer seqKey = (Integer)errorMap.getNextKey(1L);
     if(seqKey != null)
     {
      foundErrors = true;
      FailedUpdateElement elem = (FailedUpdateElement)errorMap.get(seqKey);
      //
      // Votre application doit consigner le problème ici
      logger.info("WriteBehindDumper ( " + origName + ") for key (" + elem.getKey() + ") Exception: " +
       elem.getThrowable().toString());
      //
      //
      errorMap.remove(seqKey);
     }
     else
      break;
    }
    session.commit();
   }
  } // passez à la mappe suivante
  // bouclez plus rapidement en cas d'erreurs
  if(isShardActive)
  {
```

```
    // replanifiez après une seconde en cas d'enregistrements incorrects ;
    // sinon, attendez 20 secondes.
    if(foundErrors)
     future = pool.schedule(this, 1L, TimeUnit.SECONDS);
    else
     future = pool.schedule(this, BLOCKTIME_SECS, TimeUnit.SECONDS);
   }
  }
  catch(ObjectGridException e)
  {
   logger.fine("Exception in WriteBehindDumper" + e.toString());
   e.printStackTrace();

   //ne laissez pas de transaction sur la session.
   if(session.isTransactionActive())
   {
    try { session.rollback(); } catch(Exception e2) {}
   }
  }
  return true;
 }

 public void destroy() {
  // TODO Module de remplacement de méthode généré automatiquement

 }

 public void initialize(Session arg0) {
  // TODO Module de remplacement de méthode généré automatiquement

 }

 public void transactionBegin(String arg0, boolean arg1) {
  // TODO Module de remplacement de méthode généré automatiquement

 }

 public void transactionEnd(String arg0, boolean arg1, boolean arg2,
   Collection arg3) {
  // TODO Module de remplacement de méthode généré automatiquement

 }
}
```

Concepts associés:

«Configuration de la prise en charge d'un loader en écriture différée», à la page 355
Vous pouvez activer l'écriture différée en utilisant le fichier XML de descripteur d'ObjectGrid ou par programmation avec l'interface BackingMap.

«Mise en cache en écriture différée», à la page 86
Vous pouvez utiliser la mise en cache en écriture différée pour réduire le temps système supplémentaire nécessaire lors de la mise à jour d'une base de données utilisée en tant que base de données dorsale.

«Traitement des échecs de mises à jour en écriture différée», à la page 360
Etant donné que la transaction WebSphere eXtreme Scale se termine avant que la transaction expéditrice démarre, la transaction peut avoir un faux succès.

Remarques sur la programmation de chargeurs 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 les considérations de conception d'application, voir les informations sur le plug-in JPAEntityLoader dans *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.

Concepts associés:

«Plug-ins pour communiquer avec les bases de données», à la page 338
Avec un plug-in Loader, une mappe ObjectGrid peut se comporter comme un cache pour les données généralement conservées dans un magasin persistant 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.

«Ecriture d'un chargeur», à la page 346
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.

«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.

«Utilisation d'un chargeur avec des mappes d'entité et des tuples», à la page 371
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.

«Ecriture d'un chargeur avec un contrôleur de préchargement de fragments réplique», à la page 376
Un chargeur avec un contrôleur de préchargement de fragments réplique est un chargeur qui implémente l'interface ReplicaPreloadController en plus de l'interface Loader.

«Chargeurs», à la page 90
Avec un plug-in Loader, une mappe de grille de données peut se comporter comme un cache pour les données généralement conservées dans un magasin persistant 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) peut également être utilisée comme source des données, ce qui permet de créer des caches basés sur un concentrateur à l'aide d'eXtreme Scale. Un chargeur peut lire et écrire des données vers un stockage persistant ou à partir de celui-ci.

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.

Référence associée:

«Remarques sur la programmation de chargeurs JPA», à la page 365
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.

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.

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, disponible dans le tutoriel 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 346.

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 381.

Référence associée:

«Remarques sur la programmation de chargeurs JPA», à la page 365
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.

Ecriture d'un chargeur avec un contrôleur de préchargement de fragments réplique

Un chargeur avec un contrôleur de préchargement de fragments réplique est un chargeur qui implémente l'interface ReplicaPreloadController en plus de l'interface Loader.

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 réplique 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 plus d'informations sur les chargeurs, voir les «Chargeurs», à la page 90informations sur les chargeurs dans *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 346.

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 réplique.

Référence associée:

«Remarques sur la programmation de chargeurs JPA», à la page 365
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.

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. Voir les «Stratégies de verrouillage», à la page 235informations relatives aux stratégies de verrouillage dans *Présentation du produit* pour plus d'informations.

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

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.

Emplacements de plug-in - Présentation

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 :

```
isExternalTransactionActive(Session session) booléen synchronisé public
```

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 381, 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 388.

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.

Configuration par programmation du plug-in WebSphereTransactionCallback

L'activation de WebSphereTransactionCallback dans la configuration d'ObjectGrid peut se faire par programmation ou par configuration XML. Le fragment de code suivant utilise l'application pour créer l'objet WebSphereTransactionCallback et l'ajouter à un ObjectGrid :

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
WebSphereTransactionCallback wsTxCallback= new WebSphereTransactionCallback ();
myGrid.setTransactionCallback(wsTxCallback);
```

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>
```

Programmation de l'utilisation de l'infrastructure OSGi

Vous pouvez démarrer les serveurs et les clients eXtreme Scale dans un conteneur OSGi qui permet d'ajouter et de mettre à jour dynamiquement les plug-ins eXtreme Scale dans l'environnement d'exécution.

Concepts associés:

«Présentation de la programmation du sérialiseur», à la page 309
Vous pouvez utiliser les plug-ins DataSerializer pour écrire des sérialiseurs optimisés pour stocker les objets Java et d'autres données binaires dans la grille. Le plug-in fournit également des méthodes que vous pouvez utiliser pour rechercher des attributs dans les données binaires sans avoir à augmenter l'ensemble de l'objet données.

Présentation de la sérialisation
Les données sont toujours exprimées, mais pas nécessairement stockées, comme les objets Java dans la grille de données. WebSphere eXtreme Scale utilise plusieurs processus Java pour sérialiser les données en convertissant les instances d'objet Java en octets, puis de nouveau en objets, si nécessaire, pour transférer les données entre les processus client et serveur.

Information associée:

Documentation de l'API DataSerializer

Génération de plug-ins dynamiques eXtreme Scale

WebSphere eXtreme Scale inclut les plug-ins ObjectGrid et BackingMap. Ces plug-ins sont implémentés dans Java et configurés en utilisant le fichier XML descripteur ObjectGrid. Pour créer un plug-in dynamique qui peut être mis à niveau dynamiquement, il doit connaître les événements de cycle de vie ObjectGrid et BackingMap, car il peut être nécessaire qu'il exécute des actions lors d'une mise à jour. L'amélioration d'un module d'extension avec des méthodes de rappel de cycle de vie, des programmes d'écoute d'événements ou les deux permet aux plug-ins d'effectuer ces actions au moment opportun.

Avant de commencer

Cette rubrique suppose que vous avez créé le plug-in approprié. Pour plus d'informations sur le développement de plug-ins eXtreme Scale, voir la rubrique API système et plug-ins.

Pourquoi et quand exécuter cette tâche

Tous les plug-ins eXtreme Scale s'appliquent à une instance BackingMap ou ObjectGrid. De nombreux plug-ins interagissent également avec d'autres plug-ins. Par exemple, un chargeur et un plug-in TransactionCallback fonctionnent ensemble pour interagir correctement avec une transaction de base de données et les divers appels JDBC de base de données. Certains modules d'extension peut également s'avérer nécessaires pour mettre en mémoire cache les données de configuration des autres plug-ins pour améliorer les performances.

Les plug-ins BackingMapLifecycleListener et ObjectGridLifecycleListener fournissent des opérations de cycle de vie pour les instances BackingMap et ObjectGrid correspondantes. Ce processus permet aux plug-ins d'être avertis lorsque le parent BackingMap ou ObjectGrid et ses plug-ins correspondants peuvent être modifiés. Les plug-ins BackingMap implémentent l'interface BackingMapLifecyleListener et les plug-ins ObjectGrid implémentent l'interface ObjectGridLifecycleListener. Ces plug-ins sont appelés automatiquement lorsque le cycle de vie du parent BackingMap ou ObjectGrid change. Pour plus d'informations sur les plug-ins de cycle de vie, voir la rubrique «Gestion des cycles de vie du plug-in», à la page 298.

Vous améliorerez les ensembles en utilisant les méthodes de cycle de vie ou les programmes d'écoute dans les tâches communes suivantes :

- Démarrage et arrêt des ressources, telles que les unités d'exécution ou les abonnés de messagerie.
- Indication qu'une notification se produit lorsque des plug-ins homologues ont été mis à jour, ce qui permet d'accéder directement au plug-in et de détecter les modifications.

Lorsque vous accédez directement à un autre plug-in, accédez-y via le conteneur OSGi pour que tous les composants du système fassent référence au plug-in correct. Si, par exemple, un composant dans l'application fait directement référence, ou met en mémoire cache, une instance d'un plug-in, il conserve sa référence à cette version du plug-in, même lorsque le plug-in est mis à jour dynamiquement. Ce comportement peut causer des problèmes au niveau de l'application ainsi que des fuites de mémoire. Par conséquent, écrivez le code qui dépend des plug-ins dynamiques qui obtient sa référence en utilisant OSGi, la sémantique getService(). Si l'application doit mettre en cache un ou plusieurs plug-ins, elle écoute les événements de cycle de vie en utilisant les interfaces ObjectGridLifecycleListener et BackingMapLifecycleListener. L'application doit pouvoir également régénérer sa mémoire cache lorsque cela est nécessaire en sécurisant les unités d'exécution.

Tous les plug-ins eXtreme Scale utilisés avec OSGi doivent également implémenter l'interface BackingMapPlugin ou ObjectGridPlugin correspondante. Les nouveaux plug-ins, tels que l'interface MapSerializerPlugin, appliquent cette pratique. Ces interfaces fournissent à l'environnement d'exécution eXtreme Scale et OSGi une interface cohérente pour injecter l'état dans le plug-in et contrôler son cycle de vie.

Utilisez cette tâche pour spécifier qu'une notification se produit lorsque des plug-ins homologues sont mis à jour. Vous pouvez créez une fabrique d'écoute qui produit une instance de programme d'écouter.

Procédure

- Mettez à jour la classe de plug-in ObjectGrid pour implémenter l'interface ObjectGridPlugin. Cette interface contient des méthodes qui permettent à eXtreme Scale d'initialiser et définir l'instance ObjectGrid et détruire le plug-in. Reportez-vous à l'exemple de code suivant :

```
package com.mycompany;
import com.ibm.websphere.objectgrid.plugins.ObjectGridPlugin;
...

public class MyTranCallback implements TransactionCallback, ObjectGridPlugin {

    private ObjectGrid og = null;

    private enum State {
        NEW, INITIALIZED, DESTROYED
    }

    private State state = State.NEW;

    public void setObjectGrid(ObjectGrid grid) {
        this.og = grid;
    }

    public ObjectGrid getObjectGrid() {
        return this.og;
    }
    void initialize() {
        // Traiter l'initialisation de plug-in ici.  Cela peut être appelé par
        // eXtreme Scale et non pas par le gestionnaire de bean OSGi.
        state = State.INITIALIZED;
    }
    boolean isInitialized() {
        return state == State.INITIALIZED;
    }

    public void destroy() {
        // Détruire le plug-in et libérer les ressources.  Cela
```

```
        peut être appelé par le gestionnaire de bean OSGi ou eXtreme Scale.
        state = State.DESTROYED;
    }

    public boolean isDestroyed() {
        return state == State.DESTROYED;
    }
}
```

- Mettez à jour la classe de plug-in ObjectGrid pour implémenter l'interface ObjectGridLifecycleListener. Reportez-vous à l'exemple de code suivant :

```
package com.mycompany;
import com.ibm.websphere.objectgrid.plugins.ObjectGridLifecycleListener;
      import com.ibm.websphere.objectgrid.plugins.ObjectGridLifecycleListener.LifecycleEvent;
      ...

      public class MyTranCallback implements TransactionCallback, ObjectGridPlugin, ObjectGridLifecycleListener{
          public void objectGridStateChanged(LifecycleEvent event) {
              switch(event.getState()) {
              case NEW:
              case DESTROYED:
              case DESTROYING:
              case INITIALIZING:
                  break;
              case INITIALIZED:
                  // Rechercher un chargeur ou un plug-in MapSerializerPlugin en utilisant
                  // OSGi ou directement depuis l'instance ObjectGrid.
                  lookupOtherPlugins()
                  break;
              case STARTING:
              case PRELOAD:
                  break;
              case ONLINE:
                  if (event.isWritable()) {
                      startupProcessingForPrimary();
                  } else {
                      startupProcessingForReplica();
                  }
                  break;
              case QUIESCE:
                  if (event.isWritable()) {
                      quiesceProcessingForPrimary();
                  } else {
                      quiesceProcessingForReplica();
                  }
                  break;
              case OFFLINE:
                  shutdownShardComponents();
                  break;
              }
          }
          ...
      }
```

- Mettez à jour un plug-in BackingMap. Mettez à jour la classe de plug-in BackingMap pour implémenter l'interface de plug-in BackingMap. Cette interface inclut des méthodes qui permettent à eXtreme Scale d'initialiser, définir l'instance BackingMap et détruire le plug-in. Reportez-vous à l'exemple de code suivant :

```
package com.mycompany;
import com.ibm.websphere.objectgrid.plugins.BackingMapPlugin;
...

public class MyLoader implements Loader, BackingMapPlugin {

    private BackingMap bmap = null;

    private enum State {
        NEW, INITIALIZED, DESTROYED
    }

    private State state = State.NEW;

    public void setBackingMap(BackingMap map) {
        this.bmap = map;
    }

    public BackingMap getBackingMap() {
        return this.bmap;
    }
    void initialize() {
        // Traiter l'initialisation de plug-in ici.  Cela peut être appelé par
        // eXtreme Scale et non pas par le gestionnaire de bean OSGi.
        state = State.INITIALIZED;
    }
    boolean isInitialized() {
        return state == State.INITIALIZED;
    }
```

```
    public void destroy() {
        // Détruire le plug-in et libérer les ressources.  Cela
        peut être appelé par le gestionnaire de bean OSGi ou eXtreme Scale.
        state = State.DESTROYED;
    }

    public boolean isDestroyed() {
        return state == State.DESTROYED;
    }
}
```

- Mettez à jour la classe de plug-in BackingMap pour implémenter une interface BackingMapLifecycleListener. Reportez-vous à l'exemple de code suivant :

```
package com.mycompany;

    import com.ibm.websphere.objectgrid.plugins.BackingMapLifecycleListener;
    import com.ibm.websphere.objectgrid.plugins.BackingMapLifecycleListener.LifecycleEvent;
    ...

