WebSphere® eXtreme Scale Version 7.1
Présentation du produit

Présentation du produit WebSphere eXtreme Scale

IBM

août 2010

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

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

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

Vous pouvez également consulter les serveurs Internet suivants :

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

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

Table des matières

Figures

Tableaux

A propos de la *Présentation du produit*

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

Bibliothèque WebSphere eXtreme Scale

La bibliothèque WebSphere eXtreme Scale contient les documents suivants :

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

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

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

A qui s'adresse ce document

Ce document est destiné à quiconque souhaite en savoir plus sur WebSphere eXtreme Scale.

Structure de ce document

Ce document contient des informations sur les rubriques principales suivantes :

- Le **Chapitre 1** inclut une présentation de WebSphere eXtreme Scale
- Le **Chapitre 2** inclut des informations sur les concepts de mise en cache dans le produit.
- Le **Chapitre 3** inclut des informations sur l'intégration du cache.
- Le **Chapitre 4** inclut des informations sur l'évolutivité.
- Le **Chapitre 5** inclut des informations sur la disponibilité.
- Le **Chapitre 6** inclut des informations sur la sécurité.
- Le **Chapitre 7** inclut des informations sur le traitement des transactions.
- Le **Chapitre 8** inclut des tutoriels pour les concepts de base du produit.
- Le **Chapitre 9** inclut le glossaire du produit.

Obtention des mises à jour de ce document

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

Comment envoyer vos commentaires

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

Chapitre 1. Présentation générale de WebSphere eXtreme Scale

WebSphere eXtreme Scale est une grille de données élastique et évolutive, entièrement présente en mémoire. De manière dynamique, cette grille de données met en cache, partitionne, réplique et gère les données et les logiques applicatives sur une multiplicité de serveurs. WebSphere eXtreme Scale traite avec une extrême efficacité des transactions en quantités massives tout en pouvant monter en puissance de manière linéaire. WebSphere eXtreme Scale permet également de bénéficier de qualités de services comme l'intégrité transactionnelle, la haute disponibilité et la prévisibilité des temps de réponse.

L'élasticité de l'extensibilité est rendue possible par la mise en cache d'objets répartis. Son extensibilité élastique permet à la grille de se surveiller et de se gérer elle-même. Elle peut ajouter (scale-out) des serveurs à la topologie, ou en retirer (scale-in), ce qui augmente, ou diminue, la mémoire, le débit réseau et la capacité de traitement au gré des besoins. Lorsqu'un scale-out est lancé, de la capacité est ajouté à la grille qui ne cesse pas de fonctionner et n'a pas besoin d'être redémarrée. Inversement, un scale-in supprimera de la capacité, tout aussi immédiatement. La grille de données se répare elle-même automatiquement en reprenant son activité après des pannes.

WebSphere eXtreme Scale est utilisable de plusieurs manières différentes. Il peut être utilisé comme un cache extrêmement puissant ou sous la forme d'un espace de traitement de bases de données en mémoire pour gérer l'état des applications. Autre utilisation possible : comme une plateforme permettant d'élaborer de puissantes applications Extreme Transaction Processing (XTP).

Il convient toutefois de noter qu'eXtreme Scale ne peut pas être réduit une simple base de données en mémoire, essentiellement pour la bonne raison qu'une telle base de données est trop simple pour pouvoir gérer quelques-unes des complexités dont se charge eXtreme Scale. Certes, les deux cas de figure ont en commun de présenter un certain nombre d'avantages dus au fait qu'ils fonctionnent tous deux entièrement en mémoire. Mais la première grande différence est que, si l'une des machines d'une base de données en mémoire vient à tomber en panne, la base de données sera incapable de pallier le problème. Une telle éventualité sera particulièrement catastrophique si la machine abrite l'intégralité de l'environnement.

Pour parer à ce type de défaillance, eXtreme Scale fractionne le dataset concerné en partitions, lesquelles équivalent à des schémas d'arborescences soumis à contraintes. Des schémas d'arborescences soumises à contraintes décrivent les relations entre les entités. Lorsqu'on utilise des partitions, les relations entre entités doivent modéliser une structure arborescente des données où la tête de l'arborescence est l'entité racine et la seule entité à être partitionnée. Toutes les entités enfants de cette entité racine sont stockées dans la même partition que leur entité racine. Chaque partition existe sous la forme d'une copie primaire (fragment). Les partitions contiennent également des fragments répliques destinés à sauvegarder les données. Une base de données en mémoire est incapable de fournir ce type de fonctionnalité car elle n'est ni structurée ni dynamique de cette manière, ce qui oblige à effectuer manuellement ce qu'eXtreme Scale sait faire automatiquement. Cela dit, du fait de sa nature de base de données, une base de

données en mémoire autorise les opérations SQL avec, à la clé, des vitesses de traitement incomparables à ceux qu'arrivent à fournir les bases de données qui ne sont pas en mémoire. WebSphere eXtreme Scale possède son propre langage de requêtes, qui n'est pas SQL, et qui est considérablement plus élastique, permettant le partitionnement des données avec une récupération fiable après incident.

Sa fonctionnalité de cache en écriture différée permet à WebSphere eXtreme Scale de servir de cache frontal à une base de données. L'utilisation de ce cache frontal augmente les débits tout en déchargeant et en libérant la base de données. WebSphere eXtreme Scale fournit une évolutivité croissante et décroissante à des coûts de traitement prévisibles.

Le schéma suivant montre que, dans un environnement de cache cohérent et réparti, les clients eXtreme Scale échangent des données avec la grille, laquelle peut être automatiquement synchronisée avec un dorsal de stockage de données. Le cache est dit cohérent car les clients y voient tous les mêmes données. Chaque donnée est exactement stockée sur un seul serveur inscriptible du cache, ce qui évite la prolifération de copies d'enregistrements susceptibles de contenir des versions différentes des mêmes données. Un cache cohérent contient de plus en plus 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 grille croît en taille. Les données peuvent également être répliquées, si l'on souhaite bénéficier de la tolérance aux pannes.

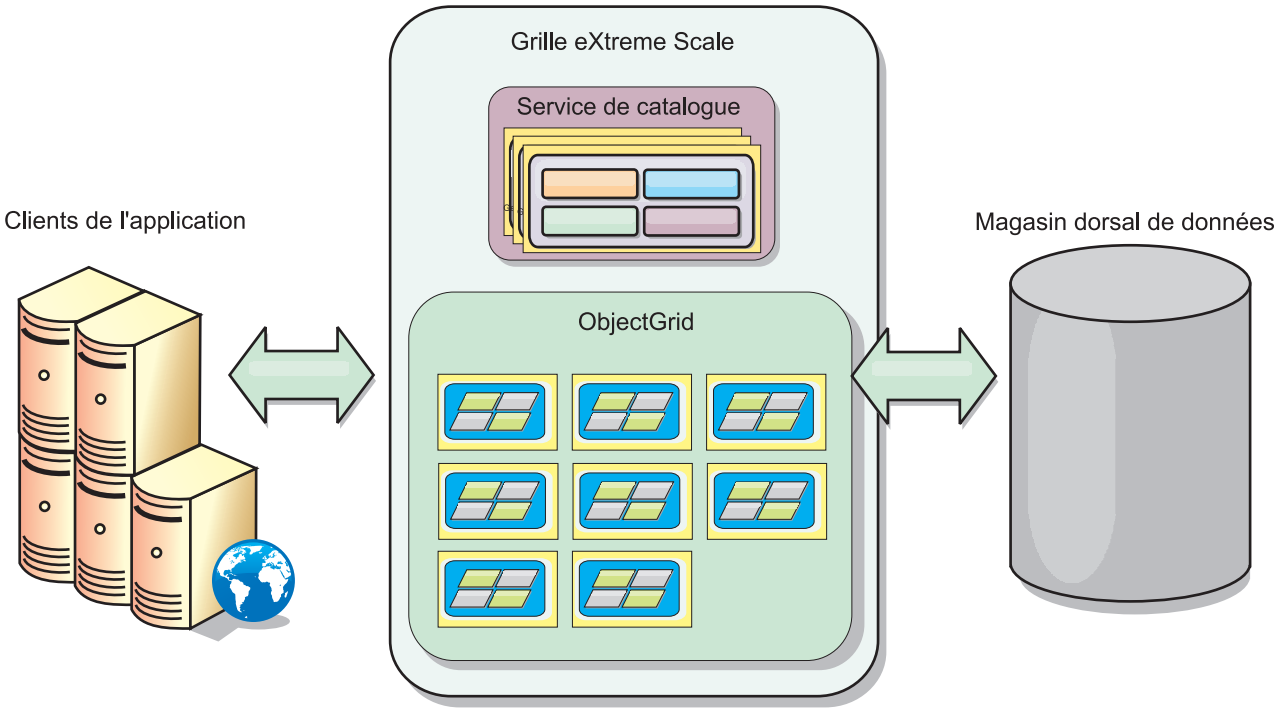


Figure 1. Topologie générale

Ce sont les serveurs de WebSphere eXtreme Scale qui constituent sa grille de données en mémoire. Ces serveurs peuvent s'exécuter au sein de WebSphere Application Server ou sur de simples machines virtuelles Java™ Standard Edition (J2SE), ce qui permet d'en avoir plus qu'une seule par machine physique. On le voit, la grille de données en mémoire peut être extrêmement importante. La grille n'est pas limitée par la mémoire ou l'espace adresse de l'application ou du serveur d'applications et elle n'a aucun impact sur cette mémoire ou cet espace adresse. La

mémoire peut être la somme résultant de la mémoire de centaines, voire de milliers de machines virtuelles Java tournant sur un grand nombre de machines différentes.

En tant qu'espace de traitement de base de données en mémoire, WebSphere eXtreme Scale peut s'appuyer sur un disque, sur une base de données ou sur les deux.

eXtreme Scale a beau fournir plusieurs API Java, la plupart de ses utilisations ne nécessitent aucune programmation de la part des utilisateurs, mais uniquement de la configuration et du déploiement dans l'infrastructure WebSphere.

Le paradigme de base

Le paradigme fondamental d'une grille de données est la paire clé-valeur où la grille stocke des valeurs (des objets Java) en leur associant une clé (un autre objet Java). La clé permet ultérieurement de récupérer la valeur. Dans eXtreme Scale, les mappes sont constituées d'entrées qui ne sont autres que ces paires clé-valeur.

WebSphere eXtreme Scale offre un grand nombre de configurations de grille, allant d'un simple cache local unique à un énorme cache réparti utilisant une multiplicité de machines virtuelles Java ou de serveurs.

Outre les objets Java, il est possible de stocker des objets avec des relations. Il est possible d'utiliser un langage de requêtes semblable à SQL avec des instructions `SELECT... FROM ... WHERE` pour extraire ces objets. Ainsi, un objet Commande sera associé à un objet Client et à plusieurs objets Articles. Dans WebSphere eXtreme Scale, les relations peuvent être de type un à un, un à plusieurs, plusieurs à un ou plusieurs à plusieurs.

WebSphere eXtreme Scale prend également en charge une interface de programmation EntityManager pour le stockage des entités dans le cache. Cette interface de programmation est très semblable aux entités Java Enterprise Edition. Les relations entre entités peuvent être détectées automatiquement à partir d'un fichier XML de descripteur d'entités ou depuis les annotations dans les classes Java. De la sorte, une entité peut être extraite du cache par sa clé primaire à l'aide de la méthode find d'EntityManager. Les entités peuvent être conservées dans la grille de données ou être retirées de cette dernière, le tout au sein d'une limite de transaction.

WebSphere eXtreme Scale fournit des fonctionnalités XTP de traitement extrême des transactions qui permettent aux applications stratégiques les plus exigeantes d'être prises en charge par une infrastructure intelligente. Il est alors possible de s'affranchir des performances limitées de l'informatique classique en générant les niveaux d'échelle globale, d'efficacité des traitements et d'aide à la décision indispensables à qui recherche des résultats plus intelligents et une différenciation compétitive durable.

Avec sa prise en charge de WebSphere Real Time, l'offre Java de temps réel leader du marché, WebSphere eXtreme Scale permet aux applications XTP de réagir avec des temps de réponse plus cohérents et plus prévisibles. Voir dans le *Guide d'administration* les explications sur la prise en charge de Real Time.

Avant de déployer eXtreme Scale dans un environnement de production, plusieurs options sont à prendre en considération : nombre de serveurs à utiliser, quantité d'espace de stockage présente sur chacun de ces serveurs, et choix de la réplication, synchrone ou asynchrone.