    public class MyLoader implements Loader, ObjectGridPlugin, ObjectGridLifecycleListener{
        ...
        public void backingMapStateChanged(LifecycleEvent event) {
            switch(event.getState()) {
            case NEW:
            case DESTROYED:
            case DESTROYING:
            case INITIALIZING:
                break;
            case INITIALIZED:
                // Rechercher un plug-in MapSerializerPlugin en utilisant
                // OSGi ou directement depuis l'instance ObjectGrid.
                lookupOtherPlugins()
                break;
            case STARTING:
            case PRELOAD:
                break;
            case ONLINE:
                if (event.isWritable()) {
                    startupProcessingForPrimary();
                } else {
                    startupProcessingForReplica();
                }
                break;
            case QUIESCE:
                if (event.isWritable()) {
                    quiesceProcessingForPrimary();
                } else {
                    quiesceProcessingForReplica();
                }
                break;
            case OFFLINE:
                shutdownShardComponents();
                break;
            }
        }
        ...
    }
```

Résultats

En implémentant l'interface ObjectGridPlugin ou BackingMapPlugin, eXtreme Scale peut contrôler le cycle de vie du plug-in au moment opportun.

En implémentant l'interface ObjectGridLifecycleListener ou BackingMapLifecycleListener, le plug-in est enregistré automatiquement comme programme d'écoute des événements de cycle de vie ObjectGrid ou BackingMap associés. L'événement INITIALIZING permet de signaler que tous les plug-ins ObjectGrid et BackingMap ont été initialisés et peuvent être recherchés et utilisés. L'événement ONLINE est utilisé pour signaler que ObjectGrid est en ligne et prêt à commencer le traitement des événements.

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.

Tâches associées:

«Traitement des problèmes des chargeurs», à la page 515
Utilisez ces informations pour traiter les problèmes liés aux chargeurs de base de données.

Configuration des chargeurs JPA
Un chargeur Java Persistence API (JPA) est une implémentation de plug-in qui utilise JPA pour interagir avec la base de données.

Chargeurs 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).

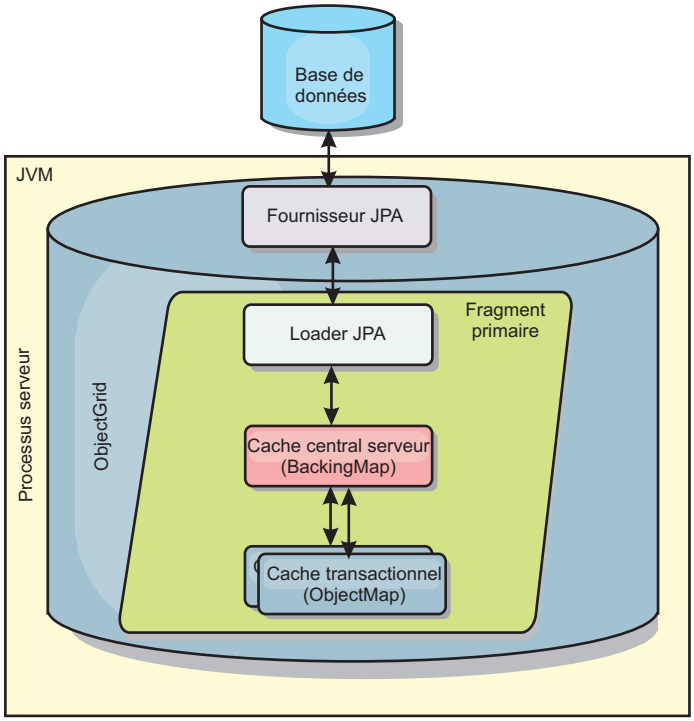


Figure 28. Architecture du chargeur JPA

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, voir «Plug-in JPAEntityLoader», à la page 368.

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.

Configuration et programmation

Si vous utilisez des chargeurs JPA dans un environnement multi-maître, voir «Remarques sur les chargeurs dans une topologie multimaître», à la page 106. Pour plus d'informations sur la configuration des chargeurs JPA, voir les informations sur les chargeurs JPA dans *Guide d'administration*. Pour plus d'informations sur la programmation des chargeurs JPA, reportez-vous au *Guide de programmation*.

Développement des chargeurs JPA basés sur le client

Vous pouvez implémenter le préchargement et le rechargement des données dans votre application avec un utilitaire JPA (Java Persistence API). Cette fonction peut simplifier le chargement des mappes lorsque les requêtes à la base de données ne peuvent pas être partitionnées.

Avant de commencer

- Vous devez utiliser un fournisseur JPA avec une base de données prise en charge.
- Avant de précharger ou de recharger des mappes, vous devez définir l'état de disponibilité `PRELOAD`, ObjectGrid. Vous pouvez effectuer cette opération avec la méthode setObjectGridState de l'interface StateManager. L'interface StateManager empêche les autres clients d'accéder à l'ObjectGrid, tant qu'il n'est pas en ligne. Après avoir préchargé ou rechargé la mappe, vous pouvez redéfinir l'état ONLINE.
- 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. Pour plus d'informations, voir Gestion de la disponibilité ObjectGrid.

Pourquoi et quand exécuter cette tâche

Lorsque vous exécutez une opération de préchargement ou rechargement sur la mappe, les actions suivantes se produisent :

1. L'action initiale qui est exécutée varie selon que vous exécutez un préchargement ou un rechargement.
   - **Préchargement de l'opération:** la mappe à précharger est effacée. Pour une mappe d'entité, si une relation est configurée comme cascade-remove, les mappes associées sont effacées.
   - **Opération de rechargement :** la requête fournie est exécutée sur la mappe et les résultats sont invalidés. Pour une mappe d'entité, si une relation est configurée avec l'option **CascadeType.INVALIDATE**, les entités associées sont également invalidées depuis leurs mappes.
2. Exécutez la requête sur JPA pour les entités d'un lot.
3. Pour chaque lot, une liste de clés et la liste de valeurs de chaque partition est générée.
4. Pour chaque partition, l'agent de grille de données est appelé pour insérer ou mettre à jour les données côté serveur directement s'il s'agit d'un client eXtreme Scale. Si la grille de données est une instance locale, les données dans les mappes sont directement insérées ou mises à jour.

Concepts associés:

«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.

Référence associée:

«Exemple : Préchargement d'une mappe avec l'interface ClientLoader», à la page 401
Vous pouvez précharger une mappe pour la remplir avec ses données avant que les clients commencent à accéder à la mappe.

«Exemple : rechargement d'une mappe avec l'interface ClientLoader», à la page 402
Recharger une mappe revient à précharger la mappe, sauf que l'argument **isPreload** a la valeur false dans la méthode ClientLoader.load.

«Exemple : appel d'un chargeur de client», à la page 403
Vous pouvez utiliser la méthode preload dans l'interface Loader pour appeler un chargeur de client.

Information associée:

Interface ClientLoader

Interface StateManager

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 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.

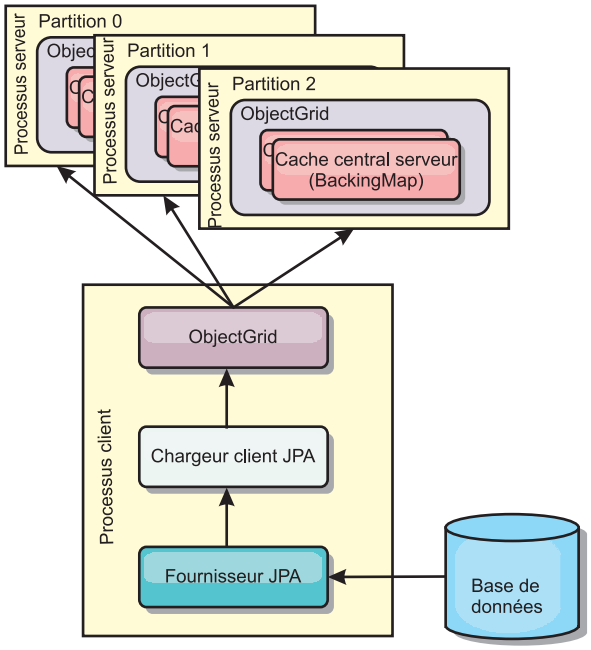


Figure 29. Chargeur client qui utilise l'implémentation JPA pour charger ObjectGrid

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 10. 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 plus d'informations sur la configuration de l'utilitaire de préchargement PJA client, voir les «Développement des chargeurs JPA basés sur le client», à la page 398informations dans *Guide de programmation*

Tâches associées:

«Développement des chargeurs JPA basés sur le client», à la page 398
Vous pouvez implémenter le préchargement et le rechargement des données dans votre application avec un utilitaire JPA (Java Persistence API). Cette fonction peut simplifier le chargement des mappes lorsque les requêtes à la base de données ne peuvent pas être partitionnées.

Référence associée:

«Exemple : Préchargement d'une mappe avec l'interface ClientLoader»
Vous pouvez précharger une mappe pour la remplir avec ses données avant que les clients commencent à accéder à la mappe.

«Exemple : rechargement d'une mappe avec l'interface ClientLoader», à la page 402
Recharger une mappe revient à précharger la mappe, sauf que l'argument **`isPreload`** a la valeur false dans la méthode ClientLoader.load.

«Exemple : appel d'un chargeur de client», à la page 403
Vous pouvez utiliser la méthode preload dans l'interface Loader pour appeler un chargeur de client.

Information associée:

Interface ClientLoader

Interface StateManager

Exemple : Préchargement d'une mappe avec l'interface ClientLoader

Vous pouvez précharger une mappe pour la remplir avec ses données avant que les clients commencent à accéder à la mappe.

Exemple de préchargement basé sur le client

L'exemple de fragment de code ci-après illustre un chargement de client simple. 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. Le paramètre **isPreload** dans la méthode load a la valeur `true`.

```
// 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", "customerPU", null,
    null, null, null, true, null);

// Réaffectez à l'état ObjectGrid la valeur ONLINE
stateMgr.setObjectGridState(AvailabilityState.ONLINE, objectGrid);
```

Concepts associés:

«Présentation de l'utilitaire de préchargement client JPA», à la page 399
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.

Tâches associées:

«Développement des chargeurs JPA basés sur le client», à la page 398
Vous pouvez implémenter le préchargement et le rechargement des données dans votre application avec un utilitaire JPA (Java Persistence API). Cette fonction peut simplifier le chargement des mappes lorsque les requêtes à la base de données ne peuvent pas être partitionnées.

Information associée:

Interface ClientLoader

Interface StateManager

Exemple : rechargement d'une mappe avec l'interface ClientLoader

Recharger une mappe revient à précharger la mappe, sauf que l'argument **isPreload** a la valeur false dans la méthode ClientLoader.load.

Exemple de rechargement basé sur le client

L'exemple suivant montre comment recharger des mappes. Cet exemple diffère de l'exemple de préchargement dans la mesure ou un loadSql et ses paramètres sont fournis. Cet exemple recharge uniquement les données client avec un ID compris entre `1000` et `2000`. Le paramètre **isPreload** dans la méthode de chargement a la valeur `false`.

```
// 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);
```

A faire : 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 : une fois pour invalider les entités ObjectGrid correpondantes et une autre fois pour charger les entités JPA correspondantes.

Concepts associés:

«Présentation de l'utilitaire de préchargement client JPA», à la page 399
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.

Tâches associées:

«Développement des chargeurs JPA basés sur le client», à la page 398
Vous pouvez implémenter le préchargement et le rechargement des données dans votre application avec un utilitaire JPA (Java Persistence API). Cette fonction peut simplifier le chargement des mappes lorsque les requêtes à la base de données ne peuvent pas être partitionnées.

Information associée:

Interface ClientLoader

Interface StateManager

Exemple : appel d'un chargeur de client

Vous pouvez utiliser la méthode preload dans l'interface Loader pour appeler un chargeur de client.

Utilisez la méthode preload dans l'interface Loader pour appeler un chargeur de client :

```
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 *Présentation du produit*.

Exemple : appel d'un chargeur de client avec la méthode preloadMap

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. Vous devez appeler le chargeur de client dans une et une seule partition.

```
void preloadMap (Session session, BackingMap backingMap) throws LoaderException {

....
 ObjectGrid objectGrid = 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 c = ClientLoaderFactory.getClientLoader();
    // Apelle le chargeur client pour charger les données
    try {
        c.load(objectGrid, "CUSTOMER", "customerPU",
     null, entityClass, null, null, true, null);
    } catch (ObjectGridException e) {
        LoaderException le = new LoaderException("Exception caught in ObjectMap " +
     ogName + "." + mapName);
        le.initCause(e);
        throw le;
    }
 }
}
```

A faire : Affectez à l'attribut "preloadMode" la valeur `true` pour que la méthode preload s'exécute de manière asynchrone. Autrement, la méthode preload bloque l'activation de l'instance ObjectGrid.

Concepts associés:

«Présentation de l'utilitaire de préchargement client JPA», à la page 399
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.

Tâches associées:

«Développement des chargeurs JPA basés sur le client», à la page 398
Vous pouvez implémenter le préchargement et le rechargement des données dans votre application avec un utilitaire JPA (Java Persistence API). Cette fonction peut simplifier le chargement des mappes lorsque les requêtes à la base de données ne peuvent pas être partitionnées.

Information associée:

Interface ClientLoader

Interface StateManager

Exemple : création d'un chargeur JPA de client personnalisé

La méthode ClientLoader.load dans l'interface Loader offre une fonction de chargement client qui convient dans la plupart des cas. Toutefois, si vous voulez charger les données sans la méthode ClientLoader.load, vous pouvez implémenter votre propre utilitaire de préchargement.

Modèle de chargeur personnalisé

Utilisez le modèle suivant pour développer le chargeur :

```
// 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
...<your preload utility implementation>...

// Réaffectez à l'état ObjectGrid la valeur ONLINE
stateMgr.setObjectGridState(AvailabilityState.ONLINE, objectGrid);
```

Développement d'un chargeur JPA-basé sur le client à l'aide d'un agent DataGrid

Le chargement depuis le client en utilisant 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 pour que les agents DataGrid dans plusieurs partitions s'exécutent en parallèle afin d'améliorer encore les performances.

Pourquoi et quand exécuter cette tâche

Pour plus d'informations sur l'agent DataGrid, voir «API DataGrid et partitionnement», à la page 260.

Une fois que vous avez créé l'implémentation de préchargement des données, vous pouvez créer un chargeur générique pour effectuer les tâches suivantes :

- Interroger les données de la base de données par lots.
- Générer une liste de clés et une liste de valeurs pour chaque partition.
- 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

Le code ci-dessous montre comment effectuer le chargement en utilisant 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;
    }
}
```

Exemple: Utilisation du plug-in Hibernate pour précharger les données dans le cache ObjectGrid

Vous pouvez utiliser la méthode preload de la classeObjectGridHibernateCacheProvider pour précharger les données dans le cache d'ObjectGrid pour une classe d'entité.

Exemple : utilisation de la classe EntityManagerFactory

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("testPU");
ObjectGridHibernateCacheProvider.preload("objectGridName", emf, TargetEntity.class, 100, 100);
```

Important : Par défaut, les entités ne font pas partie de la mémoire cache de second niveau. Dans les classes d'entité qui nécessitent la mise en cache, ajoutez l'annotation @cache. Exemple de programme :

```
import org.hibernate.annotations.Cache;
import org.hibernate.annotations.CacheConcurrencyStrategy;
@Entity
@Cache(usage=CacheConcurrencyStrategy.TRANSACTIONAL)
public class HibernateCacheTest { ... }
```

Vous pouvez remplacer cette valeur par défaut en définissant l'élément shared-cache-mode dans le fichier `persistence.xml` ou en utilisant la propriété javax.persistence.sharedCache.mode.

Exemple : utilisation de la classe SessionFactory

```
org.hibernate.cfg.Configuration cfg = new Configuration();
// utilisez la méthode de configuration addResource, addClass et setProperty pour préparer
// la configuration requise pour créer SessionFactor
SessionFactory sessionFactory= cfg.buildSessionFactory();
ObjectGridHibernateCacheProvider.preload("objectGridName", sessionFactory,
TargetEntity.class, 100, 100);
```

Remarque :

1. Dans un système réparti, ce mécanisme de préchargement peut uniquement être appelé à partir d'une machine virtuelle Java. Le mécanisme de préchargement ne peut pas être exécuté à partir de plusieurs machines virtuelles Java.
2. Avant d'exécuter le préchargement, vous devez initialiser le cache eXtreme Scale en créant EntityManager en utilisant EntityManagerFactory pour créer tous les BackingMaps correspondants. Autrement, le préchargement force l'initialisation du cache avec une mappage de sauvegarde par défaut pour prendre en charge toutes les entités. Une seule mappe de sauvegarde est ainsi partagée par toutes les entités.

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 Configuration d'un programme de mise à jour de données JPA en fonction de la date/heure les informations relatives à la configuration d'un programme de mise à jour JPA basé sur le temps dans *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 412.

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;
}
```

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.

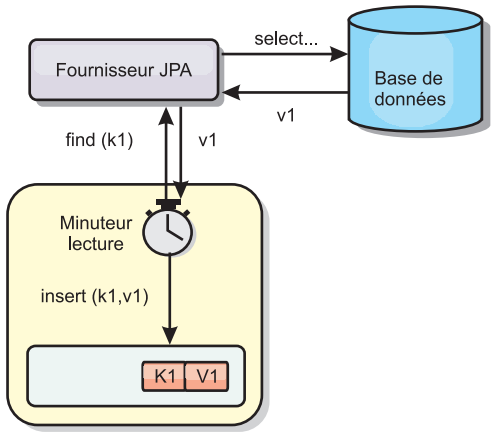


Figure 30. Actualisation régulière

>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 JPA de mise à jour des données basé sur le temps, voir les informations dans le *Guide d'administration*.

Développement d'applications avec l'infrastructure Spring

Apprenez comment intégrer vos applications eXtreme Scale à Spring.

Concepts associés:

«Présentation de l'infrastructure Spring», à la page 120
Spring est une infrastructure de développement d'applications Java. WebSphere eXtreme Scale fournit le support permettant à Spring de gérer les transactions et de configurer les clients et serveurs constituant votre grille de données en mémoire déployée.

«Prise en charge des beans d'extension Spring et des espaces de noms», à la page 420
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.

Référence associée:

«Beans d'extension gérés par Spring», à la page 419
Vous pouvez déclarer des objets POJO (plain old Java objects) à utiliser comme point d'extension dans le fichier `objectgrid.xml`. 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. WebSphere eXtreme Scale peut désormais déléguer à Spring le rôle de fabrique de beans pour obtenir des instances de ces objets plug-in.

Fichier XML du descripteur de Spring
Utilisez un fichier XML de descripteur de Spring pour configurer et intégrer eXtreme Scale à Spring.

Fichier `objectgrid.xsd` de Spring
Utilisez le fichier `objectgrid.xsd` de Spring pour intégrer eXtreme Scale à Spring afin de gérer les transactions eXtreme Scale et configurer les clients et serveurs.

Présentation de l'infrastructure Spring

Spring est une infrastructure de développement d'applications Java. WebSphere eXtreme Scale fournit le support permettant à Spring de gérer les transactions et de configurer les clients et serveurs constituant votre grille de données en mémoire déployée.

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 peut utiliser des implémentations différentes. WebSphere eXtreme Scale fournit une intégration du gestionnaire de transactions permettant à Spring de gérer les cycles de vie des transactions ObjectGrid. Voir les informations sur les transactions natives dans le *Guide de programmation* pour plus d'informations.

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.

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 pour les beans avec un ID ou des ID correspondant à cette définition de bean dans un même fragment amènent le conteneur Spring à retourner une instancce de bean spécifique.

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

stocke son état de session dans une session HTTP par défaut. Si une application Web utilise eXtreme Scale pour la gestion de session, Spring automatiquement stocke l'état avec eXtreme Scale. e plus, la tolérance aux pannes est activée de la même manière que la session.

Consultez les informations de gestion de session HTTP dans le *Présentation du produit* pour plus d'informations.

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 Java EE qui s'exécute dans un WebSphere Application Server Network Deployment étendu WebSphere Extended Deployment, placez le fichier `spring.jar` et ses fichiers associés dans les modules EAR (Enterprise Archive). Vous devez également placer le fichier `ogspring.jar` au même emplacement.

Tâches associées:

«Développement d'applications avec l'infrastructure Spring», à la page 414
Apprenez comment intégrer vos applications eXtreme Scale à Spring.

«Démarrage d'un serveur de conteneur avec Spring», à la page 423
Vous pouvez démarrer un serveur de conteneur à l'aide de beans d'extension gérés Spring et la prise en charge d'espace de nom.

«Gestion des transactions avec Spring»
est une infrastructure de développement d'applications Java communément utilisée. WebSphere eXtreme Scale permet à Spring de gérer les transactions eXtreme Scale et de configurer les clients et les serveurs eXtreme Scale.

Référence associée:

«Beans d'extension gérés par Spring», à la page 419
Vous pouvez déclarer des objets POJO (plain old Java objects) à utiliser comme point d'extension dans le fichier `objectgrid.xml`. 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. WebSphere eXtreme Scale peut désormais déléguer à Spring le rôle de fabrique de beans pour obtenir des instances de ces objets plug-in.

Fichier XML du descripteur de Spring
Utilisez un fichier XML de descripteur de Spring pour configurer et intégrer eXtreme Scale à Spring.

Fichier `objectgrid.xsd` de Spring
Utilisez le fichier `objectgrid.xsd` de Spring pour intégrer eXtreme Scale à Spring afin de gérer les transactions eXtreme Scale et configurer les clients et serveurs.

Gestion des transactions avec Spring

est une infrastructure de développement d'applications Java communément utilisée. WebSphere eXtreme Scale permet à Spring de gérer les transactions eXtreme Scale et de configurer les clients et les serveurs eXtreme Scale.

Pourquoi et quand exécuter cette tâche

L'infrastructure Spring peut être étroitement intégrée à eXtreme Scale, comme cela est expliqué dans les sections suivantes.

Procédure

- **Transactions natives :** Spring fournit des transactions gérées par conteneur dans le style d'un serveur d'applications Java Platform, Enterprise Edition, mais présente l'avantage que le mécanisme Spring peut avoir 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.
- **Gestionnaire de transaction :** pour utiliser Spring, eXtreme Scale fournit une implémentation d'un gestionnaire Spring PlatformTransactionManager. 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 :** 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.
- **Définition de l'instance ObjectGrid pour une unité d'exécution :** une seule machine virtuelle Java (JVM) 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.

  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 est implémentée par SpringLocalTxManager ObjectGrid Platform Transaction Manager et a 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().

  Restriction : WebSphere eXtreme Scale ne prend pas en charge JTA ou la validation en deux phases pour diverses raisons et notamment pour permettre l'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.

Concepts associés:

«Présentation de l'infrastructure Spring», à la page 120
Spring est une infrastructure de développement d'applications Java. WebSphere eXtreme Scale fournit le support permettant à Spring de gérer les transactions et de configurer les clients et serveurs constituant votre grille de données en mémoire déployée.

«Prise en charge des beans d'extension Spring et des espaces de noms», à la page 420
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.

Référence associée:

«Beans d'extension gérés par Spring»
Vous pouvez déclarer des objets POJO (plain old Java objects) à utiliser comme point d'extension dans le fichier `objectgrid.xml`. 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. WebSphere eXtreme Scale peut désormais déléguer à Spring le rôle de fabrique de beans pour obtenir des instances de ces objets plug-in.

Fichier XML du descripteur de Spring
Utilisez un fichier XML de descripteur de Spring pour configurer et intégrer eXtreme Scale à Spring.

Fichier `objectgrid.xsd` de Spring
Utilisez le fichier `objectgrid.xsd` de Spring pour intégrer eXtreme Scale à Spring afin de gérer les transactions eXtreme Scale et configurer les clients et serveurs.

Beans d'extension gérés par Spring

Vous pouvez déclarer des objets POJO (plain old Java objects) à utiliser comme point d'extension dans le fichier `objectgrid.xml`. 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. WebSphere 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, les objets POJO doivent être accessibles au reste de l'application.

Une application peut enregistrer une instance de fabrique de beans Spring à utiliser pour un ObjectGrid défini par nom. L'application crée une instance de BeanFactory ou un contexte d'application Spring et l'enregistre dans ObjectGrid en utilisant la méthode statique suivante :

```
void registerSpringBeanFactoryAdapter(String objectGridName, Object springBeanFactory)
```

La méthode précédente s'applique lorsque eXtreme Scale détecte un bean d'extension dont le nom de classe className commence par le préfixe {spring}. Ce type de bean d'extension, qui peut être un ObjectTransformer, Loader, TransactionCallback, etc., utilise le reste du nom comme nom de bean Spring. Ensuite, il obtient l'instance de bean à l'aide de la fabrique de beans Spring.

L'environnement de déploiement eXtreme Scale peut également créer une fabrique de beans Spring à partir d'un fichier de configuration XML Spring par défaut. Si aucune fabrique de beans n'a été enregistrée pour un ObjectGrid donné, le déploiement recherche systématiquement le fichier XML "/<ObjectGridName>_spring.xml". Par exemple, si votre grille de données s'appelle GRID, le fichier XML s'appelle "/GRID_spring.xml' et apparaît dans le chemin d'accès aux classes dans le package racine. ObjectGrid construit un contexte d'application ApplicationContext en utilisant le fichier "/<ObjectGridName>_spring.xml et des beans depuis la fabrique de beans.

Voici un exemple de nom de classe :

```
"{spring}MyLoaderBean"
```

L'utilisation du nom de classe précédent, eXtreme Scale peut utiliser Spring pour rechercher le bean "MyLoaderBean". Vous pouvez spécifier des objets POJO gérés par Spring pour n'importe quel point d'extension si la fabrique de beans a été enregistrée. Les extensions Spring se trouvent dans le fichier ogspring.jar. Ce fichier JAR doit se trouver dans le chemin d'accès aux classes du support Spring. Si une application J2EE exécutée dans WebSphere Application Server Network Deployment étendu avec WebSphere Extended Deployment, vous devez placer l'application dans le fichier spring.jar et ses fichiers associés dans les modules EAR. Le fichier ogspring.jar doit également être placé au même endroit.

Concepts associés:

«Présentation de l'infrastructure Spring», à la page 120
Spring est une infrastructure de développement d'applications Java. WebSphere eXtreme Scale fournit le support permettant à Spring de gérer les transactions et de configurer les clients et serveurs constituant votre grille de données en mémoire déployée.

«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.

Tâches associées:

«Développement d'applications avec l'infrastructure Spring», à la page 414
Apprenez comment intégrer vos applications eXtreme Scale à Spring.

«Démarrage d'un serveur de conteneur avec Spring», à la page 423
Vous pouvez démarrer un serveur de conteneur à l'aide de beans d'extension gérés Spring et la prise en charge d'espace de nom.

«Gestion des transactions avec Spring», à la page 416
est une infrastructure de développement d'applications Java communément utilisée. WebSphere eXtreme Scale permet à Spring de gérer les transactions eXtreme Scale et de configurer les clients et les serveurs eXtreme Scale.

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 scénarios, vous devez utiliser Spring pour configurer un plug-in, comme dans l'exemple suivant :

```
<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 TransactionCallback intégrée, la classe com.ibm.websphere.objectgrid.jpa.JPATxCallback, est configurée comme classe TransactionCallback. Cette classe est configurée avec la propriété **persistenceUnitName**, comme le montre 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 la grille de données s'appelle "Grid", le fichier XML s'appelle `/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 la description détaillée du fichier XSD et des beans définis dans le fichier XSD, voir Fichier XML du descripteur de Springles informations sur le fichier descripteur Spring dans *Guide d'administration*.

Utilisez l'exemple 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>
```

Ici, notez qu'au lieu de définir la classe com.ibm.websphere.objectgrid.jpa.JPATxCallback comme dans l'exemple précédent, nous utilisons directement le bean objectgrid:JPATxCallback prédéfini. Comme vous pouvez le voir, cette configuration est moins détaillée et facilite la détection d'erreurs.

Pour la description de l'utilisation des beans Spring, consultez «Démarrage d'un serveur de conteneur avec Spring».

Tâches associées:

«Développement d'applications avec l'infrastructure Spring», à la page 414
Apprenez comment intégrer vos applications eXtreme Scale à Spring.

«Démarrage d'un serveur de conteneur avec Spring»
Vous pouvez démarrer un serveur de conteneur à l'aide de beans d'extension gérés Spring et la prise en charge d'espace de nom.

«Gestion des transactions avec Spring», à la page 416
est une infrastructure de développement d'applications Java communément utilisée. WebSphere eXtreme Scale permet à Spring de gérer les transactions eXtreme Scale et de configurer les clients et les serveurs eXtreme Scale.

Référence associée:

«Beans d'extension gérés par Spring», à la page 419
Vous pouvez déclarer des objets POJO (plain old Java objects) à utiliser comme point d'extension dans le fichier `objectgrid.xml`. 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. WebSphere eXtreme Scale peut désormais déléguer à Spring le rôle de fabrique de beans pour obtenir des instances de ces objets plug-in.

Fichier XML du descripteur de Spring
Utilisez un fichier XML de descripteur de Spring pour configurer et intégrer eXtreme Scale à Spring.

Fichier `objectgrid.xsd` de Spring
Utilisez le fichier `objectgrid.xsd` de Spring pour intégrer eXtreme Scale à Spring afin de gérer les transactions eXtreme Scale et configurer les clients et serveurs.

Démarrage d'un serveur de conteneur avec Spring

Vous pouvez démarrer un serveur de conteneur à l'aide de beans d'extension gérés Spring et la prise en charge d'espace de nom.

Pourquoi et quand exécuter cette tâche

Avec plusieurs fichiers XML configurés pour Spring, vous pouvez démarrer des serveurs conteneurs eXtreme Scale de base.

Procédure

1. **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>
   ```

2. **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>
   ```

3. **Fichier XML ObjectGrid Spring :**

   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 s'appelle `Grid_spring.xml`. Notez que deux schémas sont inclus dans le fichier XML : `spring-beans-2.0.xsd` permet d'utiliser les beans gérés 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>
```

Ce fichier XML Spring contient six beans définis :

a. *objectgrid:register* : enregistre la classe de bean par défaut pour l'ObjectGrid "Grid".

b. *objectgrid:server* : définit un serveur ObjectGrid nommé "server". Ce serveur fournit également un service de catalogue, car un bean objectgrid:catalog y est imbriqué.

c. *objectgrid:catalog* : définit un point de contact de service de catalogue ObjectGrid, qui est défini sur "localhost:2809".

d. *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.

e. *objectgrid:LRUEvictor* : définit un expulseur LRUEvictor avec le nombre de files d'attente LRU à utiliser pour le définir sur 31.

f. *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.

4. **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();
    }
}
```

Concepts associés:

«Présentation de l'infrastructure Spring», à la page 120
Spring est une infrastructure de développement d'applications Java. WebSphere eXtreme Scale fournit le support permettant à Spring de gérer les transactions et de configurer les clients et serveurs constituant votre grille de données en mémoire déployée.

«Prise en charge des beans d'extension Spring et des espaces de noms», à la page 420
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.

Référence associée:

«Beans d'extension gérés par Spring», à la page 419
Vous pouvez déclarer des objets POJO (plain old Java objects) à utiliser comme point d'extension dans le fichier `objectgrid.xml`. 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. WebSphere eXtreme Scale peut désormais déléguer à Spring le rôle de fabrique de beans pour obtenir des instances de ces objets plug-in.

Fichier XML du descripteur de Spring
Utilisez un fichier XML de descripteur de Spring pour configurer et intégrer eXtreme Scale à Spring.

Fichier `objectgrid.xsd` de Spring
Utilisez le fichier `objectgrid.xsd` de Spring pour intégrer eXtreme Scale à Spring afin de gérer les transactions eXtreme Scale et configurer les clients et serveurs.

Configuration des clients dans l'infrastructure Spring

Vous pouvez remplacer les paramètres ObjectGrid côté client avec l'infrastructure Spring

Pourquoi et quand exécuter cette tâche

. L'exemple de fichier XML suivant montre comment générer un élément ObjectGridConfiguration et l'utiliser pour remplacer certains paramètres côté client. Vous pouvez créer une configuration similaire en utilisant la configuration à l'aide d'un programme ou en configurant le fichier XML de descripteur d'ObjectGrid.

Procédure

1. Créez un fichier XML pour configurer des clients avec l'infrastructure 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>
```

2. Chargez le fichier XML que vous avez créé et générez l'ObjectGrid.

```
BeanFactory beanFactory = new XmlBeanFactory(newUrlResource
  ("file:test/companyGridSpring.xml"));
ObjectGrid companyGrid = (ObjectGrid) beanFactory.getBean("companyGrid");
```

Voir «Présentation de l'infrastructure Spring», à la page 120 pour plus d'informations sur la création d'un fichier descripteur XML.

Chapitre 6. Optimisation des performances

Vous pouvez optimiser les paramètres dans l'environnement pour augmenter les performances générales de votre environnement WebSphere eXtreme Scale.