Prenons un exemple de cache réparti où la clé est un simple nom alphabétique. Le cache pourra être fractionné en quatre partitions par lettre : partition 1 pour les clés de A à E, partition 2 pour les clés de F à L, etc. Pour assurer la disponibilité, une partition a (est stockée dans) un fragment primaire et un fragment réplique. Les modifications apportées aux données du cache sont effectuées dans le fragment primaire et répliquées vers le fragment secondaire. Dans le cas d'un cache réparti (une grille de données ou un ObjectGrid en termes eXtreme Scale), l'on configure le nombre de serveurs eXtreme Scale destinés à contenir les données de la grille et eXtreme Scale se chargera de répartir dans des fragments les données entre ces instances de serveurs. Pour que la disponibilité soit effective, les fragments répliques sont placés sur des machines distinctes de celles abritant les fragments primaires.

WebSphere eXtreme Scale utilise un service de catalogue pour repérer le fragment primaire de chaque clé. Il gère le transfert des fragments entre les serveurs eXtreme Scale en cas de défaillance et de récupération de ces serveurs ou des machines physiques qui les hébergent. Si, par exemple, le serveur contenant un fragment réplique tombe en panne, eXtreme Scale allouera un nouveau fragment réplique. En cas de défaillance d'un serveur contenant un fragment primaire, le fragment réplique correspondant est promu au statut de fragment primaire et un nouveau fragment réplique est alors construit.

L'interface de programmation d'eXtreme Scale la plus simple est ObjectMap, qui est une simple interface de mappage : une méthode map.put(key,value) pour placer une valeur dans le cache et une méthode map.get(key) pour extraire ultérieurement cette valeur.

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.

Nouvelles fonctions et fonctions dépréciées dans cette version

La version 7.1 de WebSphere eXtreme Scale comporte un grand nombre de nouvelles fonctionnalités, notamment l'intégration au cache dynamique, les mappes de tableaux d'octets, etc.

Nouveautés de WebSphere eXtreme Scale Version 7.1

Tableau 1. Nouvelles fonctions dans WebSphere eXtreme Scale Version 7.1

Fonction	Description
Intégration des informations sur le client DB2	Intègre à DB2 les plug-in du chargeur JPA d'eXtreme Scale, de sorte que si DB2 est utilisé comme base de données dorsale, les informations WebSphere eXtreme Scale (nom d'utilisateur, nom du poste de travail, nom de l'application et informations de comptabilité) seront disponibles dans DB2 Performance Monitor. Cette fonction permet l'activation et la désactivation de la configuration des informations sur le client dans DB2. Cette fonction est désactivée par défaut. Pour plus d'informations, voir «Chargeurs», à la page 43.
Configuration des domaines de services de catalogue	Les domaines de services de catalogue peuvent être configurés à l'aide de la console d'administration de WebSphere Application Server ou des tâches d'administration. Les domaines de services de catalogue définissent un groupe. Pour plus d'informations, voir dans le *Guide d'administration* les explications sur la création de domaines de services de catalogue.

Tableau 1. Nouvelles fonctions dans WebSphere eXtreme Scale Version 7.1 (suite)

Fonction	Description
Réplication multi-maître	Il est possible de relier de manière asynchrone plusieurs centres de données, ce qui leur permet d'accéder en local aux données et de maintenir une disponibilité élevée. Pour plus d'informations, voir «Topologies de réplication de grilles multi-maîtres (PA)», à la page 24.
Expulseur TTL en fonction de la date de dernière modification	L'expulseur TTL prend désormais en compte également l'heure de dernière modification des entrées. Pour plus d'informations, voir dans le in the *Guide de programmation* les explications concernant l'expulseur TimeToLive (TTL)
Statistiques usedBytes pour les grilles en mémoire	Il est possible d'effectuer à l'aide de tous les fournisseurs de statistiques le suivi de la quantité de mémoire utilisée dans une mappe de sauvegarde par les entrées en cache. Pour plus d'informations, voir dans le *Guide d'administration* les explications sur le dimensionnement de l'utilisation de la mémoire cache.
Statistiques dynamiques	Il est désormais possible d'activer et de désactiver les statistiques à la demande. Pour plus d'informations, voir dans le *Guide d'administration* les explications sur la surveillance à l'aide de beans gérés.
Console de surveillance	La console graphique de surveillance permet de visualiser les statistiques actuelles et l'historique des statistiques des serveurs WebSphere eXtreme Scale. Pour plus d'informations, voir dans le *Guide d'administration* les explications sur la console Web.
Amélioration du gestionnaire de sessions HTTP	La configuration du gestionnaire de sessions HTTP a été simplifiée. Vous pouvez désormais configurer ce gestionnaire dans la console d'administration de WebSphere Application Server. Pour plus d'informations, voir dans le *Guide d'administration* les explications sur la configuration du gestionnaire de sessions HTTP.
Prise en charge des clients multihébergés	Il est possible de configurer les clients pour qu'ils utilisent un adaptateur de réseau spécifique. Pour plus d'informations, voir dans le *Guide d'administration* les explications sur le fichier des propriétés des clients.
ISA 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. Pour plus d'informations, voir dans le *Guide d'administration* les explications concernant IBM Support Assistant for WebSphere eXtreme Scale.
REST	Le service de données REST permet aux clients non Java d'accéder aux données eXtreme Scale grâce au protocole OData (Open Data Protocol) assurant une compatibilité complète avec Microsoft® WCF Data Services. Pour plus d'informations, voir Chapitre 8, «Présentation des services de données REST», à la page 181.
Installation uniquement des clients	Les clients WebSphere eXtreme Scale peuvent désormais être installés de manière indépendante, ce qui allège la charge des tâches d'installation pour les applications WebSphere eXtreme Scale. Pour plus d'informations, voir dans le *Guide d'administration* les explications concernant l'installation et le déploiement de WebSphere eXtreme Scale.

Fonctions dépréciées

Tableau 2. Fonctions dépréciées

Dépréciation	Action de migration recommandée
	Dans la console d'administration de WebSphere Application Server, créez un domaine de services de catalogue qui crée la même configuration qu'avec la propriété personnalisée. Pour plus d'informations, voir dans le *Guide d'administration* les explications sur la création de domaines de services de catalogue.
	Utilisez plutôt le bean géré CatalogServiceManagementMBean.
Partitioning Facility (WPF) : l'utilitaire de partitionnement est un ensemble d'API de programmation qui permettent aux applications Java EE de prendre en charge la mise en clusters asymétrique.	Les fonctionnalités de WPF peuvent également être utilisées dans WebSphere eXtreme Scale.
StreamQuery : Requête continue sur les données en cours stockées dans les mappes ObjectGrid.	Aucune
Configuration de grille statique : Topologie statique basée sur les clusters, qui utilise le fichier XML de déploiement des clusters.	Remplacée par la topologie de déploiement dynamique, améliorée, pour la gestion des grilles de données de grande taille.

Tableau 2. Fonctions dépréciées (suite)

Dépréciation	Action de migration recommandée
Propriétés système dépréciées : Les propriétés système permettant de spécifier les fichiers de propriétés des serveurs et des clients sont dépréciées.	Vous pouvez toujours utiliser ces arguments, mais vous devrez remplacer vos propriétés par les nouvelles valeurs. Pour plus d'informations, voir les informations sur les fichiers de propriétés dans le *Guide d'administration*.

Utiliser WebSphere eXtreme Scale

WebSphere eXtreme Scale est une grille de données élastique et évolutive, entièrement présente en mémoire. De manière dynamique, cette grille de données met en cache, partitionne, réplique et gère les données et les logiques applicatives sur une multiplicité de serveurs.

eXtreme Scale n'étant pas une base de données en mémoire, vous devez envisager les conditions spécifiques requises pour sa configuration. La première phase du déploiement d'une grille de données eXtreme Scale consiste à démarrer un groupe central et un service de catalogue qui joueront le rôle de coordinateur de toutes les autres machines virtuelles Java participant à la grille. Ce coordinateur gérera également les informations de configuration. Le démarrage des processus WebSphere eXtreme Scale s'effectue à l'aide de simples scripts de commandes appelés depuis la ligne de commande.

La phase suivante consiste à démarrer les processus serveur WebSphere eXtreme Scale pour la grille permettant de stocker et d'extraire les données. Lorsqu'ils démarrent, les serveurs s'enregistrent automatiquement auprès du groupe central et du service de catalogue, ce qui leur permet de coopérer en fournissant des services de grille. Les capacités et la fiabilité de la grille seront d'autant plus grandes qu'il y aura davantage de serveurs.

Une grille locale est une grille simple, à une seule instance, dans laquelle toutes les données se trouvent dans la grille. Pour utiliser efficacement eXtreme Scale en tant qu'espace de traitement de base de données en mémoire, il est possible de configurer et de déployer une grille répartie. Dans une grille répartie, les données sont réparties entre les divers serveurs eXtreme Scale qui les contiennent de manière à ce que chaque serveur n'en contienne qu'une partie, appelée précisément partition.

Le nombre des partitions dans la grille répartie est un paramètre clé de la configuration de cette grille. Les données de la grille sont partitionnées dans ce nombre de sous-ensembles dont chacun est appelé partition. Le service de catalogue se charge de repérer en fonction de sa clé la partition correspondant à une donnée particulière. Le nombre de partitions affecte directement la capacité et l'extensibilité de la grille . Un serveur peut contenir une ou plusieurs partitions de grille. De ce fait, la taille des partitions est limitée par l'espace mémoire du serveur. Mais, d'un autre côté, augmenter le nombre de partitions augmente la capacité de la grille. La capacité maximum d'une grille correspond au nombre de partitions multiplié par la taille de la mémoire utilisable du serveur, lequel peut être une machine virtuelle Java.

Les données d'une partition sont stockées dans un fragment. Pour permettre la disponibilité, il est possible de configurer la grille avec des fragments répliques, lesquels peuvent être synchrones ou asynchrones. Les modifications apportées aux

données de la grille sont effectuées dans le fragment primaire et répliquées vers les fragments secondaires. De ce fait, la mémoire totale consommée et requise par une grille est le produit de la taille de la grille multipliée par (1 [pour le fragment primaire] + le nombre de fragments répliques).

WebSphere eXtreme Scale répartit les fragments de la grille entre les serveurs contenant celle-ci. Ces serveurs peuvent se trouver aussi bien sur les mêmes machines physiques que sur des machines physiques distinctes. Pour que la disponibilité soit effective, les fragments répliques sont placés sur des machines distinctes de celles abritant les fragments primaires.

WebSphere eXtreme Scale surveille le statut de ses serveurs et déplace les fragments entre ces serveurs en cas de défaillance et de reprise des serveurs ou des machines physiques qui les contiennent. Si, par exemple, le serveur contenant un fragment réplique tombe en panne, eXtreme Scale allouera un nouveau fragment réplique dans lequel il répliquera les données du fragment primaire. En cas de défaillance d'un serveur contenant un fragment primaire, le fragment réplique correspondant est promu au statut de fragment primaire et un nouveau fragment réplique est alors construit. Si l'on ajoute un nouveau serveur supplémentaire à la grille, les fragments seront répartis sur tous les serveurs afin que la charge soit la plus équilibrée possible sur chacun d'entre eux. L'on appelle cela le scale-out ou l'ajout de serveurs. De la même manière pour le "scale-in" ou retrait de serveurs, il est possible d'arrêter l'un des serveurs afin de réduire les ressources consommées par une grille, et, là aussi, les fragments seront équilibrés entre les serveurs restants, tout comme cela se passe en cas de défaillance.

Déploiement d'applications

Avant de commencer à utiliser WebSphere eXtreme Scale dans un environnement de production, les points suivants sont à prendre en considération afin d'optimiser le déploiement.

Planifier le déploiement des applications

Points à prendre en considération :

- le nombre de systèmes et de processeurs : combien faut-il dans l'environnement de machines physiques et de processeurs ?
- le nombre de serveurs : combien de serveurs eXtreme Scale pour héberger les mappes eXtreme Scale ?
- le nombre de partitions : la quantité de données stockées dans les mappes est l'un des facteurs déterminant le nombre de partitions nécessaires
- le nombre de fragments répliques : combien de fragments répliques sont requis pour chacun des fragments primaires du domaine ?
- réplication synchrone ou asynchrone : les données sont-elles si vitales qu'une réplication synchrone soit indispensable ? Ou bien, est-ce que les performances sont une priorité plus importante ? Dans ce cas, la réplication asynchrone s'impose
- la taille des segments : combien de données seront-elles stockées sur chaque serveur ?

Intégration à d'autres produits WebSphere Application Server

WebSphere eXtreme Scale peut être intégré à d'autres produits de serveurs, comme WebSphere Application Server et WebSphere Application Server Community Edition.

Configuration du gestionnaire de sessions HTTP pour qu'il fonctionne avec WebSphere Application Server Community Edition

WebSphere Application Server Community Edition peut partager l'état des sessions, mais d'une manière peu efficace et non évolutive. WebSphere eXtreme Scale fournit une couche de persistance répartie à hautes performances qui peut servir à répliquer l'état mais sans s'intégrer facilement aux autres serveurs d'applications extérieurs à WebSphere Application Server. Vous pouvez intégrer ces deux produits pour offrir une solution de gestion de session évolutive. Pour plus de détails, voir le *Guide d'administration*.

Configuration du gestionnaire de sessions de WebSphere eXtreme Scale pour qu'il fonctionne avec WebSphere Application Server

Le gestionnaire de sessions HTTP a été livré la première fois avec la version 6.1.0.0 de WebSphere Extended Deployment DataGrid. Jusqu'à la version 6.1.0.5, les versions ultérieures n'ont pas changé de méthode d'utilisation puisque celle-ci répondait parfaitement à la spécification Java 2 Enterprise Edition (J2EE) pour l'intégration et l'extraction de sessions. Néanmoins, chaque édition successive comportait son lot d'améliorations en termes de performances et de qualité de service. Pour être sûr de bénéficier de la meilleure qualité de service, l'application du groupe de correctifs WebSphere eXtreme Scale version 6.1.0.5 est fortement recommandée.

Pour plus de détails, voir le *Guide d'administration*.

Noms successifs du produit

Vous devez savoir que WebSphere eXtreme Scale ne s'est pas toujours appelé ainsi.

Noms successifs du produit

Dans les autres sources de référence (documentations, supports marketing ou présentations), eXtreme Scale peut être mentionné sous ses anciens noms :

- ObjectGrid
- WebSphere Extended Deployment Data Grid

Bien que le produit en lui-même soit connu sous le nom de WebSphere eXtreme Scale, le terme d'ObjectGrid figure dans la documentation et dans d'autres sources car c'est le nom de l'artefact qui permet la technologie de grille.

Version d'essai gratuite

Pour vous initier à WebSphere eXtreme Scale, vous pouvez télécharger une version d'essai gratuite. Vous pourrez ainsi développer des applications innovantes à hautes performances en étendant à l'aide de fonctionnalités avancées la notion de mise en cache des données.

Version d'essai à télécharger

Vous pouvez télécharger une version d'essai gratuite de eXtreme Scale à partir de la page de téléchargement de la version d'évaluation d'eXtreme Scale.

Après avoir téléchargé et décompressé la version d'essai, allez au répertoire gettingstarted et prenez connaissance du fichier GETTINGSTARTED_README.txt. Ce tutoriel vous aide à faire vos premiers pas dans eXtreme Scale en vous apprenant à créer une grille sur plusieurs serveurs et à exécuter quelques applications simples vous permettant de stocker et d'extraire des données dans une grille. Avant de déployer eXtreme Scale dans un environnement de production, plusieurs options sont à prendre en considération : nombre de serveurs à utiliser, quantité d'espace de stockage présente sur chacun de ces serveurs, et choix de la réplication, synchrone ou asynchrone.

Guides de programmation et d'administration

Le guide *Présentation du produit* explique les notions fondamentales permettant de comprendre WebSphere eXtreme Scale. Deux autres manuels développent les notions expliqués dans ce guide.

Le *Guide d'administration* explique comment configurer et effectuer les tâches générales d'administration tandis que le *Guide de programmation* rentre dans le détail des API Java permettant d'accéder et de configurer la grille eXtreme Scale.

Chapitre 2. Mise en cache : présentation d'ensemble

WebSphere eXtreme Scale peut fonctionner comme un espace de traitement de la base de données en mémoire offrant une fonction de mise en cache en ligne pour une base de données dorsale ou servant de cache secondaire. La mise en cache en ligne utilise eXtreme Scale comme moyen principal d'interaction avec les données. Lorsqu'eXtreme Scale est utilisé en tant que cache secondaire, le système dorsal est utilisé conjointement avec la grille de données. Cette section décrit divers concepts et scénarios de mise en cache et compare les différentes topologies utilisables pour le déploiement d'une grille de données.

Architecture de la mise en cache : mappes, conteneurs, clients et catalogues

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.

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 le serveur eXtreme Scale et les clients se connectent à distance à ce serveur.

Les caches répartis permettent d'améliorer les performances, la disponibilité et l'évolutivité du système. Les topologies dynamiques utilisées pour les configurer permettent d'équilibrer automatiquement les serveurs. Vous pouvez également ajouter d'autres serveurs sans redémarrer les serveurs eXtreme Scale existants. Il est possible de créer des déploiements simples ou des déploiements plus vastes se chiffrant en téraoctets et comptant plusieurs milliers de serveurs.

Mappes

Une mappe est un conteneur de paires clé-valeur permettant à une application de stocker une valeur indexée par une clé. Les mappes prennent en charge les index pouvant être ajoutés pour indexer les attributs sur la clé ou sur la valeur. Ces index sont automatiquement utilisés par l'environnement d'exécution des requêtes pour déterminer le mode d'exécution des requêtes le plus efficace.

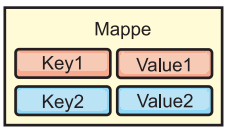


Figure 2. Mappe

Un groupe de mappes est une collection de mappes partageant un même algorithme de partitionnement. Les données contenues dans les mappes sont répliquées en fonction des règles définies par le groupe de mappes. Les groupes de mappes sont uniquement utilisés pour les topologies réparties. Pour les topologies locales, ils ne sont pas nécessaires.

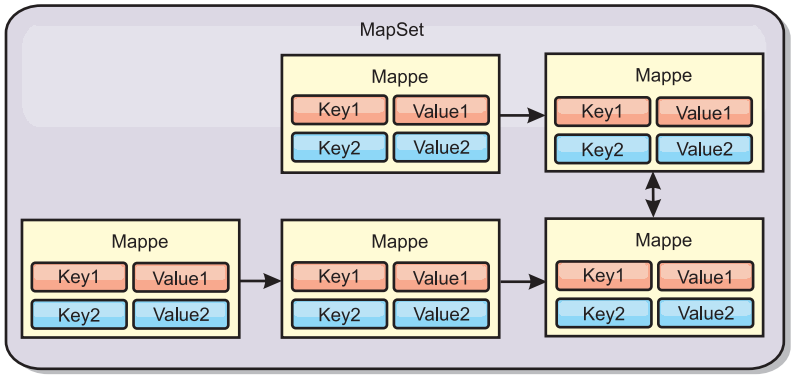


Figure 3. Groupes de mappes

Un groupe de mappes peut être associé à un schéma. Un schéma est l'ensemble des métadonnées décrivant les relations entre différentes mappes lorsque des types d'objet ou des entités hétérogènes sont utilisés.

WebSphere eXtreme Scale peut stocker des objets Java sérialisables dans chacune des mappes utilisant l'API ObjectMap. Il est possible de définir un schéma sur les mappes pour identifier la relation entre les objets dans les mappes contenant des objets d'un seul type. Il est nécessaire de définir un schéma pour pouvoir interroger le contenu des objets de la mappe. Plusieurs schémas de mappes peuvent être définis pour WebSphere eXtreme Scale. Pour obtenir plus de détails, consultez les informations sur l'API ObjectMap contenues dans le manuel *Guide de programmation*.

WebSphere eXtreme Scale peut également stocker des entités à l'aide de l'API EntityManager. Chaque entité est associée à une mappe. Le schéma d'un groupe de mappes d'entités est automatiquement reconnu à l'aide d'un fichier XML descripteur d'entité ou de classes Java annotées. Chaque entité est associée à un ensemble d'attributs clés et à un ensemble d'attributs non clés. Des relations peuvent aussi exister entre une entité et d'autres entités. WebSphere eXtreme Scale prend en charge les relations one-to-one, one-to-many, many-to-one et many-to-many. Chaque entité est physiquement mappée vers une seule mappe du groupe de mappes. Grâce aux entités, la présence de graphes d'objets complexes s'étendant sur plusieurs mappes est possible dans les applications. Une topologie répartie permet la coexistence de plusieurs schémas d'entités. Pour obtenir plus de détails, consultez les informations sur l'API EntityManager contenues dans le manuel *Guide de programmation*.

Conteneurs, partitions et fragments

Le conteneur est un service qui stocke les données d'application pour la grille. Ces données sont généralement fractionnées en différentes parties appelées partitions et hébergées par plusieurs conteneurs. A son tour, chaque conteneur héberge un sous-ensemble de données. Une machine virtuelle Java peut héberger un ou plusieurs conteneurs et chaque conteneur peut héberger plusieurs fragments.

A faire : Planifiez la taille des segments de mémoire des conteneurs qui hébergent l'ensemble de vos données. Configurez en conséquence les paramètres du segment de mémoire.

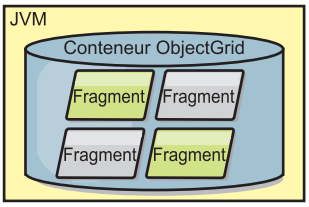


Figure 4. Conteneur

Les partitions hébergent un sous-ensemble des données dans la grille. WebSphere eXtreme Scale place automatiquement plusieurs partitions dans un même conteneur et les répartit différemment au fur et à mesure que des conteneurs deviennent disponibles.

Important : Choisissez soigneusement le nombre de partitions avant le déploiement final, car ce nombre ne peut pas être modifié dynamiquement. Un mécanisme de hachage permet de localiser les partitions dans le réseau et eXtreme Scale ne peut pas hacher à nouveau l'ensemble des données après leur déploiement. En règle générale, vous pouvez surestimer le nombre de partitions

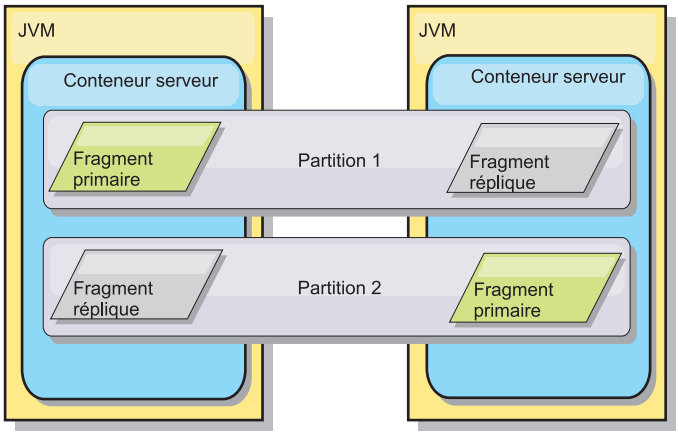


Figure 5. Partition

Les fragments sont des instances de partitions. Ils peuvent avoir un rôle primaire ou un rôle de réplique. Le fragment primaire et ses fragments répliques constituent la manifestation physique de la partition. Chaque partition contient plusieurs fragments, dont chacun héberge toutes les données contenues dans celle-ci. L'un des fragments est le fragment primaire, les autres sont les fragments répliques, c'est-à-dire des copies redondantes des données du fragment primaire. Le fragment primaire est la seule instance de partition permettant à des transactions d'écrire dans le cache. Un fragment réplique est une instance "miroir" de la partition. Il reçoit des mises à jour du fragment primaire de manière synchrone ou asynchrone. Le fragment réplique autorise uniquement les transactions à lire à partir du cache. Les fragments répliques ne sont jamais hébergés dans le même conteneur ni sur la même machine que le fragment primaire.

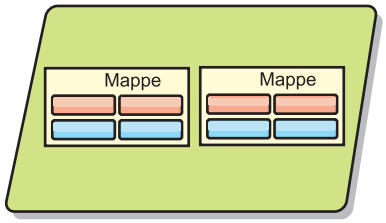


Figure 6. Fragment

Pour améliorer la disponibilité des données ou garantir la persistance, répliquez les données. La réplication augmente toutefois le coût des transactions et améliore les performances au détriment de la disponibilité. Avec eXtreme Scale, vous pouvez contrôler les coûts car les réplications synchrones et asynchrones sont prises en charge, ainsi que les modèles de réplication hybrides utilisant les deux modes de réplication. Un fragment réplique synchrone reçoit des mises à jour lors de la transaction du fragment primaire visant à garantir la cohérence des données. Un fragment réplique synchrone peut doubler le temps de réponse car la transaction doit valider le fragment primaire et le fragment réplique synchrone avant que la transaction se termine. Un fragment réplique asynchrone reçoit des mises à jour après que la transaction a validé la limitation de l'impact sur les performances, mais introduit la possibilité de perte de données car les fragments réplique asynchrones peuvent impliquer le traitement de plusieurs transactions après le fragment primaire.