Optimisation de l'agent de dimensionnement du cache pour des estimations précises de l'utilisation de la mémoire

WebSphere eXtreme Scale prend en charge le dimensionnement de la consommation de la mémoire par les instances de BackingMap dans les grilles de données réparties. Le dimensionnement de la consommation de mémoire de la taille n'est pas pris en charge pour les instances de grilles de données locales. La valeur signalée par WebSphere eXtreme Scale pour une mappe donnée est très proche de la valeur signalée par l'analyse des clichés de segments. Si l'objet de mappe est complexe, le dimensionnement peut être moins précis. Le message CWOBJ4543 s'affiche dans le journal pour n'importe quel objet d'entrée de cache qui ne peut pas être dimensionné avec précision parce qu'il est trop complexe. Vous pouvez obtenir une mesure plus précise en évitant une complexité de mappe inutile.

Procédure

- Activez l'agent de dimensionnement.

  Si vous utilisez une machine virtuelle Java 5 ou supérieure, utilisez l'agent de dimensionnement. Avec cet agent, WebSphere eXtreme Scale peut obtenir des informations supplémentaires de la machine JVM afin d'améliorer ses estimations. Il est possible de charger l'agent en ajoutant l'argument suivant à la ligne de commande de la machine virtuelle Java :

  ```
  -javaagent:rép_biblio_WXS/wxssizeagent.jar
  ```

  Dans le cas d'une topologie imbriquée, ajoutez l'argument à la ligne de commande du processus WebSphere Application Server.

  Dans le cas d'une topologie répartie, ajoutez l'argument à la ligne de commande des processus eXtreme Scale (conteneurs) et du processus WebSphere Application Server.

  Lorsque le chargement s'est effectué correctement, le message suivant est écrit dans le fichier `SystemOut.log` :

  ```
  CWOBJ4541I: La fonction de définition optimisée de la taille de la
  mémoire BackingMap est activée.
  ```

- Dans la mesure du possible, utilisez de préférence des types de données Java plutôt que des types de données personnalisés.

  WebSphere eXtreme Scale sait estimer avec précision le coût en mémoire des types suivants :

  – java.lang.String et les tableaux où String est la classe de composant (String[])
  – tous les types d'encapsuleurs primitifs (Byte, Short, Character, Boolean, Long, Double, Float, Integer) et les tableaux où ces encapsuleurs primitifs sont le type de composant (par exemple, Integer[], Character[])
  – java.math.BigDecimal et java.math.BigInteger ainsi que les tableaux où ces deux classes sont le type de composant (BigInteger[] et BigDecimal[])

- les types temporels (java.util.Date, java.sql.Date, java.util.Time, java.sql.Timestamp)
- java.util.Calendar et java.util.GregorianCalendar

- Evitez d'utiliser l'internement d'objet lorsque cela est possible.

  Lorsqu'un objet est inséré dans une mappe, WebSphere eXtreme Scale suppose qu'il détient la seule référence à l'objet et tous les objets auxquels l'objet fait référence directement. Si vous insérez 1 000 objets personnalisés dans une mappe et que chacun de ces objets comporte une référence à la même instance de chaîne, WebSphere eXtreme Scale dimensionne l'instance de chaîne 1 000 fois en surestimant la taille réelle de la mappe dans le segment de mémoire. Toutefois, WebSphere eXtreme Scale compense correctement dans les scénarios d'internement courants suivants :

  - références aux énums Java 5
  - références aux classes conformes au pattern Typesafe Enum. Les classes conformes à ce modèle n'ont que des constructeurs privés définis et au moins une zone finale statique privée ayant son propre type et si elles implémentent Serializable, la classe implémente la méthode readResolve().
  - confinement d'un encapsuleur primitif Java 5. Par exemple, avec Integer.valueOf(1) à la place du nouveau Integer(1)

  Si vous devez utiliser l'internement, utilisez l'une des techniques précédentes pour obtenir des estimations plus précises.

- Utilisez les types personnalisés à bon escient.

  Lorsque vous devez utiliser des types personnalisés, privilégiez les types primitifs de données pour les champs, plutôt que les types d'objets.

  Préférez également à vos propres implémentations personnalisées les types d'objets répertoriés dans l'entrée 2.

  Lorsque vous utilisez des types personnalisés, veillez à ce que l'arborescence des objets n'ait pas plus d'un niveau. Lorsqu'on insère un objet personnalisé dans une mappe, WebSphere eXtreme Scale ne calculera que le coût de l'objet inséré, ce qui inclut tous les champs primitifs et tous les objets directement référencés par l'objet inséré. WebSphere eXtreme Scale n'ira pas plus loin pour suivre les références dans l'arborescence. Si vous insérez un objet dans la mappe et que WebSphere eXtreme Scale détecte des références qui n'ont pas été suivies lors du dimensionnement, un message codé CWOBJ4543 contenant le nom de la classe qui n'a pu être dimenssionnée est généré. Lorsque cette erreur se produit, traitez les statistiques de taille de la mappe comme des données de tendance au lieu de vous baser sur des statistiques de taille comme total exact.

- Utilisez le mode de copie CopyMode.COPY_TO_BYTES, si possible.

  Utilisez le mode de copie CopyMode.COPY_TO_BYTES pour supprimer toute incertitude liée aux dimensionnement des objets valeur insérés dans la mappe, même lorsqu'une arborescence d'objets comporte trop de niveaux pour être dimensionnée normalement (ce qui génère le message CWOBJ4543).

Concepts associés:

«Définition de la taille du cache en fonction de son utilisation»
WebSphere eXtreme Scale peut estimer avec précision l'utilisation du segment de mémoire Java d'une mappe de sauvegarde en octets. Cette capacité vous permet de dimensionner correctement les segments de mémoire de vos machines virtuelles Java et de définir de manière appropriée les règles d'expulsion. Le comportement de cette évaluation varie selon le degré de complexité des objets placés dans la mappe de sauvegarde et selon la configuration de la mappe. Actuellement, cette fonctionnalité n'est prise en charge que pour les grilles de données réparties. Les instances de la grille de données locales ne prennent pas en charge le dimensionnement des octets utilisé.

Définition de la taille du cache en fonction de son utilisation

WebSphere eXtreme Scale peut estimer avec précision l'utilisation du segment de mémoire Java d'une mappe de sauvegarde en octets. Cette capacité vous permet de dimensionner correctement les segments de mémoire de vos machines virtuelles Java et de définir de manière appropriée les règles d'expulsion. Le comportement de cette évaluation varie selon le degré de complexité des objets placés dans la mappe de sauvegarde et selon la configuration de la mappe. Actuellement, cette fonctionnalité n'est prise en charge que pour les grilles de données réparties. Les instances de la grille de données locales ne prennent pas en charge le dimensionnement des octets utilisé.

Remarques sur la consommation du segment de mémoire

eXtreme Scale stocke toutes ses données à l'intérieur de l'espace du segment de mémoire des processus JVM qui constituent la grille de données. Pour une mappe donnée, le segment mémoire qu'utilise cette mappe peut se décomposer dans les éléments suivants :

- la taille de tous les objets key actuellement dans la mappe
- la taille de tous les objets value actuellement dans la mappe
- la taille de tous les objets EvictorData qui sont en cours d'utilisation par le plug-in Evictor dans la mappe
- la taille de la structure de données sous-jacente

Le nombre d'octets utilisés qui est retourné par les statistiques de dimensionnement correspondra donc à la somme de ces quatre composants. Ces valeurs sont calculées en fonction de chaque entrées dans les opérations de mappe d'insertion, de mise à jour et de suppression, ce qui implique que eXtreme Scale contient toujours la valeur actualisée du nombre d'octets qu'une mappe de sauvegarde donnée consomme.

Lorsque les données sont partitionnées, chaque partition contient une partie de la mappe de sauvegarde. Comme les statistiques de dimensionnement sont calculées au niveau le plus bas du code eXtreme Scale, chaque partition d'une mappe se sauvegarde suit sa propre taille. Vous pouvez utiliser les API Statistics d'eXtreme Scale pour suivre la taille cumulée de l'ensemble de la mappe ainsi, d'ailleurs, que les tailles individuelles des partitions qui composent cette mappe.

En règle générale, utilisez les données de taille comme mesure de tendance des données dans le temps et non pas comme une mesure exacte de l'espace de mémoire segmenté qui est utilisé par la mappe. Ainsi, si la taille signalée d'une mappe double de 5 à 10 Mo, considérez que l'utilisation de la mémoire par la mappe a doublé. La mesure réelle 10 Mo peut être inexacte pour diverses raisons.

Si vous tenez compte de ces raisons et que vous appliquez les bonnes pratiques que nous préconisons, le dimensionnement sera presque aussi exact que le post-traitement d'un cliché de segment mémoire Java.

Le principal problème avec l'exactitude vient de ce que le modèle mémoire de Java n'est pas suffisamment restrictif et qu'il tolère une certaine marge de précision dans les mesures de la mémoire. Le problème fondamental réside dans le fait qu'un objet peut figurer dans le segment de mémoire parce qu'il existe des références. Par exemple, si une même instance d'objet 5 Ko est insérée dans trois mappes distinctes, ces trois instances empêchent la récupération de la place occupée par l'objet. Dans ce cas de figure, les mesures suivantes sont tout aussi valables :

- La taille de chaque mappe augmente de 5 Ko.
- La taille de la première mappe dans laquelle l'objet est placé augmente de 5 Ko.
- La taille des deux autres mappes n'augmente pas. La taille de chaque mappe augmente d'une fraction de la taille de l'objet.

Cette ambiguïté est la raison pour laquelle ces mesures doivent être considérées comme des données de tendance, sauf si vous avez supprimé l'ambiguïté en effectuant des choix de conception, en appliquant les pratiques recommandées et en comprenant les choix d'implémentation pour fournir des statistiques plus précises.

eXtreme Scale présuppose que chaque mappe est la seule à détenir la référence durable aux objets Key et Value qu'elle contient. Si le même objet de 5 Ko est placé dans trois mappes, la taille de chaque mappe augmente de 5 Ko. L'augmentation n'est généralement pas un problème, car la fonctionnalité n'est prise en charge que pour les grilles de données réparties. Si vous insérez le même objet dans trois mappes différentes sur un client distant, chaque mappe reçoit sa propre copie de l'objet. Par ailleurs, les paramètres par défaut du mode transactionnel COPY MODE garantissent que chaque mappe ait sa propre copie d'un objet donné.

Internement d'objet

L'internement des objets peut poser des problèmes pour estimer l'utilisation du segment de mémoire. Lorsque vous implémentez l'internement des objets, le code de l'application fait en sorte que toutes les références à une valeur d'objet pointe vers la même instance d'objet et donc le même emplacement en mémoire. Voici un exemple avec la classe ci-dessous :

```
public class ShippingOrder implements Serializeable,Cloneable{

    public static final STATE_NEW = "new";
    public static final STATE_PROCESSING = "processing";
    public static final STATE_SHIPPED = "shipped";

    private String state;
    private int orderNumber;
private int customerNumber;

public Object clone(){
       ShippingOrder toReturn = new ShippingOrder();
       toReturn.state = this.state;
       toReturn.orderNumber = this.orderNumber;
       toReturn.customerNumber = this.customerNumber;
       return toReturn;
    }

    private void readResolve(){
        if (this.state.equalsIgnoreCase("new")
```

```
            this.state = STATE_NEW;
        else if (this.state.equalsIgnoreCase("processing")
            this.state = STATE_PROCESSING;
        else if (this.state.equalsIgnoreCase("shipped")
            this.state = STATE_SHIPPED:
    }
}
```

L'internement d'objet entraîne une surestimation par les statistiques de dimensionnement, car eXtreme Scale suppose que les objets utilisent des emplacements de mémoire différents. Si un million d'objets ShippingOrder existe, les statistiques de dimensionnement affichent le coût d'un million de chaînes détenant les informations d'état. En réalité, seules trois chaînes existent qui sont des membres de classe statique. Le coût de la mémoire pour les membres d'une classe statique ne devrait jamais être ajouté à une mappe eXtreme Scale. Toutefois, cette situation ne peut pas être détecté lors de l'exécution. Il existe des dizaines de façons d'utiliser l'internement d'une manière similaire pour les objets et 'est la raison pour laquelle la détection est difficile. eXtreme Scale ne peut pas fournir une protection contre toutes les implémentations possibles. Toutefois, eXtreme Scale offre une protection contre les types usuels d'internement des objets. Pour optimiser l'internement d'objet, implémentez l'internement uniquement sur les objets personnalisés qui se divisent dans les deux catégories suivantes pour améliorer la précision des statistiques d'utilisation de la mémoire :

- eXtreme Scale s'ajuste automatiquement pour les énumérations et le pattern Typesage Enum Java 5, comme indiqué dans Java 2 Platform Standard Edition 5.0 Overview: Enums.
- eXtreme Scale tient compte automatiquement de l'internement automatique des types d'encapsuleurs de type primitif, tels que Integer. L'internement automatique des types d'encapsuleurs de type primitif a été introduit dans Java 5 avec l'utilisation des méthodes valueOf statiques.

Statistiques de consommation de mémoire

Pour accéder aux statistiques d'utilisation de la mémoire, utilisez l'une des méthodes suivantes.

API Statistics

Utilisez la méthode La MapStatsModule.getUsedBytes() qui fournit les statistiques pour une seule mappe unique, notamment le nombre d'entrées et le taux de réussite.

Pour des détails, voir Modules des statistiques.

Beans gérés (MBeans)

Utilisez la statistique de bean géré MapUsedBytes. Vous pouvez utiliser plusieurs types de beans gérés JMX (Java Management Extensions) différents pour administrer et surveiller les déploiements. Chaque bean géré fait référence à une entité spécifique, telle qu'une mappe, eXtreme Scale, un serveur, un groupe de réplication ou un membre de groupe de réplication.

Pour des détails, voir Administration avec les beans gérés (MBeans).

Modules PMI

Vous pouvez surveiller les performances de vos applications avec les modules PMI. Particulièrement, utilisez le module PMI de mappe pour les conteneurs imbriqués dans WebSphere Application Server.

Pour des détails, voir Modules PMI.

Console WebSphere eXtreme Scale

Avec la console, vous pouvez visualiser les statistiques d'utilisation de la mémoire. Voir Surveillance à l'aide de la console Web.

Toutes ces méthodes accèdent à la base aux mêmes mesures de l'utilisation de la mémoire par une instance BaseMap donnée. L'environnement d'exécution WebSphere eXtreme Scale s'efforce de calculer le nombre d'octets de segment de mémoire consommé par les objets key et value stockés dans la mappe et la charge de la mappe elle-même. Vous pouvez déterminer la quantité de segment de mémoire qu'utilise chaque mappe dans l'ensemble de la grille de données répartie.

Dans la plupart des cas, la valeur signalée par WebSphere eXtreme Scale pour une mappe donnée est très proche de la valeur signalée par l'analyse des clichés de segments. WebSphere eXtreme Scale calcule avec précision sa propre charge, mais vous ne pouvez pas tenir compte de chaque objet pouvant être placé dans une mappe. Les meilleures pratiques décrites dans «Optimisation de l'agent de dimensionnement du cache pour des estimations précises de l'utilisation de la mémoire», à la page 429 peuvent améliorer la précision des mesures de taille en octets fournies par WebSphere eXtreme Scale.

Tâches associées:

«Optimisation de l'agent de dimensionnement du cache pour des estimations précises de l'utilisation de la mémoire», à la page 429
WebSphere eXtreme Scale prend en charge le dimensionnement de la consommation de la mémoire par les instances de BackingMap dans les grilles de données réparties. Le dimensionnement de la consommation de mémoire de la taille n'est pas pris en charge pour les instances de grilles de données locales. La valeur signalée par WebSphere eXtreme Scale pour une mappe donnée est très proche de la valeur signalée par l'analyse des clichés de segments. Si l'objet de mappe est complexe, le dimensionnement peut être moins précis. Le message CWOBJ4543 s'affiche dans le journal pour n'importe quel objet d'entrée de cache qui ne peut pas être dimensionné avec précision parce qu'il est trop complexe. Vous pouvez obtenir une mesure plus précise en évitant une complexité de mappe inutile.