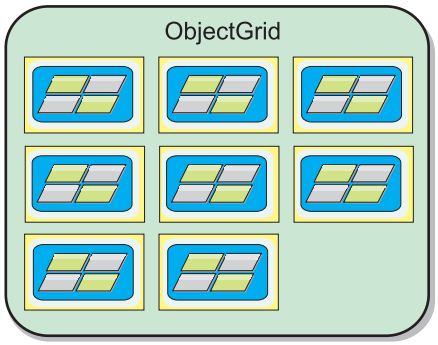


Figure 7. ObjectGrid

Clients

Les clients se connectent à un service de catalogue, extraient une description de la topologie du serveur et communiquent directement avec chaque serveur. Lorsque la topologie du serveur est modifiée en raison de l'ajout de nouveaux serveurs ou de la défaillance de certains serveurs existants, le service de catalogue dynamique achemine le client vers le serveur hébergeant les données approprié. Les clients doivent examiner les clés des données d'application pour déterminer vers quelle partition la demande doit être acheminée. Les clients peuvent lire les données de plusieurs partitions dans une même transaction. Ils peuvent cependant mettre à jour une seule partition dans une transaction. Une fois que le client a mis à jour certaines entrées, la transaction client doit utiliser cette partition pour les mises à jour.

La liste suivante répertorie les combinaisons de déploiement possibles :

- Un service de catalogue existe dans sa propre grille de machines virtuelles Java. Le même service de catalogue peut être utilisé pour gérer plusieurs clients ou serveurs eXtreme Scale.
- Un conteneur peut être démarré dans une JVM ou chargé dans une JVM arbitraire avec d'autres conteneurs destinés à des instances ObjectGrid différentes.
- Un client peut exister dans une JVM et communiquer avec une ou plusieurs instances ObjectGrid. Un client peut également exister dans la même JVM que le conteneur.

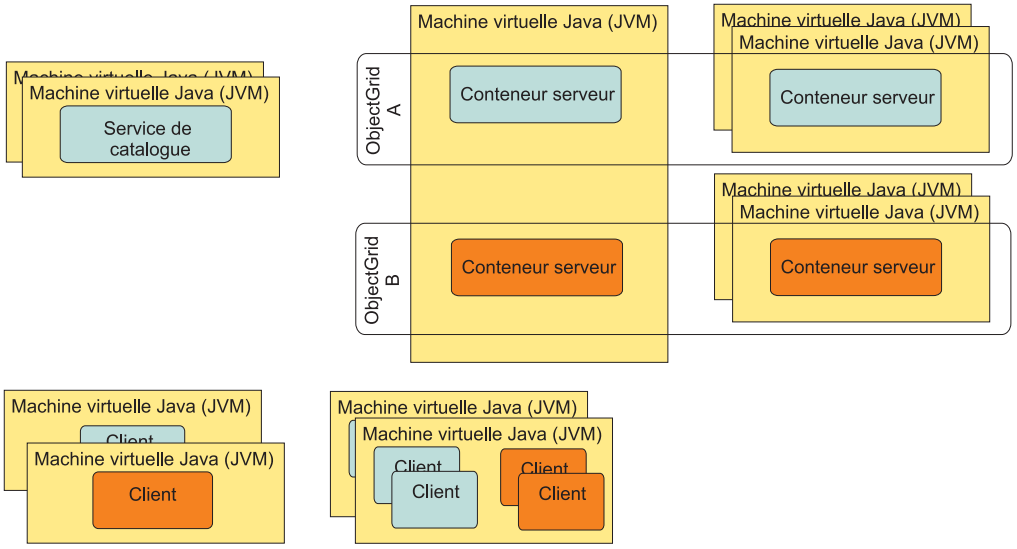


Figure 8. Topologies possibles

Services de catalogue (serveurs de catalogue)

Le service de catalogue héberge une logique qui doit être inactive lorsque l'état est stabilisé et qui a peu d'influence sur l'évolutivité. Le service de catalogue est généré pour gérer plusieurs centaines de conteneurs devenant disponibles simultanément. Il exécute des services en vue de leur gestion.

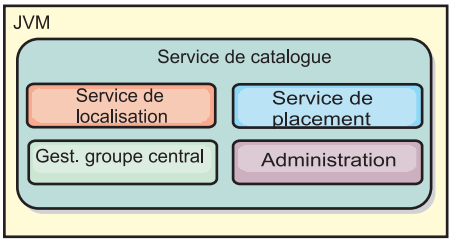


Figure 9. Service de catalogue

Les responsabilités du catalogue de service sont les suivantes :

Service de localisation

Le service de localisation fournit une localité aux clients qui recherchent des conteneurs hébergeant des applications et aux conteneurs cherchant à

enregistrer des applications hébergées à l'aide du service de positionnement. Il s'exécute dans tous les membres de la grille et permet de supprimer cette fonction.

Service de positionnement
Le service de positionnement constitue le système nerveux central de la grille. Il est responsable de l'allocation des fragments à leur conteneur hôte. Il s'exécute en tant que service Un sur N choisi dans le cluster. Comme c'est la règle Un sur N qui est utilisée, il y a toujours une et une seule instance de ce service en cours d'exécution. Si cette instance doit s'arrêter, un autre processus prend la relève. A des fins de redondance, tous les états du service de catalogue sont répliqués sur tous les serveurs hébergeant le service de catalogue.

Gestionnaire du groupe central
Le gestionnaire du groupe central gère le regroupement homologue en vue de la surveillance de la santé, organise les conteneurs en petits groupes de serveurs et fédère automatiquement les groupes de serveurs. Lorsqu'un conteneur contacte le service de catalogue pour la première fois, le conteneur attend d'être affecté à un nouveau groupe ou à un groupe existant de plusieurs machines virtuelles Java. Chaque groupe de machines virtuelles Java surveille la disponibilité de chacun de ses membres à l'aide du signal de présence. L'un des membres du groupe relaie les informations sur la disponibilité au service de catalogue afin de lui permettre de réagir aux défaillances en procédant à des réallocations et à des réacheminements.

Administration
L'administration de l'environnement WebSphere eXtreme Scale se compose de quatre phases : la planification, le déploiement, la gestion et la surveillance. Pour plus d'informations sur chacune de ces étapes, consultez le manuel *Guide d'administration*.

Pour la disponibilité, configurez un domaine de services de catalogue. Un domaine de services de catalogue consiste en plusieurs machines virtuelles Java, dont une machine maîtresse et plusieurs machines virtuelles Java de sauvegarde.

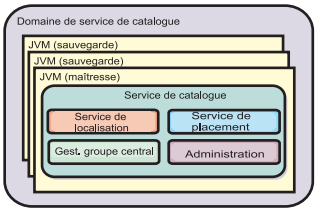


Figure 10. Domaine de services de catalogue

Topologie des mises en cache : mise en cache en mémoire et mise en cache par répartition

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.

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 le serveur eXtreme Scale et les clients se connectent à distance à ce serveur.

Les caches répartis permettent d'améliorer les performances, la disponibilité et l'évolutivité du système. Les topologies dynamiques utilisées pour les configurer permettent d'équilibrer automatiquement les serveurs. Vous pouvez également ajouter d'autres serveurs sans redémarrer les serveurs eXtreme Scale existants. 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 locale

Dans le cas le plus simple, 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 locale eXtreme Scale peuvent être indexées et extraites à l'aide du support de requête de WebSphere eXtreme Scale. La fonction d'interrogation des données peut représenter un outil précieux pour les développeurs utilisant des fichiers volumineux stockés dans la mémoire contrairement au support de structure de données limité offert par la machine virtuelle Java (JVM), prête à l'utilisation en l'état.

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

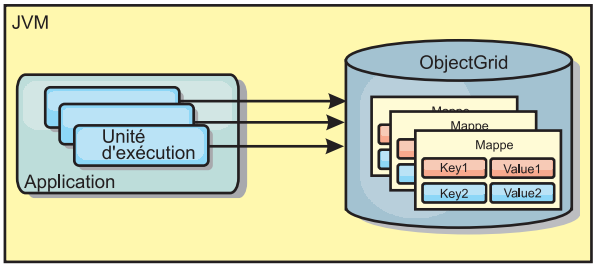


Figure 11. Scénario de cache local en mémoire

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 interne en local répliqué sur des homologues

Dans le cas d'un cache WebSphere eXtreme Scale local, vous devez vous assurer que le cache est bien synchronisé s'il existe plusieurs processus avec des instances indépendantes de cache. Pour ce faire, activez avec JMS un cache répliqué sur des homologues.

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 Java Messaging Service (JMS).

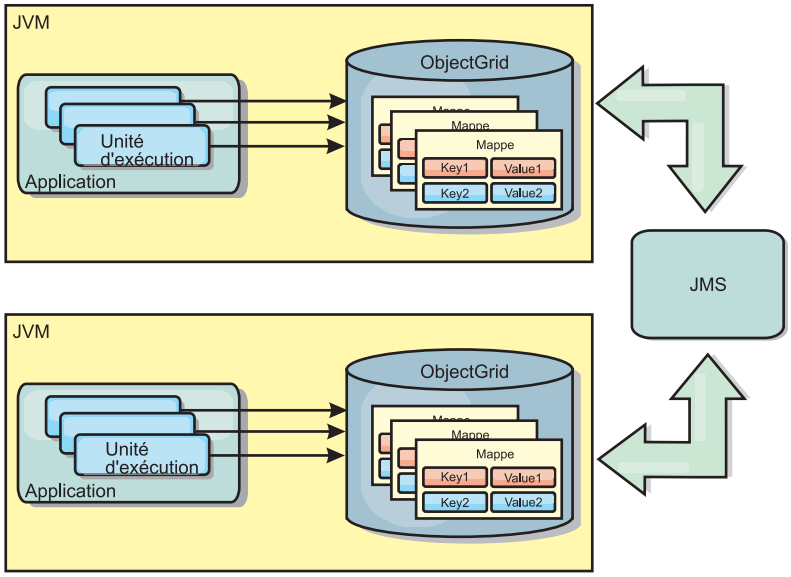


Figure 12. Cache répliqué sur des homologues avec des modifications qui sont propagées à l'aide de JMS

Si vous tournez en environnement WebSphere Application Server, vous pouvez également utiliser le plug-in TranPropListener. Ce plug-in utilise le gestionnaire de haute disponibilité (HA) pour propager les modifications à chacune des instances de cache eXtreme Scale homologue.

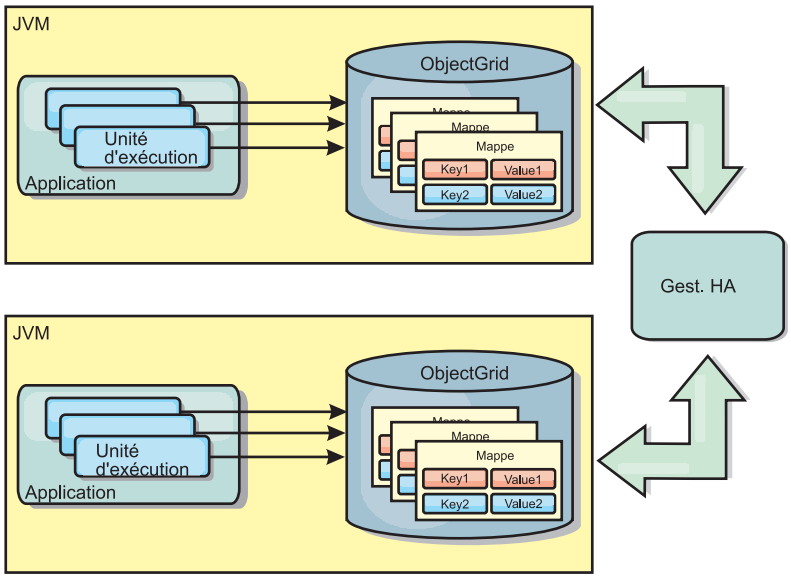


Figure 13. Cache répliqué sur des homologues avec des modifications qui sont propagées à l'aide du gestionnaire de haute disponibilité

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

Cette topologie de déploiement ne doit être utilisée que lorsque la quantité de données à mettre en cache est de taille réduite (qu'elle peut tenir sur une seule machine virtuelle Java) et qu'elles sont relativement stables.

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.

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 également davantage de données au fur et à mesure que des serveurs sont ajoutés à la grille et que le cache évolue de façon linéaire en même temps que la grille s'agrandit. Comme les clients accèdent aux données de cette grille à l'aide d'appels procéduraux, cette mémoire est également appelée cache distant (ou éloigné). Grâce au partitionnement des données, chaque processus contient un sous-ensemble unique de données. Les grilles de grande taille peuvent contenir davantage de données et gérer un plus grand nombre de demandes. Par ailleurs, il n'est plus nécessaire d'insérer les données d'invalidation autour de la grille 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.

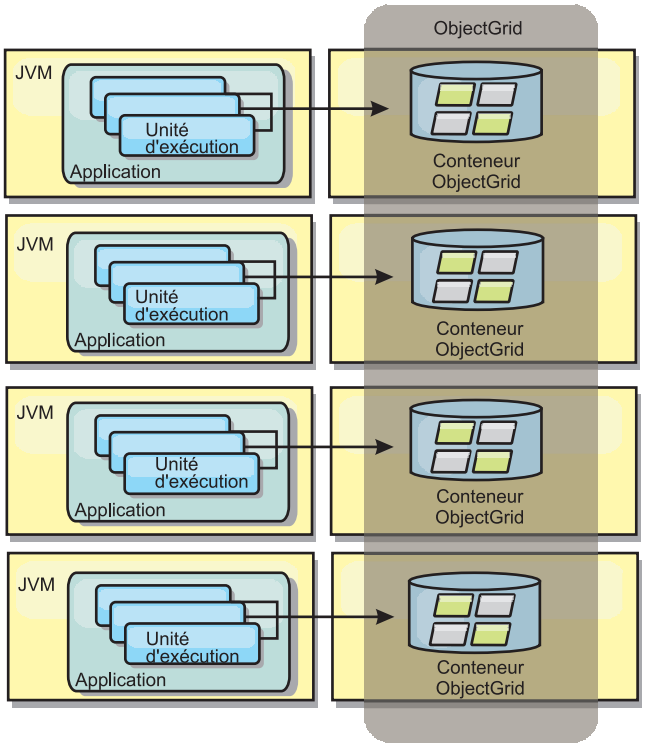


Figure 14. Cache réparti

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.

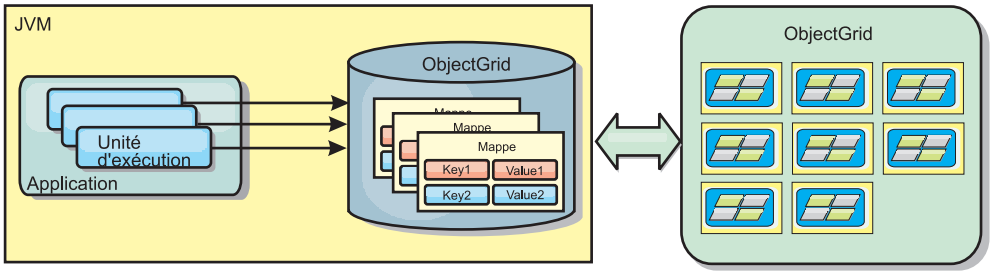


Figure 15. Cache local

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 de données, celles-ci, si elles existent, sont extraites du cache local et insérées dans le cache de transactions.
3. La grille du groupe des serveurs contient la majorité des données. 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

Inconvénients

- longévité plus importante des données périmées
- obligation d'utiliser un expulseur pour invalider les données et éviter ainsi de manquer de mémoire

Utilisation

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

Cache imbriqué

Les grilles eXtreme Scale peuvent s'exécuter dans des processus existants tels que des serveurs eXtreme Scale imbriqués. Elles peuvent également être gérées en tant que 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.

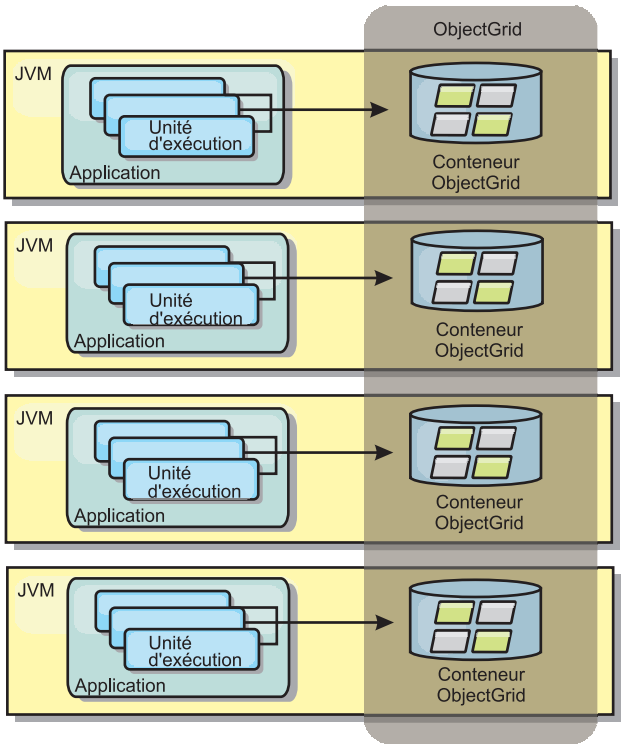


Figure 16. Cache imbriqué

Avantages

- simplification de l'administration en raison du nombre inférieur de processus à gérer
- simplification du déploiement d'applications en raison de l'utilisation par la grille du chargeur de classe d'application du 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.

Pour plus d'informations, consultez la rubrique relative à l'activation du mécanisme d'invalidation du client dans le manuel *Guide d'administration*.

Topologies de réplication de grilles multi-maîtres (PA)

La réplication asynchrone multi-maître permet à deux grilles, voire plus, de devenir des miroirs exacts les unes des autres. Cette mise en miroir est effectuée à l'aide d'une réplication asynchrone entre des liens interconnectant les grilles. Chaque grille est hébergée au sein d'un domaine complètement indépendant, possédant son propre service de catalogue et ses propres serveurs conteneurs et portant un nom exclusif. La réplication asynchrone multi-maître permet d'utiliser des liens pour interconnecter une collection de ces domaines et de synchroniser ensuite ces derniers grâce, précisément, à la réplication via ces liens. eXtreme Scale permet de construire quasiment n'importe quelle topologie, car la définition des liens entre les domaines est laissée à l'appréciation et à l'initiative des administrateurs.

Les domaines : des grilles avec des caractéristiques spécifiques

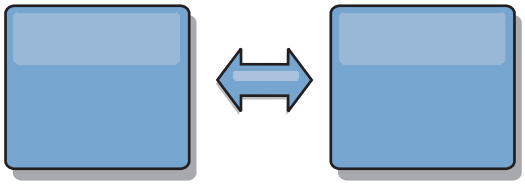
Les grilles utilisées dans les topologies de réplication multi-maître sont également désignées sous le nom de *domaines*. Chaque domaine doit avoir les caractéristiques suivantes :

- disposer d'un service de catalogue privé avec un nom de domaine exclusif
- avoir le même nom de grille que les autres grilles du domaine
- avoir le même nombre de partitions que les autres grilles du domaine
- être une grille FIXED_PARTITION (les grilles PER_CONTAINER ne peuvent être répliquées)
- avoir le même nombre de partitions (sans forcément pour autant avoir le même nombre et le même type de fragments répliques)
- avoir les mêmes types de données à répliquer que les autres grilles du domaine
- avoir le même nom de groupes de mappes (mapsets), le même nom de mappes et les mêmes modèles de mappes dynamiques que les autres grilles du domaine

Tous les groupes de mappes ayant les caractéristiques énoncées ci-dessus seront répliqués après que les domaines de services de catalogue auront été démarrés. Voir «Topologies de réplication de grilles multi-maîtres (PA)» pour connaître les groupes de mappes qui ne seront pas répliquées.

Liens connectant des domaines

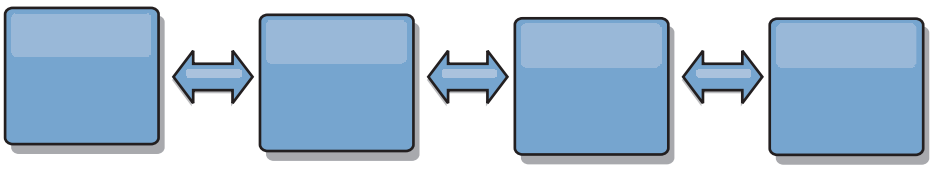
Une infrastructure de grilles de réplication est un graphe connecté de domaines reliés par des liens bidirectionnels. Un lien permet à deux domaines d'échanger des modifications de données. Ainsi, la topologie la plus simple sera une paire de domaines reliés par un seul lien. Les domaines sont nommés en partant de A, puis B, et ainsi de suite, à partir de la gauche. Le lien peut traverser un réseau WAN étendu, couvrant de longues distances. En cas d'interruption du lien, les modifications peuvent toujours être apportées aux données dans l'un ou l'autre des deux domaines. La réconciliation des modifications s'effectue plus tard, lorsque le lien recommence à connecter les domaines. Les liens tentent automatiquement de se reconnecter si la connexion réseau est interrompue.



Une fois que les liens ont été définis, eXtreme Scale va tout d'abord tenter de rendre identique chaque domaine, puis il va essayer ensuite de les maintenir dans un état identique lorsque des modifications se produiront dans l'un de ces domaines. L'objectif d'eXtreme Scale est que chaque domaine soit un miroir exact de tout autre domaine connecté par les liens. Les liens de réplication entre les domaines aident à garantir que toute modification effectuée dans un domaine est copiée vers les autres domaines.

Topologies linéaires

Bien qu'elle figure au rang des topologies les plus simples, la topologie linéaire illustre un certain nombre de qualités des liens. Tout d'abord, il n'est pas nécessaire qu'un domaine soit directement connecté à chacun des autres domaines pour recevoir des modifications. Le domaine B ira prendre des modifications dans le domaine A. Le domaine C reçoit les modifications du domaine A via le domaine B, lequel connecte les domaines A et C. De la même manière, 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.



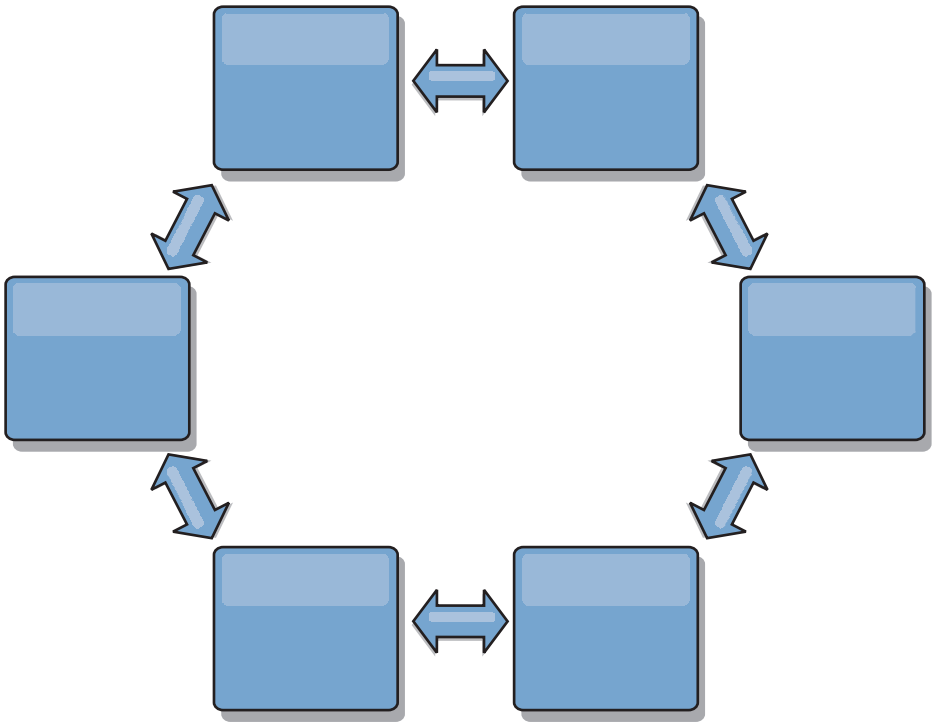
Et si le domaine C tombait en panne, les événements suivants se produiraient :

1. Le domaine D serait orphelin jusqu'au redémarrage du domaine C.
2. Le domaine C se synchroniserait avec le domaine B, lequel est une copie du domaine A.
3. Le domaine D utiliserait le domaine C pour se synchroniser avec les modifications intervenues sur les domaines A et B pendant que le domaine D était orphelin (et que le domaine C était arrêté).

A la fin, les domaines A, B, C et D redeviendraient tous identiques.