Optimisation et performances pour le développement d'applications

Pour améliorer les performances de votre grille de données interne ou l'espace de traitement de base de données, vous pouvez examiner plusieurs éléments, par exemple à l'aide des meilleures pratiques des fonctionnalités du produit, telles que le verrouillage, la sérialisation et les performances des requêtes.

Optimisation du mode de copie

WebSphere eXtreme Scale effectue une copie de la valeur en fonction des paramètres CopyMode disponibles. C'est à vous de déterminer lequel de ces paramétrages fonctionne le mieux pour les besoins de votre déploiement.

Vous pouvez utiliser la méthode setCopyMode(CopyMode, valueInterfaceClass) de l'API BackingMap pour définir le mode de copie dans l'une des zones statiques finales suivantes définies 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 que dans la transaction de grille de données qu'a obtenue 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

Lorsque vous utilisez une mappe qui est associée à une entité d'API EntityManager, la mappe retourne toujours directement sans copie les objets Tuple d'entité sans procéder à une copie, sauf si vous utilisez le mode de copie COPY_TO_BYTES. 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 à la 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. En utilisant le mode COPY_TO_BYTES, vous pouvez réduire la mémoire que peut occuper un grand graphique d'objets. Pour des informations complémentaires, voir «Amélioration des performances avec des mappes de tableaux d'octets», à la page 440.

COPY_TO_BYTES_RAW

7.1.1+ Avec COPY_TO_BYTES_RAW, vous pouvez directement accéder au formulaire sérialisé de vos données. Ce mode de copie offre un moyen efficace pour interagir avec les octets sérialisés, ce qui vous permet d'ignorer le processus de désérialisation pour accéder aux objets en mémoire.

Dans le fichier descripteur XML d'ObjectGrid, vous pouvez définir le mode de copie COPY_TO_BYTES et définir à l'aide d'un programme le mode de copie COPY_TO_BYTES_RAW dans les instances où vous souhaitez accéder aux données brutes sérialisée. Définissez le mode de copie COPY_TO_BYTES _RAW dans le fichier descripteur XML d'ObjectGrid uniquement lorsque votre application utilise les données brutes dans un processus d'application principal.

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.

Référence associée:

Fichier XML du descripteur d'ObjectGrid
Pour configurer WebSphere eXtreme Scale, utilisez un fichier XML de descripteur d'ObjectGrid et l'API ObjectGrid.

Amélioration des performances avec des mappes de tableaux d'octets

Vous pouvez stocker les valeurs de vos mappes dans un tableau d'octets plutôt que dans un formulaire POJO pour réduire la consommation de mémoire d'un graphique 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 «Optimisation des performances de sérialisation», à la page 447.

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" />
```

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.

Référence associée:

Fichier XML du descripteur d'ObjectGrid
Pour configurer WebSphere eXtreme Scale, utilisez un fichier XML de descripteur d'ObjectGrid et l'API ObjectGrid.

Optimisation des opérations de copie à l'aide de 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.

 L'interface ObjectTransformer a été remplacée par les plug-ins DataSerializer que vous pouvez utiliser pour stocker efficacement les données arbitraires dans WebSphere eXtreme Scale pour que les API de produit existantes puissent interagir efficacement avec vos données.

Présentation

Des copies des valeurs sont toujours créées, 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 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.

Optimisation des expulseurs

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 améliorent les performances des expulseurs LFU (least frequently used) et LRU (least recently used).

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.

Tâches associées:

Activation des expulseurs à l'aide d'un programme
Les expulseurs sont associés à des instances BackingMap.

Activation des expulseurs avec la 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 :

Référence associée:

Fichier XML du descripteur d'ObjectGrid
Pour configurer WebSphere eXtreme Scale, utilisez un fichier XML de descripteur d'ObjectGrid et l'API ObjectGrid.

Optimisation des 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, voir «Gestionnaire de verrous», à la page 234:

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.

Voir pour plus d'informations. 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.

Optimisation des 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.

 L'interface ObjectTransformer a été remplacée par les plug-ins DataSerializer que vous pouvez utiliser pour stocker efficacement les données arbitraires dans WebSphere eXtreme Scale pour que les API de produit existantes puissent interagir efficacement avec vos données.

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.

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. 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, des objets doivent être modifiés pour utiliser la sérialisation personnalisée, telle que pour l'implémentation de l'interface java.io.Externalizable ou en implémentant 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

Lorsque vous utilisez l'API ObjectMap ou DataGrid, les objets key et value sont sérialisés lorsque le client interagit avec la grille de données et que 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.

Un plug-in MapSerializerPlugin peut être associé à un plug-in BackingMap lorsque vous utilisez le mode de copie COPY_TO_BYTES ou COPY_TO_BYTES_RAW. Cette association permet de stocker les données dans le format sérialisé en mémoire et non pas dans le format d'objet Java natif. Le stockage des données sérialisées permet d'économiser la mémoire et améliore la réplication et les performances sur le client et le serveur. Vous pouvez utiliser un plug-in DataSerializer pour développer des flux de sérialisation haute performance qui peuvent être compressés, chiffrés, améliorés et interrogés.

Optimisation de la sérialisation

Le plug-in DataSerializer expose les métadonnées qui indiquent à WebSphere eXtreme Scale les attributs qu'il peut et ne peut pas utiliser directement pendant la sérialisation, le chemin d'accès aux données qui seront sérialisées, ainsi que le type des données stockées en mémoire. Vous pouvez optimiser la sérialisation d'objet et les performances de l'inflation afin que vous puissiez interagir efficacement avec le tableau d'octets.

Présentation

 L'interface ObjectTransformer a été remplacée par les plug-ins DataSerializer que vous pouvez utiliser pour stocker efficacement les données arbitraires dans WebSphere eXtreme Scale pour que les API de produit existantes puissent interagir efficacement avec vos données.

Des copies des valeurs sont toujours créées, 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. Toutefois, pour améliorer les performances dans la plupart des cas, vous pouvez utiliser les plug-ins DataSerializer pour sérialiser les objets. Vous devez utiliser soit le mode de copie COPY_TO_BYTES ou COPY_TO_BYTES_RAW pour exploiter les plug-ins DataSerializer. Pour plus d'informations, voir Sérialisation à l'aide des plug-ins DataSerializer.

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.

7.1.1+ Les plug-ins DataSerializer remplacent les plug-ins ObjectTransformer qui sont obsolètes. Pour sérialiser vos données de la façon la plus efficace, utilisez les plug-ins DataSerializer pour améliorer les performances dans la plupart des cas. Par exemple, si vous souhaitez utiliser des fonctions, telles que la requête et l'indexation, vous pouvez immédiatement bénéficier de l'amélioration des performances que procurent les plug-ins DataSerializer sans modifier la configuration ou effectuer des modifications à l'aide d'un programme dans le code de l'application.

Optimisation des performances des 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.

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» 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 interagit avec les ObjectMaps 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 conséquences de légères modifications dans une chaîne de recherche sur l'exécution d'une requête par eXtreme Scale.

Le plan de requête peut être affiché de deux manières :

- Méthodes d'API getPlan pour EntityManager Query ou ObjectQuery
- la trace de diagnostics d'ObjectGrid

Méthode getPlan

La méthode getPlan sur les interfaces ObjectQuery et Requête les interfaces renvoie une chaîne qui décrit le plan de requête. Cette chaîne peut être affichée de manière standard ou dans un journal.

Remarque : Dans un environnement réparti, la méthode getPlan ne s'exécute pas sur le serveur et elle ne reflète 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 «Collecte de trace», à la page 502 pour savoir comment activer la trace et repérer les fichiers de trace.

Exemples de plans de requête

Le plan de requête utilise le mot (word) for pour indiquer que la requête opère une itération dans une collection ObjectMap ou, dans une collection dérivée par exemple : 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 utilisée. 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, et la boucle interne ne s'exécute donc 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. Cependant, la requête suivante est plus performant car elle commence par restreindre le résultat à un seul objet department en utilisant l'index unique qui est défini sur le numéro de zone de 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 exécute la boucle interne 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 plus d'informations sur la configuration de HashIndex, voir «Plug-ins d'indexation des données», à la page 326.

Pour plus d'informations sur l'indexation, voir «Indexation», à la page 97.

Pour une manière plus efficace de rechercher les objets mis en cache, voir «Utilisation d'un index composite», à la page 335.

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 451. 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'
```

Optimisation des performances de l'interface EntityManager

L'interface EntityManager sépare les applications de l'état conservé dans son magasin de données de grille de données 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 essentielle lorsque l'application est complète. Vous pouvez modifier le code qui utilise des interfaces EntityManager pour utiliser des mappes et des bloc de données. En règle générale, cette modification peut être nécessaire pour 10 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'implémentation de ObjectTransformer. Les applications sont optimisées automatiquement.

Modification du code 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 bloc de données value a trois attributs : **firstName**, **middlename**, et **surname**, indexed at 0, 1, and 2. Le bloc de données key possède un seul attribut ; numéro d'ID est indexé à 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.

Tâches associées:

«Tutoriel : Stockage des informations de commande dans des entités», à la page 7
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.

Référence associée:

«Agent d'instrumentation des performances d'entité»
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.

«Définition d'un schéma d'entité», à la page 170
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.

«Programmes d'écoute d'entité et méthodes de rappel», à la page 186
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.

«Exemple de programme d'écoute d'entité», à la page 190
Vous pouvez écrire des programmes d'écoute d'entité en fonction de vos exigences. Vous trouverez ci-après plusieurs exemples de script.

«Interface EntityTransaction», à la page 201
Vous pouvez utiliser l'interface EntityTransaction pour démarquer les transactions.

Agent d'instrumentation des performances d'entité

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.

Concepts associés:

«Optimisation des performances de l'interface EntityManager», à la page 462
L'interface EntityManager sépare les applications de l'état conservé dans son magasin de données de grille de données de serveurs.

«Mise en cache d'objets et de leurs relations (API EntityManager)», à la page 166
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 dans WebSphere Application Server, entre autres, appliquent cette approche. Les API fondées sur les mappes connaissent toutefois des limitations. L'API EntityManager simplifie l'interaction avec la grille de données en facilitant la déclaration et l'interaction avec un graphique complexe d'objets associés.

«Gestionnaire d'entités dans un environnement distribué», à la page 179
Vous pouvez utiliser l'API EntityManager avec un ObjectGrid local ou dans un environnement distribué eXtreme Scale. 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.

«Interactions avec EntityManager», à la page 183
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.

«Stratégie FetchPlan de l'EntityManager», à la page 192
Un plan d'extraction (FetchPlan) est la stratégie utilisée par EntityManager pour extraire des objets associés si l'application doit accéder aux relations.

«Files d'attente des requêtes d'entité», à la page 196
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.

Tâches associées:

«Tutoriel : Stockage des informations de commande dans des entités», à la page 7
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.

Chapitre 7. Sécurité

WebSphere eXtreme Scale permet de sécuriser l'accès aux données et l'intégration de fournisseurs de sécurité externes. Les éléments de sécurité comprennent l'authentification, l'autorisation, la sécurité du transport, la sécurité de la grille de données, la sécurité locale et la sécurité JMX (MBean).

Configuration des profils de sécurité pour l'utilitaire `xscmd`

En créant un profil de sécurité, vous pouvez utiliser les paramètres de sécurité enregistrés pour utiliser l'utilitaire **xscmd** avec des environnements sécurisés.

Avant de commencer

Pour plus d'informations sur la configuration de l'utilitaire **xscmd**, voir Administration avec l'utilitaire **xscmd**.

Pourquoi et quand exécuter cette tâche

Vous pouvez utiliser le paramètre **-ssp** *profile_name* ou **--saveSecProfile** *profile_name* avec le reste de la commande **xscmd** pour enregistrer un profil de sécurité. Le profil peut contenir des paramètres pour les noms d'utilisateur et les mots des générateurs de données d'identification, des fichiers de clés, des fichiers de clés certifiées et des types de transport.

Le groupe de commandes **ProfileManagement** dans l'utilitaire **xscmd** contient des commandes de gestion de vos profils de sécurité.

Procédure

- Enregistrez un profil de sécurité.

  Pour enregistrer un profil de sécurité, utilisez le paramètre **-ssp** *profile_name* ou **--saveSecProfile** *profile_name* avec le reste de la commande. L'ajout de ce paramètre à la commande enregistre les paramètres suivants :

  ```
  -al,--alias <alias>
  -arc,--authRetryCount <integer>
  -ca,--credAuth <support>
  -cgc,--credGenClass <className>
  -cgp,--credGenProps <property>
  -cxpv,--contextProvider <provider>
  -ks,--keyStore <filePath>
  -ksp,--keyStorePassword <password>
  -kst,--keyStoreType <type>
  -prot,--protocol <protocol>
  -pwd,--password <password>
  -ts,--trustStore <filePath>
  -tsp,--trustStorePassword <password>
  -tst,--trustStoreType <type>
  -tt,--transportType <type>
  -user,--username <username>
  ```

  Les profils de sécurité sont enregistrés dans le répertoire *user_home*`\.scmd\profiles\security\<`*profile_name*`>.properties`.
- Utilisez un profil de sécurité enregistré.

Pour utiliser un profil de sécurité enregistré, ajoutez le paramètre **-sp** *profile_name* ou **--securityProfile** *profile_name* à la commande que vous exécutez. Exemple de commande : `xscmd -c listHosts -cep myhost.mycompany.com -sp myprofile`

- Listez les commandes dans le groupe de commandes **ProfileManagement**.
  Exécutez la commande **xscmd -lc ProfileManagement**.
- Listez les profils de sécurité existants.
  Exécutez la commande **xscmd -c listProfiles -v**.
- Affichez les paramètres enregistrés dans un profil de sécurité.
  Exécutez la commande **xscmd -c showProfile -pn** *profile_name*.
- Supprimez un profil de sécurité existant.
  Exécutez la commande **xscmd -c RemoveProfile -pn** *profile_name*.

Référence associée:

Migration de l'outil **xsadmin** vers l'outil **xscmd**
Dans les versions précédentes, l'outil **xsadmin** était un exemple d'utilitaire de ligne de commande pour surveiller l'état de l'environnement. L'outil **inattendue** a été introduit comme outil officiel de ligne de commande d'administration et de surveillance. Si vous utilisiez l'outil **xsadmin**, migrez les commandes vers le nouvel outil **xscmd**.

Programmation de la sécurité

Utilisez des interfaces de programmation pour gérer divers aspects de la sécurité dans un environnement WebSphere 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.

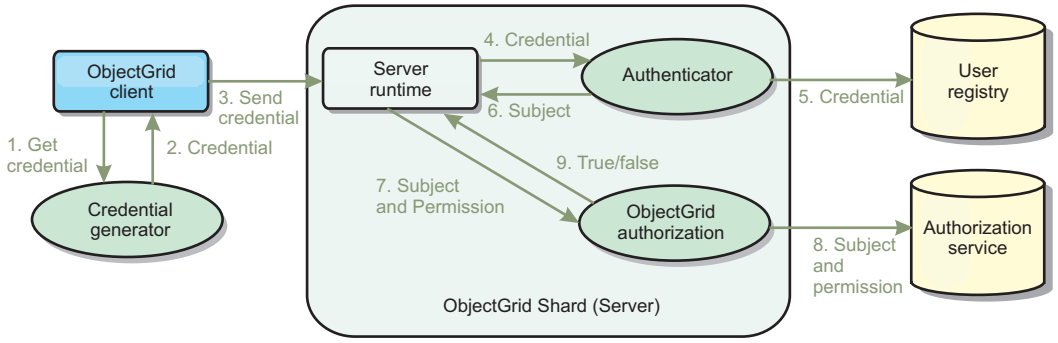


Figure 31. Flux d'authentification et d'autorisation du client

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 de données : la rubrique Authentification de grille de données explique comment vous pouvez utiliser SecureTokenManager pour transporter de manière sécurisée les valeurs serveur secrètes.

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 de client

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. 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;
    }

    /**
     * Obtient le nom d'utilisateur de 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 de 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 le code de hachage de cet objet
     */
    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 accepté.
 * <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 stocké.
 *
 * @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 );

// Obtenir la mappe d'employé
ObjectMap om = session.getMap("employee");

// Démarrer 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 );

// Obtenir la mappe d'employé
om = session.getMap("employee");

// Démarrer 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).