Topologies en anneau

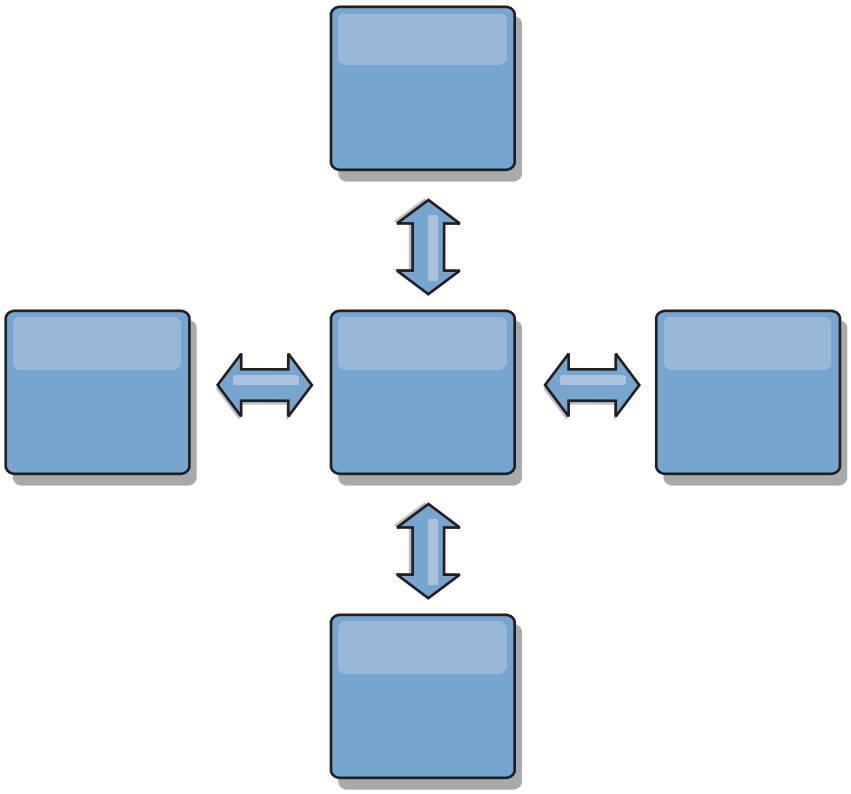
Les topologies en anneau sont un exemple de topologie encore plus résilientes. En effet, la défaillance d'un domaine ou d'un lien n'empêche pas les domaines survivants de continuer à obtenir des modifications en circulant autour de l'anneau et en s'éloignant de la défaillance. Chaque domaine a deux liens vers les autres domaines. Chaque domaine a au maximum deux liens, quelle que soit la taille de la topologie en anneau. Le délai de propagation des modifications peut être important, car les modifications d'un domaine particulier peuvent avoir besoin de traverser plusieurs domaines avant que la totalité des domaines ne puisse voir la totalité des modifications. Une topologie linéaire a le même problème.



Représentez-vous une topologie en anneau plus sophistiquée, avec un domaine racine au centre de l'anneau. Le domaine racine joue le rôle de chambre centrale de compensation tandis que les autres domaines font office de chambres distantes de compensation pour les modifications se produisant dans le domaine racine. Le domaine racine peut arbitrer les modifications entre les domaines. Si une topologie en anneau contient plusieurs anneaux autour d'un domaine racine, ce dernier ne pourra arbitrer les modifications que dans les domaines de l'anneau le plus proche du centre. Mais les résultats de l'arbitrage seront ventilés vers les domaines des autres anneaux.

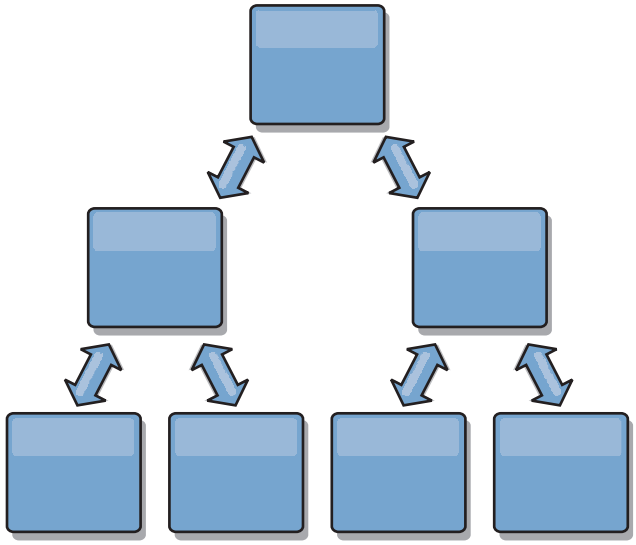
Topologies en étoile

Si, dans une topologie en étoile, les délais d'attente sont moindres, les modifications voyageant au maximum sur un domaine intermédiaire (le concentrateur), cette topologie ne va pas sans autres problèmes. Elle a un domaine central qui fait office de « moyeu » ou de concentrateur. Ce domaine concentrateur est connecté via un lien à chacun des domaines qui jouent le rôle de « rayons » de la roue. On le voit, la charge de la répartition des modifications entre les domaines repose toute entière sur le concentrateur. Ce dernier fait office de chambre de compensation en cas de collisions, ce qui peut s'avérer important dans certains scénarios. Dans un environnement soumis à une fréquence élevée de modifications, pour pouvoir rester actif et opérationnel, le concentrateur peut avoir besoin de s'exécuter sur plus de matériels que les autres noeuds. eXtreme Scale est conçu pour évoluer de manière linéaire, ce qui signifie que l'on peut, si nécessaire, étoffer le concentrateur sans difficultés. Mais, si le concentrateur vient à tomber en panne, il faudra attendre qu'il ait redémarré pour que les changements puissent être répartis. Toutes les modifications intervenues sur les domaines autres que le concentrateur seront réparties après la reconnexion de ce dernier.



Topologies en arbre

Dernier exemple de topologie : une arborescence acyclique dirigée. Par acyclique, l'on entend qu'aucun cycle (aucune boucle) ne se passe sur l'arborescence. Dirigée signifie qu'il n'existe de liens qu'entre des parents et des enfants. Cette configuration peut être utile pour les topologies comprenant tant de domaines qu'il ne serait pas pratique d'avoir un concentrateur connecté de manière centralisée à chaque ordinateur possible. Autre cas de figure où cette topologie trouve toute son utilité : le besoin d'ajouter des domaines enfants sans avoir à modifier le domaine racine.



Cette topologie peut toujours avoir une chambre de compensation centrale avec le domaine racine, mais le deuxième niveau peut faire office de chambres de compensation pour les modifications se produisant dans le domaine situé au niveau inférieur. Le domaine racine ne peut arbitrer que les modifications entre les domaines de ce deuxième niveau. Des arbres n-aires sont également possibles. Un arbre n-aire a n enfants à chaque niveau. Chaque domaine a un déploiement de n.

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 pour qu'ils aient tous la même quantité de processeurs, de mémoire et de ressources réseau. Vous vous rendrez sans doute compte que les domaines gérant les collisions de modifications (arbitrage) utilisent plus de ressources que les autres domaines. Les collisions sont détectées de manière automatique. Deux mécanismes permettent de les résoudre :

- **Arbitre par défaut**. Le protocole par défaut est d'utiliser les modifications provenant du domaine dont le nom vient en premier par ordre alphabétique. Supposons par exemple que les domaines A et B génèrent un conflit pour un enregistrement, dans ce cas, la modification du domaine B sera ignorée. Le domaine A conserve sa version et l'enregistrement dans le domaine B est modifié de manière à correspondre à celui du domaine A. Cela s'applique également pour les applications où les utilisateurs ou les sessions sont liées de manière normale ou ont une affinité avec l'une des grilles.
- **Arbitre personnalisé**. Les applications peuvent très bien fournir un arbitre personnalisé. Lorsqu'un domaine détecte une collision, il fait appel à l'arbitre. Pour savoir comment développer un arbitre personnalisé de bonne qualité, voir Développement d'arbitres personnalisés pour la réplication multi-maître.

Si des collisions doivent être possibles dans les topologies que vous envisagez, pensez à utiliser une topologie en étoile ou en arbre. Il est en effet plus facile dans ces deux topologies d'éviter des collisions sans fin, ce qui peut arriver lorsque :

1. plusieurs domaines subissent une collision
2. chaque domaine résout la collision en local, ce qui produit des révisions
3. les révisions entrent en collision, d'où des révisions de révisions
4. et ainsi de suite, au fur et à mesure que les révisions sont propagées dans les divers domaines tout en essayant d'atteindre la synchronicité

Pour éviter des collisions sans fin, choisissez un domaine spécifique – un *domaine d'arbitrage* – comme gestionnaire des collisions d'un sous-ensemble de domaines. Exemple : une topologie en étoile pourra utiliser le concentrateur comme gestionnaire des collisions. Le gestionnaire de collisions ignore les collisions détectées par les sous-domaines. Le domaine du concentrateur va créer des révisions en empêchant les révisions de collisions qui échappent à tout contrôle. Le domaine qui s'est vu attribuer la gestion des collisions doit se lier à tous les domaines dont il est chargé de résoudre les collisions. Dans une topologie en arbre, tous les domaines parents internes résolvent les collisions pour leurs enfants immédiats. Par contraste, si vous utilisez une topologie en anneau, vous ne pouvez désigner un domaine de l'anneau comme résolvant les collisions.

Le tableau qui suit récapitule les approches en matière d'arbitrage qui sont les plus compatibles avec les diverses topologies.

Tableau 3. Approches en matière d'arbitrage. Ce tableau énonce si l'arbitrage entre applications est compatible avec les diverses topologies.

Topologie	Arbitrage entre applications ?	Commentaires
Linéaire à deux domaines	Oui	Choisissez l'un des domaines comme arbitre.
Linéaire à trois domaines	Oui	L'arbitre doit être le domaine du milieu. Pensez à ce domaine comme au concentrateur d'une topologie en étoile simple.
Linéaire à plus de trois domaines	Non	L'arbitrage entre applications n'est pas pris en charge.
Concentrateur avec n "rayons"	Oui	Le domaine d'arbitrage doit être le concentrateur avec des liens vers tous les "rayons".
Anneau de n domaines	Non	L'arbitrage entre applications n'est pas pris en charge.
Arbre dirigé acyclique (arbre n-aire)	Oui	Tous les noeuds racine ne doivent arbitrer que leurs descendants directs.

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**

  Les temps d'attente des modifications sont déterminés par le nombre de domaines intermédiaires que doivent traverser les modifications avant d'arriver à destination.

  Les topologies qui offrent les meilleurs temps d'attente sont celles qui éliminent les domaines intermédiaires en liant chacun des domaines à chacun des autres domaines. Mais un domaine doit effectuer la réplication à proportion du nombre de ses liens. Pour les topologies de grande taille, le nombre de liens à définir peut constituer une lourde charge pour les administrateurs.

  La vitesse à laquelle est une modification copiée vers les autres domaines dépend de facteurs supplémentaires :

  – le processeur et la bande passante réseau sur le domaine source
  – le nombre de domaines intermédiaires et de liens entre le domaine source et le domaine cible
  – les ressources de processeur et de réseau utilisables par les domaines source, cible et intermédiaires

- **Tolérance aux pannes**

  La tolérance aux pannes est déterminée par le nombre de chemins existant entre deux domaines pour la réplication des modifications.

  S'il n'existe qu'un seul lien entre les domaines, en cas de défaillance du lien, les modifications ne seront pas propagées. Si ce seul lien entre deux domaines passe par des domaines intermédiaires, les modifications ne seront pas non plus propagées si l'un des domaines intermédiaires tombe en panne.

  Prenons la topologie linéaire à trois domaines, A, B et C :

  ```
  A <-> B <-> C
  ```

Si l'une des situations suivantes se présente, le domaine C ne verra pas les modifications de A :

– le domaine A est actif et le domaine B est arrêté
– le lien entre A et B ne fonctionne pas
– le lien entre B et C ne fonctionne pas

Par contraste, une topologie en anneau permet à chaque domaine d'extraire les modifications dans un sens ou dans l'autre.

```
A <-> B <-> C <-> retour vers A
```

Par exemple, si le domaine B est arrêté, le domaine C continue d'extraire les modifications directement depuis le domaine A.

Une conception en étoile dépend du concentrateur par qui passent toutes les modifications ; s'il s'arrête, tout s'arrête. Mais il ne faut pas oublier qu'un domaine reste une grille totalement tolérante aux pannes et qui peut souffrir des défaillances du réseau WAN ou des centres de données physiques.

- **Performances**

  Le nombre des liens définis sur un domaine affecte les performances. Plus de liens utilisent davantage de ressources, et cela peut se traduire par une chute des performances de la réplication. La capacité d'un domaine A à extraire les modifications via d'autres domaines décharge efficacement ce domaine A de répliquer partout ses transactions. *La charge de la répartition des modifications sur un domaine est limitée au nombre des liens qu'utilise ce domaine. Cela n'a rien à voir avec le nombre de domaines dans la topologie.* Cette propriété permet l'évolutivité et autorise à partager la charge de la répartition des modifications entre les domaines de la topologie, plutôt que d'imposer cette charge à un seul domaine.

  Un domaine peut extraire les modifications indirectement via d'autres domaines. Prenons une topologie linéaire à cinq domaines :

  ```
  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 et les modifications de A, B et C via D

  La charge de la répartition sur les domaines A et E est la plus faible, car ces domaines ont chacun un seul lien avec un seul domaine. La charge de la répartition sur les domaines B, C et D est le double de celle sur les domaines A et E car B, C et D ont chacun un lien avec deux domaines. Cette répartition des charges demeurerait constante même si la topologie linéaire contenait 1 000 domaines, car la charge dépend du nombre de liens de chaque domaine et non du nombre globale de domaines dans la topologie.

Points à prendre en considération concernant les performances

Les limitations suivantes sont à prendre en compte pour les topologies de réplication multi-maître.

- **Optimisation de la répartition des modifications** (abordée plus haut).
- **Temps d'attente de la réplication** (abordée plus haut).

- **Performances des liens de réplication** eXtreme Scale crée un seul socket TCP/IP entre n'importe quelle paire de machines virtuelles Java. Tout le trafic entre ces machines virtuelles Java passe par ce socket, y compris la réplication multi-maître. Les domaines étant hébergés sur au moins n machines virtuelles Java conteneurs, fournissant au moins n liens TCP vers les domaines homologues, les domaines ayant le plus grand nombre de conteneurs sont ceux qui offrent les plus hauts niveaux de performances pour la réplication. Plus de conteneurs est synonyme de davantage de ressources processeur et réseau.
- **Optimisation de la fenêtre dynamique TCP et RFC 1323** Activer la prise en charge de la RFC 1323 aux deux extrémités d'un lien permet à davantage de données d'effectuer des aller-retour, ce qui donne des débits plus élevés. Cette technique étend les capacités de la fenêtre d'un facteur d'environ 16 000.

  N'oubliez pas que les sockets TCP utilisent un mécanisme de fenêtre dynamique pour contrôle le flux des données en vrac, qui limite normalement le socket à 64 Ko pour un intervalle d'aller-retour. Si cet intervalle 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 les débit sur les liens à hauts temps d'attente.

  Plusieurs facteurs peuvent affecter les performances de la réplication :
  - la vitesse à laquelle eXtreme Scale peut extraire les modifications
  - la vitesse à laquelle eXtreme Scale peut extraire les demandes de réplication
  - la capacité de la fenêtre dynamique
  - l'optimisation de la mémoire tampon réseau des deux côtés d'un lien pour permettre à eXtreme Scale d'extraire les modifications sur le socket à la vitesse maximale possible
- **Sérialisation des objets** Toutes les données doivent être sérialisables. Si un domaine n'utilise pas COPY_TO_BYTES, il doit utiliser la sérialisation Java ou ObjectTransformers pour optimiser les performances de la sérialisation.
- **Compression** eXtreme Scale compresse par défaut toutes les données envoyées entre les domaines. La version actuelle ne permet pas de désactiver la compression.
- **Optimisation de la mémoire** *L'utilisation de la mémoire pour une topologie de réplication multi-maître est largement indépendante du nombre de domaines présents dans la topologie.*

  Activer la réplication multi-maître ajoute du temps système fixe par entrée de mappe pour gérer la vérification des versions. Chaque conteneur suit également une quantité fixe de données pour chacun des domaines de la topologie. Une topologie à deux domaines utilise approximativement la même mémoire qu'une topologie à cinquante domaines. eXtreme Scale n'utilise pas dans son implémentation de replay logs ou d'autres files d'attente du même genre. Cela veut dire que, si un lien de réplication est indisponible pendant une période assez longue, il n'y a aucune structure de données en train de croître en taille dans l'attente d'une reprise de la réplication au redémarrage du lien.

Centres de données multiples avec FIXED_PARTITION

Vous pouvez à présent utiliser une grille FIXED_PARTITION entre au moins deux centres de données. Chaque centre de données a besoin de son propre domaine, en termes de réplication multi-maître. Chaque centre de données peut lire et écrire des données sur le domaine local. Ces modifications seront propagées vers l'autre

centre de données à l'aide des liens que vous aurez définis.

Clients intégralement répliqués

Cette variante de la topologie implique une paire de serveurs eXtreme Scale s'exécutant comme concentrateur. Chaque client crée une grille à conteneur unique autonome avec un catalogue dans sa machine virtuelle Java. Un client utilise sa grille 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 seront réparties rapidement via le concentrateur entre les machines virtuelles Java clients. 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.

Limitations

Les limitations suivantes sont à prendre en compte pour décider s'il y a lieu d'utiliser des topologies de réplication multi-maître et, si oui, quand.

- **Précautions à prendre dans la configuration de chargeurs de classes avec plusieurs domaines**

  Les domaines doivent avoir accès à toutes les classes qui sont utilisées comme clés et comme valeurs. Toutes les dépendances doivent être reflétées dans tous les chemins de classes des machines virtuelles Java conteneurs de grille de la totalité des 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 fait appel à l'arbitre.
- **L'utilisation de chargeurs n'est pas recommandée**

  Les chargeurs peuvent servir à l'interfaçage des modifications entre une grille et une base de données. Il y a fort peu de probabilités que toutes les grilles (domaines) d'une topologie cohabitent géographiquement avec la même base de données. De toute façon, vu les temps d'attente du WAN et d'autres facteurs, ce cas de figure est peu souhaitable.

  Autre problème qui nécessite que l'on aborde la conception avec précaution : le préchargement des grilles. D'ordinaire, lorsqu'elle redémarre, une grille est préchargée à nouveau. Le préchargement n'est pas nécessaire ni même souhaitable lorsqu'on utilise la réplication multi-maître. Dès qu'un domaine est en ligne, il se recharge automatiquement avec le contenu des domaines auxquels il est lié. Il en découle qu'il n'est pas nécessaire de lancer un préchargement manuel d'une grille qui est un domaine dans une topologie de réplication multi-maître.

Les chargeurs obéissent en général à des règles d'insertion et d'actualisation. Dans le cadre de la réplication multi-maître, les insertions doivent être traitées comme des fusions. Lorsque les données sont extraites à distance après le redémarrage d'un domaine, les données existantes seront "insérées" dans le domaine local. Comme il se peut que ces données se trouvent déjà dans la base de données locale, une insertion classique dans la base de données échouera avec une exception de clé en double. Plutôt que des insertions classiques, il faut utiliser une sémantique des fusions.

Il est possible de configurer eXtreme Scale pour qu'il effectue des préchargements en fonction des fragments en utilisant les méthodes preload des plug-in Loader. N'utilisez pas cette technique dans une topologie de réplication multi-maître. Utilisez plutôt un préchargement en fonction du client lorsque la topologie est démarrée (initialement). Autorisez la topologie multi-maître à actualiser les domaines redémarrés à l'aide d'une copie actuelle de ce qui est stocké dans les autres domaines de la topologie. Après que les domaines ont été redémarrés, la responsabilité de leur synchronisation incombera à la topologie multi-maître.

- **Pas de prise en charge d'EntityManager**

  Un groupe de mappes contenant une mappe d'entités n'est pas répliqué entre les domaines.

- **Pas de prise en charge des mappes de tableaux d'octets**

  Un groupe de mappes contenant une mappe qui est configurée avec COPY_TO_BYTES n'est pas répliqué entre les domaines.

- **Pas de prise en charge de l'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 domaines.

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 eXtreme Scale pour qu'il fonctionne en tant qu'espace extrêmement flexible de traitement de base de données en mémoire. Cela dit, eXtreme Scale n'est pas un associateur relationnel d'objets. Il ignore l'origine des données d'eXtreme Scale. 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 quu'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.

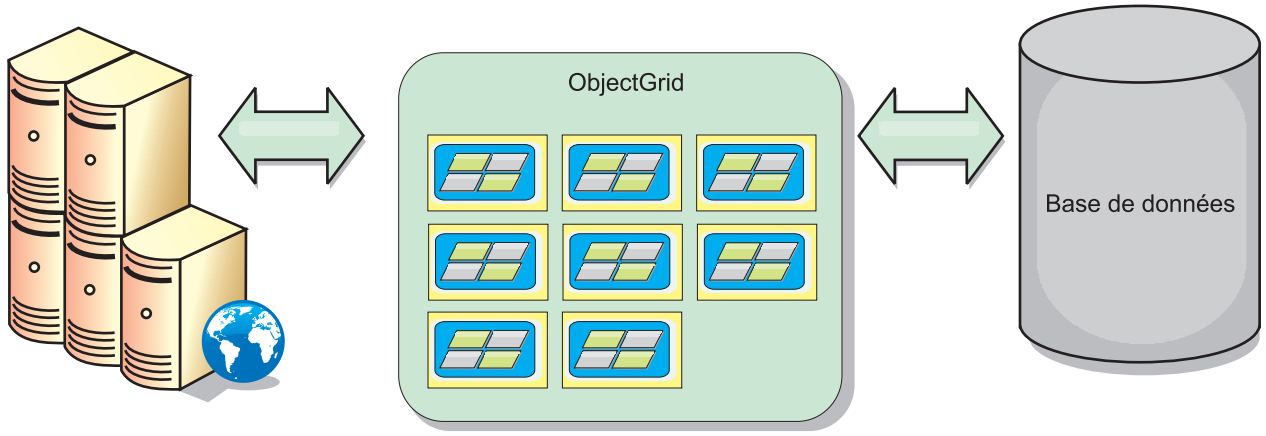


Figure 17. ObjectGrid en tant que mémoire tampon de base de données

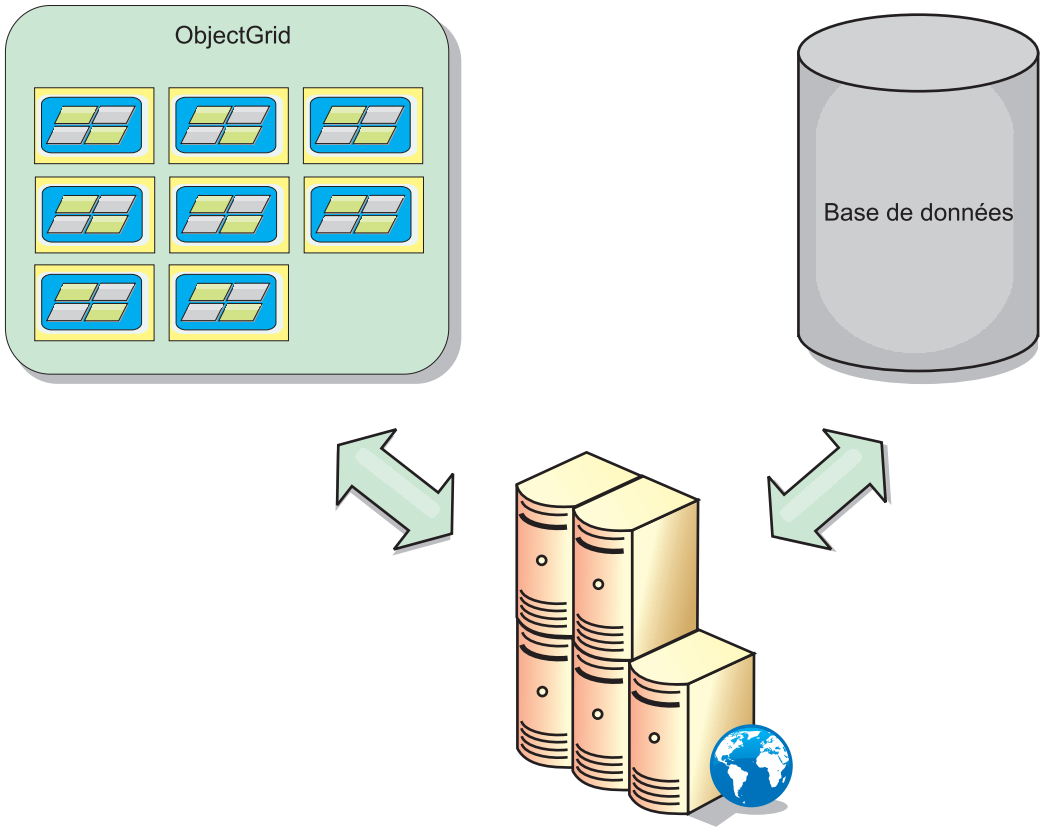


Figure 18. ObjectGrid en tant que cache secondaire

Cache partiel et cache complet