Information associée:
API Credential

Programmation d'autorisations client

WebSphere eXtreme Scale prend en charge l'autorisation JAAS (Java Authentication and Authorization Service) déjà prête à être utilisée et prend également en charge l'autorisation personnalisée en utilisant l'interface ObjectGridAuthorization.

Le plug-in ObjectGridAuthorization permet d'autoriser les accès ObjectGrid, ObjectMap et JavaMap aux principaux représentés par un objet Subject d'une manière personnalisée. Une implémentation type de ce plug-in est d'extraire les principaux d'un objet Subject et de 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 11. Liste des méthodes et de la MapPermission requise

Permission	**ObjectMap/JavaMap**
read	Boolean containsKey(Object)
	Boolean equals(Object)
	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 12. 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 13. 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 de la version SDK 5 et des versions suivantes.

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 est constitué d'un groupe de principaux et chaque principal représente une identité pour l'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`, accordent toutes les autorisations à ces mappes 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 est utilisé comme registre d'utilisateurs, le nom de machine est MACH1, l'ID utilisateur est user1 et principal_name est `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 Boolean checkPermission(Subject subject, Permission permission) de l'interface ObjectGridAuthorization est appelée par l'environnement d'exécution

WebSphere eXtreme Scale pour vérifier que l'objet subject envoyé a l'autorisation d'envoi. L'implémentation de l'interface ObjectGridAuthorization retourne true si l'objet a l'autorisation et false s'il ne l'a pas.

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;
}
```

Information associée:

Module 4 : Utilisation de l'autorisation JAAS (Java Authentication and Authorization Service) dans WebSphere Application Server
Maintenant que vous avez configuré l'authentification pour les clients, vous pouvez configurer les autorisations de manière plus précise pour accorder aux utilisateurs des autorisations différentes. Par exemple, un "opérateur" peut être autorisé uniquement à afficher les données, alors qu'un "gestionnaire" peut exécuter toutes les opérations.

Authentification d'une grille de données

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.

Programmation de la 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.

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 8. Résolution des incidents

Outre les journaux et de trace, les messages et les notes sur l'édition mentionnés dans la présente section, vous pouvez utiliser des outils de surveillance pour identifier et résoudre les incidents tels que l'emplacement des données dans l'environnement, la disponibilité des serveurs dans la grille de données, 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'utilitaire **xscmd** pour afficher des informations textuelles sur votre environnement.

Activation de la consignation

Vous pouvez utiliser des journaux pour surveiller et traiter les problèmes liés à votre environnement.

Pourquoi et quand exécuter cette tâche

Les journaux sont enregistrés dans des emplacements différents et les formats dépendant de votre configuration.

Procédure

- **Activez des journaux dans un environnement autonome.**

  Avec les serveurs de catalogue autonomes, les journaux se trouvent dans le répertoire dans lequel vous exécutez la commande **startOgServer**. Pour les serveurs de conteneur, vous pouvez utiliser l'emplacement par défaut ou définir un emplacement de journal personnalisé :

  – **Emplacement de journal par défaut :** les journaux se trouvent dans le répertoire où la commande serveur a été exécutée. Si vous démarrez les serveurs dans le répertoire *rép_base_wxs*/bin, les journaux et les fichiers de trace se trouvent dans les sous-répertoires logs/<*nom_serveur*> du répertoire bin.

  – **Emplacement de journal personnalisé :** pour définir un autre emplacement pour les journaux des serveurs de conteneur, créez un fichier de propriétés, tel que server.properties contenant :

  ```
  workingDirectory=<directory>
  traceSpec=
  systemStreamToFileEnabled=true
  ```

  La propriété **workingDirectory** est le répertoire racine des journaux et du fichier de trace facultatif. WebSphere eXtreme Scale crée un répertoire avec le nom du serveur de conteneur avec un fichier SystemOut.log, un fichier SystemErr.log et un fichier de trace. Pour utiliser un fichier de propriétés au démarrage des conteneurs, utilisez l'option **-serverProps** et spécifiez l'emplacement du fichier de propriétés du serveur.

- **Activez les journaux dans WebSphere Application Server.**

  Voir WebSphere Application Server: Activation et désactivation de la consignation pour plus d'informations.

- **Extrayez les fichiers 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
```

Que faire ensuite

Affichage des fichiers journaux dans leur emplacement spécifié. Les messages courants à rechercher dans le fichier `SystemOut.log` sont les messages de confirmation du démarrage, comme dans l'exemple suivant :

```
CWOBJ1001I: ObjectGrid Server catalogServer01 is ready to process requests.
```

Pour plus d'informations sur un message spécifique dans les fichiers journaux, voir Messages.

Référence associée:

«Options de trace», à la page 504
Vous pouvez activer la trace pour fournir des informations sur votre environnement au service d'assistance IBM.

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.

Collecte de trace

Vous pouvez utiliser une trace pour surveiller et traiter les problèmes liés à votre environnement. Vous devez fournir une trace pour un serveur lorsque vous contactez le support IBM.

Pourquoi et quand exécuter cette tâche

La collecte d'une trace peut vous aider à surveiller et corriger les problèmes dans votre déploiement de WebSphere eXtreme Scale. La manière dont vous collectez la trace dépend de votre configuration. Voir «Options de trace», à la page 504 pour la liste des spécifications de trace que vous pouvez collecter.

Procédure

- **Collectez la trace dans un environnement WebSphere Application Server.**

  Si votre catalogue et les serveurs de conteneur se trouvent dans un environnement WebSphere Application Server, voir WebSphere Application Server : Utilisation avec la fonction de trace pour plus d'informations.

- **Collectez la trace avec le catalogue autonome ou la commande de démarrage de serveur.**

  Vous pouvez définir la trace sur un service de catalogue ou serveur de conteneur en utilisant les paramètres **`-traceSpec`** et **`-traceFile`** avec la commande **`startOgServer`**. Par exemple :

  ```
  startOgServer.sh catalogServer -traceSpec ObjectGridPlacement=all=enabled -traceFile  /home/user1/logs/trace.log
  ```

  Le paramètre **`-traceFile`** est facultatif. Si vous ne définissez pas un emplacement **`-traceFile`**, le fichier de trace est placé dans le même endroit que les fichiers journaux du système.Pour plus d'informations sur ces paramètres, voir la documentation sur le script startOgServer dans *Guide d'administration*.

- **Collectez la trace sur le catalogue autonome ou serveur conteneur avec un fichier de propriétés.**

  Pour collecter une trace à partir d'un fichier de propriétés, créez un fichier, tel que `server.properties`, avec le contenu suivant :

  ```
  workingDirectory=<directory>
  traceSpec=<trace_specification>
  systemStreamToFileEnabled=true
  ```

  La propriété **workingDirectory** est le répertoire racine des journaux et du fichier de trace facultatif. Si la valeur **workingDirectory** n'est pas définie, le répertoire de travail par défaut est l'emplacement utilisé pour démarrer les serveurs (par exemple, *rép_base_wxs*`/bin`). Pour utiliser un fichier de propriétés au cours du démarrage du serveur, utilisez le paramètre **-serverProps** avec la commande **startOgServer** et fournissez l'emplacement du fichier de propriétés du serveur.Pour plus d'informations sur le fichier de propriétés du serveur et l'utilisation du fichier, voir les les informations relatives au fichier des propriétés du serveur dans *Guide d'administration*.

- **Collectez la trace sur un client autonome.**

  Vous pouvez démarrer la collecte de trace sur un client autonome en ajoutant des propriétés système au script de démarrage pour l'application client. Dans l'exemple suivant, les paramètres de trace sont spécifiés pour l'application `com.ibm.samples.MyClientProgram` :

  ```
  java -DtraceSettingsFile=MyTraceSettings.properties
  -Djava.util.logging.manager=com.ibm.ws.bootstrap.WsLogManager
  -Djava.util.logging.configureByServer=true com.ibm.samples.MyClientProgram
  ```

  Voir WebSphere Application Server : Activation de la trace sur les applications client et autonomes pour plus d'informations.

- **Collectez la trace avec l'interface ObjectGridManager.**

  Vous pouvez également 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");
  ```

  Pour plus d'informations sur l'interface ObjectGridManager, voir les informations sur l'interaction avec ObjectGrid en utilisant l'interface ObjectGridManager dans *Guide de programmation*.

- **Collectez la trace sur les serveurs de conteneur avec l'utilitaire xscmd.**

  Pour collecter une trace à l'aide de l'utilitaire **xscmd**, utilisez la commande **setTraceSpec**. Utilisez l'utilitaire **xscmd** pour collecter la trace sur un environnement autonome lors de la phase d'exécution et non pas au démarrage. Vous pouvez collecter la trace sur tous les serveurs et services de catalogue ou filtrer les serveurs en fonction du nom ObjectGrid, et d'autres propriétés. Par exemple, pour collecter la trace ObjectGridReplication avec un accès au serveur de service de catalogue, exécutez :

  ```
  xscmd -c setTraceSpec "ObjectGridReplication=all=enabled"
  ```

  Vous pouvez également désactiver la trace en affectant à la spécification de trace la valeur `*=all=disabled`.

Résultats

Les fichiers de trace sont écrits dans l'emplacement défini.

Référence associée:

«Options de trace»
Vous pouvez activer la trace pour fournir des informations sur votre environnement au service d'assistance IBM.

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.

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. Vous pouvez définir 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.

ObjectGridClientInfo
Informations sur le client DB2.

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.

ObjectGridMonitor
Infrastructure de la surveillance de l'historique.

7.1.1+ **ObjectGridNative**
Trace de code natif WebSphere eXtreme Scale, y compris le code natif eXtremeMemory.

7.1.1+ **ObjectGridOSGi**
Les composants d'intégration OSGi WebSphere eXtreme Scale.

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.

7.1.1+ **ObjectGridSerializer**
Infrastructure de plug-in DataSerializer.

ObjectGridStats
Statistiques d'ObjectGrid.

ObjectGridStreamQuery
API de la requête de flux.

7.1.1+ **ObjectGridTransactionManager**
Gestionnaire de transaction WebSphere eXtreme Scale.

ObjectGridWriteBehind
Ecriture différée d'ObjectGrid.

7.1.1+ **ObjectGridXM**
Trace IBM eXtremeMemory générale.

7.1.1+ **ObjectGridXMEviction**
Trace d'expulsion eXtremeMemory.

7.1.1+ **ObjectGridXMTransport**
Trace de transport générale eXtremeMemory.

7.1.1+ **ObjectGridXMTransportInbound**
Trace de transport entrant eXtremeMemory.

7.1.1+ **ObjectGridXMTransportOutbound**
Trace de transport sortant eXtremeMemory.

Projector
Moteur dans l'API EntityManager.

QueryEngine
Moteur de requête des API Object Query et EntityManager Query.

QueryEnginePlan
Trace du plan de requête.

7.1.1+ **TCPChannel**
Canal TCP/IP IBM eXtremeIO.

7.1.1+ **XsByteBuffer**
Trace de mémoire tampon d'octets WebSphere eXtreme Scale.

Tâches associées:

«Activation de la consignation», à la page 501
Vous pouvez utiliser des journaux pour surveiller et traiter les problèmes liés à votre environnement.

«Collecte de trace», à la page 502
Vous pouvez utiliser une trace pour surveiller et traiter les problèmes liés à votre environnement. Vous devez fournir une trace pour un serveur lorsque vous contactez le support IBM.

Démarrage des serveurs autonomes
Lorsque vous exécutez une configuration autonome, l'environnement se compose de serveurs de catalogue, de serveurs de conteneur et de processus client. Les serveurs WebSphere eXtreme Scale peuvent être également intégrés à des applications Java existantes en utilisant l'API Embedded Server. Vous devez manuellement configurer et démarrer ces processus.

Administration avec l'utilitaire **xscmd**
Avec **xscmd**, vous pouvez effectuer des tâches d'administration dans l'environnement, telles qu'établir des liens de réplication multimaître, remplacer un quorum et arrêter des groupes de serveurs avec la commande teardown.

Analyse des journaux et des données de trace

Vous pouvez utiliser les outils d'analyse de journal pour analyser le fonctionnement de votre environnement d'exécution et résoudre les problèmes qui s'y produisent.

Pourquoi et quand exécuter cette tâche

Vous pouvez générer des rapports à partir des fichiers journaux et de trace existants dans l'environnement. Ces rapports graphiques peuvent être utilisés pour les objectifs suivants :

- **Analyse de l'état et des performances de l'environnement d'exécution :**
  - Cohérence de l'environnement de déploiement
  - Fréquence de consignation
  - Exécution de la topologie et topologie configurée
  - Modifications non planifiées de la topologie
  - Etat de quorum
  - Etat de la réplication des partitions
  - Statistiques de mémoire, rendement, utilisation du processeur, etc.

- **Pour traiter les problèmes dans l'environnement :**
  - Vues topologiques à des points spécifiques dans le temps
  - Statistiques de mémoire, rendement, utilisation du processeur au cours des problèmes
  - Niveaux de groupe de correctifs en cours, paramètres d'optimisation
  - Etat de quorum

Présentation de l'analyse du journal

Vous pouvez utiliser l'outil **xsLogAnalyzer** pour vous aider traiter les problèmes dans l'environnement.

Tous les messages de reprise en ligne

Affiche le nombre total de messages de reprise en ligne sous la forme d'un graphique dans le temps. Affiche également une liste des messages de reprise en ligne, y compris les serveurs qui ont été affectés.

Tous les messages critiques eXtreme Scale

Affiche les ID de message et les explications associées et les actions utilisateur qui peuvent vous faire gagner du temps dans la recherche des messages.

Toutes les exceptions

Affiche les cinq premières exceptions, y compris les messages et leur nombre et les serveurs affectés par l'exception.

Résumé de la topologie

Affiche le diagramme de la configuration de la topologie en fonction des fichiers journaux. Vous pouvez utiliser ce récapitulatif pour comparer avec votre configuration réelle, en identifiant les erreurs de configuration.

Cohérence de topologie : tableau de comparaison ORB (Object Request Broker)

Affiche les paramètres ORB de l'environnement. Vous pouvez utiliser ce tableau pour déterminer si les paramètres de l'environnement sont cohérents.

Vue Tableau chronologique d'événements

Affiche un diagramme chronologique des différentes actions qui ont eu lieu sur la grille de données, y compris les événements de cycle de vie, les exceptions, les messages critiques et les événements de diagnostic de premier niveau (FFDC).

Exécution de l'analyse du journal

Vous pouvez exécuter l'outil **xsLogAnalyzer** sur un ensemble de fichiers journaux et de trace à partir de n'importe quel ordinateur.

Avant de commencer

- Activez les journaux et la trace. Pour plus d'informations, reportez-vous aux rubriques «Activation de la consignation», à la page 501 et «Collecte de trace», à la page 502.

- Collectez vos fichiers journaux. Les fichiers journaux peuvent se trouver dans des emplacements différents selon la façon dont vous les avez configurés. Si vous utilisez les paramètres de journal par défaut, vous pouvez obtenir les fichiers journaux dans les emplacements suivants :
  - Dans une installation autonome : *racine_install_wxs*/bin/logs/<*server_name*>
  - Dans une installation intégrée à WebSphere Application Server : *racine_was*/logs/<*server_name*>
- Collectez vos fichiers de trace. Ils peuvent se trouver dans des emplacements différents selon la façon dont vous les avez configurés. Si vous utilisez les paramètres de trace par défaut, vous pouvez obtenir les fichiers de trace dans les emplacements suivants :
  - Dans une installation autonome : si aucune valeur de trace n'est définie, les fichiers de trace sont écrits dans le même emplacement que les fichiers journaux système.
  - Dans une installation intégrée à WebSphere Application Server : *racine_was*/profiles/*server_name*/logs.

  Copiez les fichiers journaux et de trace vers l'ordinateur à partir duquel vous avez l'intention d'utiliser l'outil d'analyse de journal.
- Si vous voulez créez des scanners personnalisés, créez un fichier de propriétés de spécifications de scanner et un fichier de configuration avant d'exécuter l'outil. Pour plus d'informations, voir «Création de scanners personnalisés pour l'analyse de journal», à la page 509.

Procédure

1. Exécutez l'outil **xsLogAnalyzer**.

   Le script se trouve dans les emplacements suivants :
   - Dans une installation autonome : *racine_install_wxs*/ObjectGrid/bin
   - Dans une installation intégrée à WebSphere Application Server : *racine_was*/bin

   Conseil : Si les fichiers journaux sont volumineux, utilisez les paramètres **-startTime**, **-endTime**, et **-maxRecords** lorsque vous exécutez le rapport pour limiter le nombre d'entrées de journal analysées. L'utilisation de ces paramètres lorsque vous exécutez le rapport améliore la clarté et l'exécution du rapport. Vous pouvez exécuter plusieurs rapports sur un même groupe de fichiers journaux.

   ```
   xsLogAnalyzer.sh|bat -logsRoot c:\myxslogs -outDir c:\myxslogs\out
   -startTime 11.09.27_15.10.56.089 -endTime 11.09.27_16.10.56.089 -maxRecords 100
   ```

   -logsRoot
   Spécifie le chemin absolu du répertoire des journaux à évaluer (requis).

   -outDir
   Spécifie un répertoire pour y placer la sortie du rapport. Si vous ne définissez pas une valeur, le rapport est écrit dans l'emplacement racine de l'outil **xsLogAnalyzer**.

   -startTime
   Spécifie l'heure de début de l'évaluation dans les journaux. La date a le format *year.month.day.hour.minute.second.millisecond*

   -endTime
   Spécifie l'heure de fin de l'évaluation dans les journaux. La date a le format *year.month.day.hour.minute.second.millisecond*

-trace Spécifie une chaîne de trace, telle que `ObjectGrid*=all=enabled`.

-maxRecords
Spécifie le nombre maximal d'enregistrements pour générer le rapport. La valeur par défaut est `100`. Si vous définissez la valeur 50, les 50 premiers enregistrements sont générés pour la période définie.

2. Ouvrez les fichiers générés. Si vous n'avez pas défini de répertoire de sortie, les rapports sont générés dans le dossier `report_date_time`. Pour ouvrir la page principale des rapports, ouvrez le fichier `index.html`.
3. Utilisez les rapports pour analyser les données des journaux. Suivez les conseils ci-dessous pour optimiser les performances du rapport affiché :
   - Pour optimiser les performances des requêtes sur les données des journaux, utilisez des informations aussi spécifiques que possibles. Par exemple, la recherche de `server` dure plus longtemps et retourne plus de résultats que `server_host_name`.
   - Certaines vues ont un nombre limité de points de données affichés simultanément. Vous pouvez ajuster le segment de temps affiché en changeant les données en cours, telles que les heures de début et de fin, dans la vue.

Que faire ensuite

Pour plus d'informations sur le traitement de l'outil **xsLogAnalyzer** et les rapports générés, voir «Traitement des problèmes d'analyse de journal», à la page 510.

Création de scanners personnalisés pour l'analyse de journal

Vous pouvez créer des scanners personnalisés pour l'analyse de journal. Après avoir configuré le scanner, les résultats sont générés dans les rapports lorsque vous exécutez l'outil **xsLogAnalyzer**. Le scanner personnalisé recherche les enregistrements d'événement dans les journaux en fonction des expressions régulières que vous avez définies.

Procédure

1. Créez un fichier de propriétés de spécification de scanner qui définit l'expression générale à exécuter pour le scanner personnalisé.
   a. Créez et enregistrez un fichier de propriétés. Le fichier doit se trouver dans le répertoire *loganalyzer_root*`/config/custom`. Vous pouvez attribuer le nom de choix. Le fichier est utilisé par le nouveau scanner ; il est donc utile de nommer le scanner dans le fichier des propriétés. Par exemple, *my_new_server_scanner_spec*`.properties`.
   b. Incluez les propriétés suivantes dans le fichier *my_new_server_scanner_spec*`.properties` :

      ```
      include.regular_expression = REGULAR_EXPRESSION_TO_SCAN
      ```

      La variable *REGULAR_EXPRESSION_TO_SCAN* est une expression régulière en fonction de laquelle vous filtrez les fichiers journaux.

      Exemple : pour analyser les instances des lignes qui contiennent les chaînes `"xception"` et `"rror"`, quel que soit l'ordre, affectez la valeur suivante à la propriété **include.regular_expression** :

      ```
      include.regular_expression = (xception.+rror)|(rror.+xception)
      ```

      Cette expression régulière permet d'enregistrer les événements si la chaîne `"rror"` se trouve avant ou après la chaîne `"xception"`.

Exemple :Pour analyser chaque ligne des journaux pour rechercher les lignes qui contiennent la chaîne "xception" ou "rror" quel que soit l'ordre, affectez la valeur suivante à la propriété **include.regular_expression** :

```
include.regular_expression = (xception)|(rror)
```

Cette expression régulière permet d'enregistrer les événements si la chaîne "rror" se trouve avant ou après la chaîne "xception".

2. Créez un fichier de configuration que l'outil **xsLogAnalyer** utilise pour créer le scanner.
   a. Créez et enregistrez un fichier de configuration. Le fichier doit se trouver dans le répertoire *loganalyzer_root*/config/custom. Vous pouvez nommer le fichier *scanner_name*Scanner.config, où *scanner_name* est le nom unique du nouveau scanner. Par exemple, vous pouvez nommer le fichier serverScanner.config
   b. Incluez les propriétés suivantes dans le fichier *scanner_name*Scanner.config :

```
scannerSpecificationFiles = LOCATION_OF_SCANNER_SPECIFICATION_FILE
```

La variable *LOCATION_OF_SCANNER_SPECIFICATION_FILE* est le chemin et l'emplacement du fichier de spécification que vous avez créé au cours de l'étape précédente. Par exemple : *loganalyzer_root*/config/custom/my_new_scanner_spec.properties. Vous pouvez aussi définir plusieurs fichiers de spécification de scanner en utilisant une liste d'éléments séparés par un point-virgule :

```
scannerSpecificationFiles = LOCATION_OF_SCANNER_SPECIFICATION_FILE1;LOCATION_OF_SCANNER_SPECIFICATION_FILE2
```

3. Exécutez l'outil **xsLogAnalyzer**. Pour plus d'informations, voir «Exécution de l'analyse du journal», à la page 507.

Résultats

Après avoir exécuté l'outil **xsLogAnalyzer**, le rapport contient de nouveaux onglets pour les scanners personnalisés que vous avez configurés. Chaque onglet contient les vues suivantes :

Graphiques
Graphique qui illustre les événements enregistrés. Les événements sont affichés dans leurs ordre de découverte.

Tableaux
Représentation tabulaire des événements enregistrés.

Etats récapitulatifs

Traitement des problèmes d'analyse de journal

Utilisez les informations de dépannage pour identifier et éliminer les problèmes avec l'outil **xsLogAnalyzer** et ses rapports générés.

Procédure

- **Problème :** manque de mémoire lors de l'utilisation de l'outil **xsLogAnalyzer** pour générer des rapports. Exemple d'erreur possible : java.lang.OutOfMemoryError: GC overhead limit exceeded.

  Solution : l'outil **xsLogAnalyzer** s'exécute dans une machine JVM (Java virtual machine). Vous pouvez configurer la machine JVM pour augmenter la taille de segment avant d'exécuter l'outil **xsLogAnalyzer** en définissant certains paramètres lorsque vous exécutez l'outil. L'augmentation de la taille du segment

permet de stocker plus d'enregistrements dans la mémoire JVM. Commencez avec 2 048 M en supposant que le système d'exploitation dispose d'une mémoire principale suffisante. Dans la même instance de ligne de commande dans laquelle vous voulez exécuter l'outil **xsLogAnalyzer**, définissez la taille de segment de mémoire JVM maximale :

```
java -Xmx HEAP_SIZE m
```

La valeur *HEAP_SIZE* peut être un entier et représente le nombre de mégaoctets alloués au segment de mémoire JVM. Par exemple, vous pouvez exécuter `java -Xmx2048m`. Si vous continuez de recevoir des messages indiquant un manque de mémoire ou que vous ne disposez pas des ressources pour allouer 2 048 Mo ou plus, limitez le nombre d'événements stockés dans le segment de mémoire. Vous pouvez limiter le nombre d'événements dans le segment de mémoire en envoyant le paramètre **-maxRecords** dans la commande **xsLogAnalyzer**.

- **Problème :** lorsque vous ouvrez un rapport généré depuis l'outil **xsLogAnalyzer**, le navigateur se bloque et ne charge pas la page.

  Cause : les fichiers HTML générés sont trop volumineux et le navigateur ne peut pas les charger. Ces fichiers sont volumineux, car la portée des fichiers journaux que vous analysez est trop grande.

  Solution : utilisez les paramètres **-startTime**, **-endTime**, et **-maxRecords** lorsque vous exécutez l'outil **xsLogAnalyzer** pour limiter le nombre d'entrées de journal analysées. L'utilisation de ces paramètres lorsque vous exécutez le rapport améliore la clarté et l'exécution du rapport. Vous pouvez exécuter plusieurs rapports sur un même groupe de fichiers journaux.

Traitement des problèmes de connectivité

Il existe plusieurs problèmes courants propres aux clients et à la connectivité client que vous pouvez résoudre comme indiqué dans les sections suivantes.

Procédure

- **Problème** Si vous utilisez l'API EntityManager ou des mappes de tableaux d'octets avec le mode de copie COPY_TO_BYTES, des méthodes d'accès aux données client génèrent des exceptions de sérialisation ou une exception NullPointerException.
  - L'erreur suivante se produit lorsque vous utilisez le mode de copie COPY_TO_BYTES :

```
java.lang.NullPointerException
        at com.ibm.ws.objectgrid.map.BaseMap$BaseMapObjectTransformer2.inflateObject(BaseMap.java:5278)
        at com.ibm.ws.objectgrid.map.BaseMap$BaseMapObjectTransformer.inflateValue(BaseMap.java:5155)
```

  - L'erreur suivante se produit lorsque vous utilisez l'API EntityManager :

```
java.lang.NullPointerException
  at com.ibm.ws.objectgrid.em.GraphTraversalHelper.fluffFetchMD(GraphTraversalHelper.java:323)
  at com.ibm.ws.objectgrid.em.GraphTraversalHelper.fluffFetchMD(GraphTraversalHelper.java:343)
  at com.ibm.ws.objectgrid.em.GraphTraversalHelper.getObjectGraph(GraphTraversalHelper.java:102)
  at com.ibm.ws.objectgrid.ServerCoreEventProcessor.getFromMap(ServerCoreEventProcessor.java:709)
  at com.ibm.ws.objectgrid.ServerCoreEventProcessor.processGetRequest(ServerCoreEventProcessor.java:323)
```

  Cause : L'API EntityManager et le mode de copie COPY_TO_BYTES utilisent un référentiel de métadonnées intégré à la grille de données. Lorsque les clients se connectent, la grille de données stocke les identificateurs de référentiel dans le client et met en cache les identificateurs pour la durée de la connexion client. Si vous redémarrez la grille de données, vous perdez toutes les métadonnées et les identificateurs régénérés ne correspondent pas aux identificateurs en cache sur le client.

Solution : Si vous utilisez l'API EntityManager ou le mode de copie COPY_TO_BYTES, déconnectez et reconnectez tous les clients si l'ObjectGrid est arrêté et redémarré. La déconnexion et reconnexion des clients actualisent la mémoire cache des identificateurs des métadonnées. Vous pouvez déconnecter les clients à l'aide de la méthode ObjectGridManager.disconnect ou la méthode ObjectGrid.destroy.

- **Problème :** le client se bloque au cours d'un appel de la 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.

  Cause : le client attend que les métadonnées d'entité sur le serveur ObjectGrid deviennent disponibles.

  Solution : cette erreur peut se produire lorsqu'un serveur de conteneur a été démarré, mais pas le placement. Procédez comme suit :

  – Examinez la stratégie de déploiement de l'ObjectGrid et vérifiez que le nombre de conteneurs actifs est supérieur ou égal aux deux attributs numInitialContainers et minSyncReplicas dans le fichier descripteur de la stratégie de déploiement.

  – Examinez la valeur de la propriété **placementDeferralInterval** dans le fichier des propriétés du serveur de conteneur pour déterminer le délai qui doit s'écouler avant l'exécution du placement.

  – Si vous avez utilisé la commande **xscmd -c suspendBalancing** pour arrêter l'équilibrage des fragments d'une grille de données et d'un groupe de mappes, utilisez la commande **xscmd -c resumeBalancing** pour redémarrer l'équilibrage.

Concepts associés:

«Création d'instances avec l'interface ObjectGrid ObjectGridManager», à la page 136
Chacune de ces méthodes crée une instance locale d'ObjectGrid.

Traitement des problèmes d'intégration du cache

Utilisez ces informations pour traiter les problèmes de la configuration de l'intégration du cache, y compris ceux associés aux configurations de session HTTP et de cache dynamique.

Procédure

- 7.1.1+ **Problème :** les ID de session HTTP ne sont pas réutilisés.

  Cause : vous pouvez réutiliser les ID de session. Si vous créez une grille de données pour la persistance des sessions dans la version 7.1.1 ou une version ultérieure, la réutilisation des ID de session est automatiquement activée. Toutefois, si vous avez créé des configurations dans des versions antérieures, ce paramètre est peut être déjà défini avec une valeur incorrecte.

  Solution : vérifiez les paramètres suivants pour déterminer si vous avez activé la réutilisation des ID de session HTTP :

  – La propriété `reuseSessionId` dans le fichier `splicer.properties` doit avoir la valeur `true`.

  – La propriété personnalisée `HttpSessionIdReuse` doit avoir la valeur `true`. Cette propriété personnalisée peut être définie dans l'un des chemins suivants dans la console d'administration WebSphere Application Server :

    - **Serveurs** > *server_name* > **Getion de session** > **Propriétés personnalisées**

- **Clusters dynamiques** > *dynamic_cluster_name* > **Modèle de serveur** > **Gestin de session** > **Propriétés personnalisés**
- **Serveurs** > **Types de serveur** > **Serveurs d'applications WebSphere** > *server_name*, puis sous Infrastructure du serveur, cliquez sur **Java et gestion des processus** > **Définition de processus** > **Java virtual machine** > **Propriétés personnalisées**
- **Serveurs** > **Types de serveur** > **Serveurs d'applications WebSphere** > *server_name* > **Paramètres de conteneur Web** > **Conteneur Web**

Si vous mettez à jour les valeurs des propriétés personnalisées, reconfigurez la gestion des sessions eXtreme Scale afin que le fichier `splicer.properties` détecte la modification

- **Problème :** lorsque vous utilisez un grille de données pour stocker les sessions HTTP et que la charge des transactions est élevée, le message CWOBJ0006W figure dans le fichier `SystemOut.log`.

  ```
  CWOBJ0006W: An exception occurred:
  com.ibm.websphere.objectgrid.ObjectGridRuntimeException:
  java.util.ConcurrentModificationException
  ```

  Ce message apparaît lorsque le paramètre **`replicationInterval`** dans le fichier `splicer.properties` a une valeur supérieure à zéro et que l'application Web modifie l'objet List défini comme attribut dans la session HTTPSession.

  Solution : clonez l'attribut qui contient l'objet List modifié et placez l'attribut cloné dans l'objet session.

Référence associée:

Fichiers XML de configuration du gestionnaire de sessions HTTP
Lorsque vous démarrez un serveur de conteneur qui stocke les données de session HTTP, vous pouvez utiliser les fichiers XML par défaut ou spécifier des fichiers XML personnalisés. Ces fichiers créent des noms ObjectGrid spécifiques, un nombre de répliques, etc.

Paramètres d'initialisation du contexte de servlet
La liste qui suit de paramètres d'initialisation du contexte de servlet peut être spécifiée dans le fichier splicer.properties en fonction de la méthode de raccord choisie.

Fichier `splicer.properties`
Le fichier `splicer.properties` contient toutes les options de configuration pour configurer un gestionnaire de sessions basé sur un filtre de servlet.

Traitement des problèmes du plug-in de mémoire cache JPA

Utilisez ces informations pour traiter les problèmes de configuration du plug-in de mémoire cache JPA. Ces problèmes peuvent se produire dans les deux configurations Hibernate et OpenJPA.

Procédure

- **Problème :** l'exception suivante s'affiche : `CacheException: Failed to get ObjectGrid server`.

  Avec la valeur d'attribut `EMBEDDED` ou `EMBEDDED_PARTITION` **`ObjectGridType`**, la mémoire cache eXtreme Scale tente d'obtenir une instance de serveur de l'environnement d'exécution. Dans un environnement Java Platform, Standard Edition, un serveur eXtreme Scale avec un service de catalogue intégré est démarré. Le service de catalogue intégré essaie d'écouter sur le port 2809. Si ce port est utilisé par un autre processus, l'erreur se produit.

Solution : si des noeuds finaux de service de catalogue externes sont spécifiés, par exemple, avec le fichier `objectGridServer.properties`, cette erreur se produit si le nom d'hôte ou le port ne sont spécifiés correctement. Corrigez le conflit de port.

- **Problème :** l'exception suivante s'affiche : `CacheException: Failed to get REMOTE ObjectGrid for configured REMOTE ObjectGrid. objectGridName = [ObjectGridName], PU name = [persistenceUnitName]`

  Cette erreur se produit, car le cache ne peut pas obtenir l'instance ObjectGrid à partir des noeuds finaux de service de catalogue fournis.

  Solution : ce problème se produit généralement lorsque le nom ou le port hôte est incorrect.

- **Problème :** l'exception suivante s'affiche : `CacheException: Cannot have two PUs [persistenceUnitName_1, persistenceUnitName_2] configured with same ObjectGridName [ObjectGridName] of EMBEDDED ObjectGridType`

  Cette exception se produit si un grand nombre d'unités de persistance sont configurées et que les mémoires caches eXtreme Scale de ces unités sont configurées avec le même nom ObjectGrid et la même valeur d'attribut `EMBEDDED` **`ObjectGridType`**. Ces configurations d'unités de persistance peuvent être dans les mêmes fichiers `persistence.xml` ou dans des fichiers différents.

  Solution : vous devez vérifier que le nom d'ObjectGrid est unique pour chaque unité de persistance lorsque la valeur de l'attribut **`ObjectGridType`** est `EMBEDDED`.

- **Problème :** l'exception suivante s'affiche : `CacheException: REMOTE ObjectGrid [ObjectGridName] does not include required BackingMaps [mapName_1, mapName_2,...]`

  Avec un type ObjectGrid REMOTE, si l'ObjectGrid obtenu côté client ne dispose pas des mappes de sauvegarde complètes d'entités pour prendre en charge le cache de l'unité de persistance, cette exception se produit. Supposons par exemple que cinq classes d'entités sont répertoriées dans la configuration des unités de persistance mais que l'ObjectGrid obtenu ne dispose que de deux mappes de sauvegarde. Même si l'ObjectGrid obtenu peut avoir 10 BackingMaps, si l'une des cinq BackingMaps d'entité requises est introuvable dans le 10 mappes de sauvegarde, cette exception se produit toujours.

  Solution : vérifiez que votre configuration de mappes de sauvegarde prend en charge la mémoire cache de l'unité de persistance.

Traitement des problèmes d'administration

Utilisez les informations suivantes pour traiter les problèmes d'administration, notamment le démarrage et l'arrêt des serveurs, en utilisant l'utilitaire **`xscmd`**, etc.

Procédure

- **Problème :** les scripts d'administration manquent dans le répertoire *`profile_root`*`/bin` d'une installation WebSphere Application Server.

  Cause : lorsque vous mettez à jour l'installation, les nouveaux fichiers scripts ne sont pas installés automatiquement dans les profils.

  Solution : si vous voulez exécuter un script depuis le répertoire *`profile_root`*`/bin`, annulez l'extension et étendez de nouveau le profils avec la dernière version. Pour plus d'informations, voir Annulation de l'extension d'un profil en utilisant l'invite de commande et Création et augmentation de profils pour WebSphere eXtreme Scale.

- **Problème :** lorsque vous exécutez une commande **`xscmd`**, le message suivant s'affiche :

```
java.lang.IllegalStateException: Placement service MBean not available.
[]
        at
com.ibm.websphere.samples.objectgrid.admin.OGAdmin.main(OGAdmin.java:1449)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:60)
        at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:37)
        at java.lang.reflect.Method.invoke(Method.java:611)
        at com.ibm.ws.bootstrap.WSLauncher.main(WSLauncher.java:267)
Ending at: 2011-11-10 18:13:00.000000484
```

Cause : problème de connexion avec le serveur de catalogue.

Solution : vérifiez que les serveurs de catalogue sont actifs et disponibles via le réseau. Ce message peut aussi être généré lorsque vous disposez d'un domaine de services de catalogue défini et que moins de deux serveurs de catalogue sont actifs. L'environnement n'est pas disponible tant que deux serveurs de catalogue ne sont pas démarrés.

Concepts associés:

Meilleure pratique : Mise en cluster du service de catalogue avec les domaines de services de catalogue
Lorsque vous utilisez le service de catalogue, un minimum de deux serveurs de catalogue sont requis pour éviter un point de défaillance unique. Selon le nombre de noeuds dans votre environnement, vous pouvez créer des configurations différentes pour qu'au moins deux serveurs de catalogue soient toujours en cours d'exécution.

Administration

Traitement des problèmes de plusieurs configurations de centre de données

Utilisez ces informations pour traiter les problèmes de plusieurs configurations, y compris de la liaison entre les domaine de services de catalogue.

Procédure

Problème : données manquantes dans un ou plusieurs domaine de services de catalogue. Par exemple, vous pouvez exécuter la commande `xscmd -c establishLink`. Lorsque vous examinez les données de chaque domaine de services de catalogue lié, les données sont différentes par rapport à la commande `xscmd -c showMapSizes`.
Solution : vous pouvez traiter ce problème avec la commande `xscmd -c showLinkedPrimaries`. Cette commande consigne chaque fragment primaire, y compris les fragments primaires externes qui sont liés.
Dans le scénario décrit, vous pouvez déterminer à partir de la commande `xscmd -c showLinkedPrimaries` que les fragments primaires du premier domaine de services de catalogue sont liés aux fragments primaires du second domaine de services de catalogue et que ce dernier n'a pas de liaison au premier domaine de services de catalogue. Vous pouvez réexécuter la commande `xscmd -c establishLink` depuis le second domaine de services de catalogue vers le premier domaine de services de catalogue.

Traitement des problèmes des chargeurs

Utilisez ces informations pour traiter les problèmes liés aux chargeurs de base de données.

Procédure

- **Problème :** Lorsque vous utilisez un chargeur OpenJPA avec DB2 dans WebSphere Application Server, une exception de curseur fermé se produit.

  L'exception suivante provient de DB2 et figure dans le fichier journal org.apache.openjpa.persistence.PersistenceException :

  ```
  [jcc][t4][10120][10898][3.57.82] Invalid operation: result set is closed.
  ```

  Solution : par défaut, le serveur d'applications attribue à la propriété personnalisée resultSetHoldability la valeur `2` (CLOSE_CURSORS_AT_COMMIT). Cette propriété amène DB2 à fermer son resultSet/curseur au niveau des limites de la transaction. Pour supprimer l'exception, affectez à la propriété personnalisée, la valeur `1` (HOLD_CURSORS_OVER_COMMIT). Définissez la propriété personnalisée resultSetHoldability dans le chemin suivant dans la cellule WebSphere Application Server : **Ressources** > **Fournisseur JDBC** > **DB2 Universal JDBC Driver Provider** > **DataSources** > *data_source_name* > **Propriétés personnalisées** > **Nouveau**.

- **Problème** DB2 affiche une exception : `The current transaction has been rolled back because of a deadlock or timeout. Reason code "2".. SQLCODE=-911, SQLSTATE=40001, DRIVER=3.50.152`

  Cette exception se produit en raison d'un problème de conflit de verrouillage lorsque vous exécutez OpenJPA avec DB2 dans WebSphere Application Server. Le niveau d'isolement par défaut pour WebSphere Application Server est Lecture reproductible (RR), qui obtient des verrous de longue durée avec DB2.**Solution :**

  Définissez le niveau d'isolement Read Committed pour réduire les conflits de verrouillage. Définissez la propriété personnalisée de source de données webSphereDefaultIsolationLevel pour spécifier le niveau d'isolement 2(TRANSACTION_READ_COMMITTED) dans le chemin suivant dans la cellule WebSphere Application Server : **Ressources** > **Fournisseur JDBC** > *JDBC_provider* > **Sources de données** > *data_source_name* > **Propriétés personnalisées** > **Nouveau**. Pour plus d'informations sur la propriété personnalisée webSphereDefaultIsolationLevel et les niveaux d'isolement de transaction, voir Conditions de définition des niveaux d'isolement de l'accès aux données.

- **Problème :** lorsque vous utilisez la fonction de préchargement de JPALoader ou JPAEntityLoader, le message CWOBJ1511 suivant ne s'affiche pas pour la partition dans un serveur de conteneur : `CWOBJ1511I: GRID_NAME:MAPSET_NAME:PARTITION_ID (primary) is open for business.`

  A la place, une exception TargetNotAvailableException est généré dans le serveur de conteneur qui active la partition définie par la propriété preloadPartition.

  Solution : affectez à l'attribut preloadMode la valeur `true` si vous utilisez un chargeur JPALoader ou JPAEntityLoader pour précharger les données dans la mappe. Si la propriété preloadPartition du chargeur JPALoader et JPAEntityLoader a une valeur comprise entre `0` et `total_number_of_partitions - 1`, il tente de précharger les données à partir de la base de données dorsale dans la mappe. Le fragment de code ci-dessous illustre comment l'attribut preloadMode est défini pour activer le préchargement asynchrone :

  ```
  BackingMap bm = og.defineMap( "map1" );
  bm.setPreloadMode( true );
  ```

  Vous pouvez également définir l'attribut preloadMode à l'aide d'un fichier XML, comme le montre l'exemple suivant :

```
<backingMap name="map1" preloadMode="true" pluginCollectionRef="map1"
 lockStrategy="OPTIMISTIC" />
```

Concepts associés:

«Programmation de l'intégration de JPA», à la page 395
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.

Configuration de l'intégration du cache
WebSphere eXtreme Scale peut s'intégrer aux autres produits de mise en cache. Vous pouvez aussi utiliser le fournisseur de cache dynamique WebSphere eXtreme Scale pour connecter WebSphere eXtreme Scale au composant de cache dynamique WebSphere Application Server. Autre extension de WebSphere Application Server : le gestionnaire de sessions HTTP WebSphere eXtreme Scale, qui permet la mise en cache des sessions HTTP.

Traitement des problèmes d'interblocage

Les sections ci-dessous décrivent certains scénarios d'interblocage courants et des suggestions permettant de les éviter.

Avant de commencer

Implémentez la gestion des exceptions dans l'application. Pour plus d'informations, voir «Implémentation de gestion des exceptions dans les scénarios de verrouillage» , à la page 250.

L'exception suivante s'affiche :

```
com.ibm.websphere.objectgrid.plugins.LockDeadlockException: Message
```

Ce message représente la chaîne transmise en tant que paramètre lorsque l'exception est créée et émise.

Procédure

- **Problème :** exception LockTimeoutException.

  Description : lorsqu'une transaction ou un client demande qu'un verrou doit être octroyé pour une entrée de mappe spécifique, la demande attend généralement que le client en cours lève le verrou avant d'être envoyée. Si la demande de verrou reste inactive pendant une longue période et qu'aucun verrou n'est jamais accordé, exception LockTimeoutException est créée pour empêcher un interblocage, qui est décrit plus en détail dans la section suivante. il est fort probable que vous receviez cette exception lorsque vous utilisez une stratégie de verrouillage pessimiste, car le verrou n'est jamais levé jusqu'à la validation de la transaction.

  Extraire plus de détails :

  L'exception LockTimeoutException contient la méthode getLockRequestQueueDetails qui renvoie une chaîne. Vous pouvez utiliser cette méthode pour visualiser la description détaillée de la situation qui déclenche l'exception. Voici un exemple de code qui intercepte l'exception et affiche un message d'erreur.

```
try {
    ...
}
catch (LockTimeoutException lte) {
    System.out.println(lte.getLockRequestQueueDetails());
}
```

The output result is:

```
lock request queue
->[TX:163C269E-0105-4000-E0D7-5B3B090A571D, state =
    Granted 5348 milli-seconds ago, mode = U]
->[TX:163C2734-0105-4000-E024-5B3B090A571D, state =
    Waiting for 5348 milli-seconds, mode = U]
->[TX:163C328C-0105-4000-E114-5B3B090A571D, state =
    Waiting for 1402 milli-seconds, mode = U]
```

Si vous recevez l'exception dans un bloc catch d'exception ObjectGridException, le code suivant détermine l'exception et affiche les détails de la file d'attente. Il utilise également la méthode de d'utilitaire findRootCause.

```
try {
...
}
catch (ObjectGridException oe) {
    Throwable Root = findRootCause( oe );
    if (Root instanceof LockTimeoutException) {
        LockTimeoutException lte = (LockTimeoutException)Root;
        System.out.println(lte.getLockRequestQueueDetails());
    }
}
```

Solution : une exception LockTimeoutException empêche les blocages possibles dans votre application. Une exception de ce type se produit lorsque l'exception attend un laps de temps défini. Vous pouvez définir le délai d'attente de l'exception à l'aide de la méthode setLockTimeout(int) qui est disponible pour la mappe de sauvegarde BackingMap. Si aucun interblocage réel n'existe dans l'application, ajustez le délai d'attente de verrouillage pour éviter le LockTimeoutException.

Le code suivant montre comment créer un objet ObjectGrid, définir une mappe et affecter la valeur 20 secondes à LockTimeout :

```
ObjectGrid objGrid = new ObjectGrid();
BackingMap bMap = objGrid.defineMap("MapName");
bMap.setLockTimeout(30);
```

Utilisez l'exemple précédent codé en dur pour définir ObjectGrid et les propriétés de la mappe. Si vous créez ObjectGrid à partir d'un fichier XML, définissez l'attribut **LockTimeout** dans l'élément backingMap. Voici un exemple d'élément backingMap qui définit une valeur LockTimeout de mappe à 30 secondes.

```
<backingMap name="MapName" lockStrategy="PESSIMISTIC" lockTimeout="30">
```

- **Problème :** interblocages avec une seule clé.

  Description : les scénarios suivants décrivent comment des interblocages peuvent se produire lors de l'accès à une clé unique 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 14. 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.

Tableau 14. Scénario d'interblocage à clé unique (suite)

	Unité d'exécution 1	**Unité d'exécution 2**	
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 15. 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.
7		session.commit()	Interblocage : le verrou S pour key1 ne peut pas être mis à niveau.

Tableau 16. 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 17. 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 au lieu de get pour obtenir un verrou U à la place d'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 à partir d'un client sont uniquement possibles si le cache de transaction est explicitement invalidé par le même client.
3. Utilisez la stratégie de verrouillage optimiste. L'utilisation de la stratégie de verrouillage optimiste requiert le traitement des exceptions de conflits optimistes.

- **Problème :** interblocage ordonné à plusieurs clés

  Description : ce scénario décrit ce qui se produit si deux transactions tentent de mettre à jour la même entrée et maintiennent des verrous S sur les autres entrées.

Tableau 18. 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.

Tableau 18. Cas d'interblocage à clés multiples (suite) (suite)

	Unité d'exécution 1	Unité d'exécution 2	
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 19. 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.
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 la méthode getForUpdate au lieu de 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.

- **Problème :** erreur d'ordre avec verrou U

  Description : si l'ordre dans lequel les clés sont demandées ne peut pas être garanti, un interblocage peut toujours se produire.

Tableau 20. 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.

Concepts associés:

«Les verrous», à la page 246
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.

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/).

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 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

G

H

I

J

L

M

O

P

R

S

T

V

X