WebSphere eXtreme Scale peut s'utiliser en tant que cache partiel ou que cache complet. A la différence du cache complet qui conserve la totalité des données, le cache partiel ne conserve qu'un sous-ensemble des données totales et il peut être rempli à la demande en lazy loading. En règle générale, l'accès aux caches partiels s'effectue à l'aide de clés (et non à l'aide d'index ou de requêtes), dans la mesure où les données ne sont que partiellement disponibles.

Si une clé est absente (absence de cache), le groupe de serveurs suivant est appelé et les données sont extraites, puis insérées dans le groupe de caches respectif. En cas d'utilisation d'une requête ou d'un index, l'accès s'effectue uniquement sur les valeurs actuellement chargées et les requêtes ne sont pas transférées aux autres groupes de serveurs. 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 avant que les applications ne les utilisent et peut fonctionner en tant que remplacement de base de données. Une fois les données chargées, vous pouvez les traiter comme une base de données. Puisque toutes les données sont disponibles, les requêtes et les index peuvent être utilisés pour rechercher et agréger les données.

Cache secondaire et cache en ligne

WebSphere eXtreme Scale peut proposer une fonction de mise en cache en ligne pour la base de données dorsale ou être utilisé en tant que cache secondaire. La mise en cache en ligne utilise eXtreme Scale comme moyen principal d'interaction avec les données. Lorsqu'eXtreme Scale est utilisé en tant que cache secondaire, le système dorsal est utilisée conjointement avec eXtreme Scale.

Cache secondaire

eXtreme Scale peut être utilisé en tant que cache secondaire pour la couche d'accès aux données d'une application. Dans ce scénario, eXtreme Scale permet de stocker temporairement des objets qui seraient normalement extraits d'une base de données dorsale. Les applications vérifient qu'eXtreme Scale contient les données voulues. Si tel est le cas, ces données sont renvoyées à l'appelant. Dans le cas contraire, les données sont extraites du système dorsal et insérées dans eXtreme Scale afin que la demande suivante puisse utiliser la copie mise en cache. Le diagramme suivant illustre comment 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

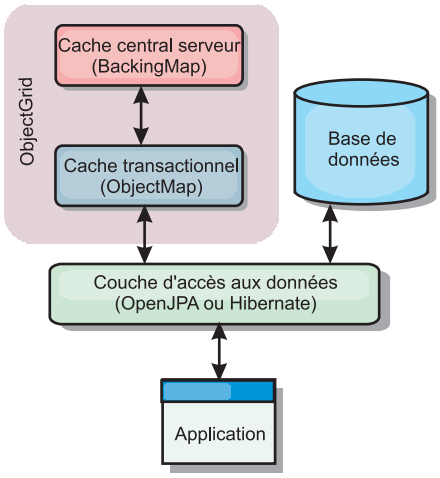


Figure 19. Cache secondaire

Les plug-in de cache pour OpenJPA et Hibernate sont inclus dans eXtreme Scale, ce qui vous permet d'utiliser eXtreme Scale en tant que cache secondaire automatique. L'utilisation d'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. 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 cache, tous les autres clients peuvent l'utiliser.

Cache en ligne

Lorsqu'eXtreme Scale est utilisé en tant que cache en ligne, il interagit avec le système dorsal à l'aide d'un plug-in de chargeur. Ce scénario permet de simplifier l'accès aux données en autorisant les demandes d'accès direct aux API eXtreme Scale. Plusieurs scénarios de mises en cache sont pris en charge par eXtreme Scale afin de garantir la synchronisation des données du cache et des données du système dorsal. Le diagramme suivant illustre l'interaction entre le cache en ligne, l'application et le système dorsal.

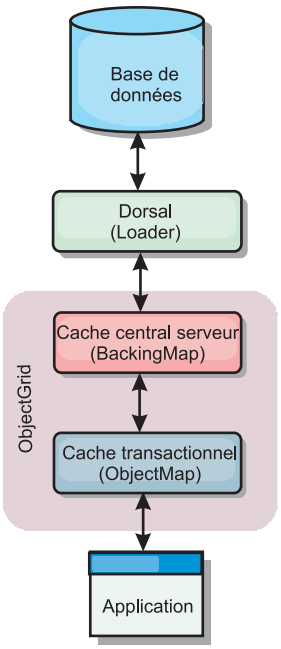


Figure 20. Cache en ligne

Mise en cache en ligne

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 du plug-in Loader.

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.

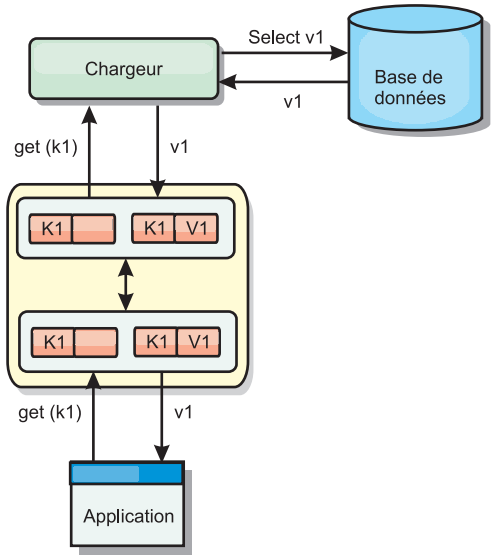


Figure 21. Mise en cache sans interruption

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.

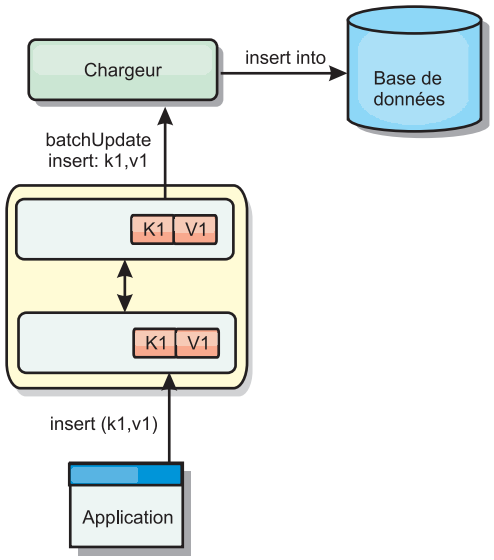


Figure 22. Mise en cache à écriture immédiate

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. Pour plus d'informations, voir «Mise en cache à écriture différée», à la page 39.

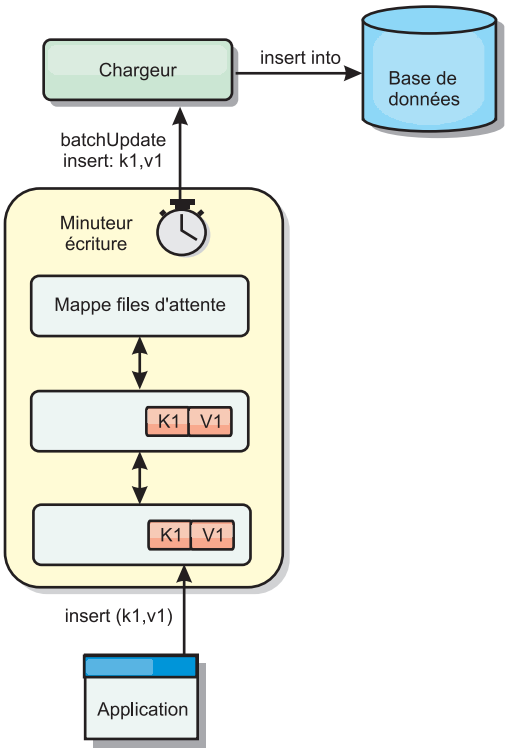


Figure 23. Mise en cache en écriture différée

Pour plus d'informations, voir «Mise en cache à écriture différée».

Mise en cache à écriture différée

Vous pouvez utiliser la mise en cache à 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.

Introduction

La mise en cache à écriture différée met les mises à jour en file d'attente de manière asynchrone dans le plug-in du Loader. Une amélioration des performances est possible si vous déconnectez les mises à jour, les insertions et les suppressions pour une mappe, le temps système supplémentaire de mise à jour de la base de données dorsale. La mise à jour asynchrone est exécutée après un délai exprimé par une durée (par exemple cinq minutes) ou un délai exprimé par un nombre d'entrées (1000 entrées).

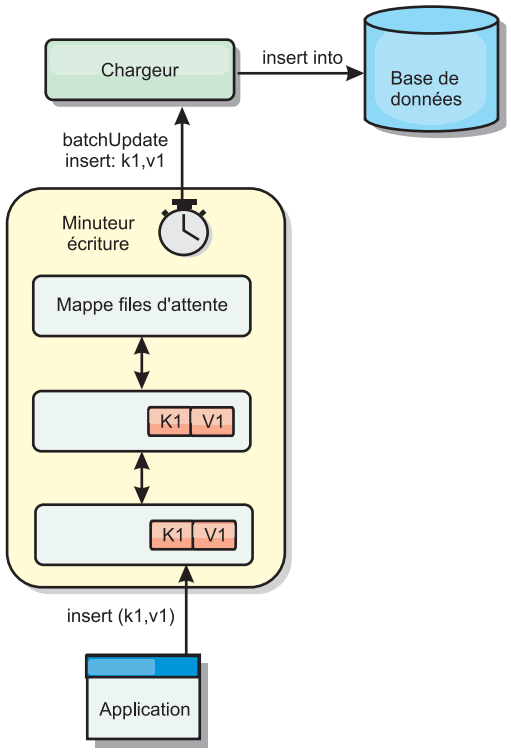


Figure 24. Mise en cache en écriture différée

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.

Avantages

L'activation de la prise en charge 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 la prise en charge de l'écriture différée est simple, mais la conception d'une application devant prendre en charge ce type d'écriture demande réflexion. 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 la prise en charge de l'écriture différée possède sa propre unité d'exécution à écriture différée permettant d'insérer les données dans 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 clé1 dans la mappe Mappe1 et la clé clé2 dans la mappe Mappe2. 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.

Comportement de la mappe de files d'attente en matière de verrouillage

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.

Du fait de ce comportement supplémentaire, vous pouvez constater des différences dans les comportements relatifs au verrouillage.

- Si la mappe des utilisateurs est configurée en tant que stratégie de verrouillage de type pessimiste, la différence de comportement est peu importante. A chaque appel d'un vidage ou d'une validation, un verrou S est placé sur la même clé de la mappe de files d'attente. Pendant la durée de la validation, un verrou X est acquis pour la clé de la mappe des utilisateurs et pour la clé de la mappe de files d'attente.
- Si la mappe des utilisateurs est configurée en tant que stratégie de verrouillage de type OPTIMISTIC ou NONE, la transaction utilisateur suit le modèle de la stratégie de verrouillage de type PESSIMISTIC. A chaque appel d'un vidage ou d'une validation, un verrou S est acquis pour la même clé de la mappe de files d'attente. Pendant la durée de la validation, un verrou X est acquis pour la clé de la mappe de files d'attente utilisant la même transaction.

Nouvelles tentatives des transactions 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.

Echec du chargeur

Un échec du plug-in du chargeur est possible lorsque celui-ci ne parvient pas à communiquer avec la base de données dorsale. Cette situation peut se produire si la connexion réseau ou le serveur de base de données est inactif. Le chargeur à écriture différée met les mises à jour en file d'attente et tente périodiquement d'insérer les modifications apportées aux données dans le 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 détectée par le chargeur à écriture différée, celui-ci considère que l'échec est lié aux données, par exemple si une clé en double a été identifiée. Le chargeur à écriture différée détaille la mise à jour et examine chaque enregistrement séparément pour isoler la raison de l'échec. 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 en écriture différée attend 15 secondes, puis tente à nouveau la mise à jour par lots de la base de données.

Une erreur fréquente consiste à générer une exception LoaderException alors qu'une exception LoaderNotAvailableException serait plus appropriée. 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.

Considérations sur les performances

La prise en charge de la mise en cache à écriture différée augmente le temps de réponse en supprimant la mise à jour du chargeur de la transaction. Elle permet aussi d'améliorer le débit de la base de données en combinant plusieurs bases de données. 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 nombre maximal de mises à jour ou leur durée maximale doivent être ajustés en fonction des modèles d'utilisation et de l'environnement attendus. Si la valeur associée à ce nombre ou à cette durée est trop faible, le temps système supplémentaire nécessaire à l'unité d'exécution risque d'être supérieur aux avantages. Le choix d'une valeur élevée pour ces deux paramètres permet d'améliorer l'utilisation de la mémoire lors de la mise en file d'attente des données et retarder le moment de péremption des enregistrements de la base de données.

Pour des performances optimales, réglez les paramètres de l'écriture différée selon les facteurs suivants :

- Rapport entre les transactions de lecture et d'écriture
- Fréquence identique de mise à jour des enregistrements
- Temps d'attente de la mise à jour de la base de données

Chargeurs

Avec un plug-in Loader d'eXtreme Scale, une mappe eXtreme Scale peut être utilisée comme cache pour les données généralement conservées dans un stockage de persistance sur le même système ou un autre système. Généralement, une base de données ou un système de fichiers est utilisé comme stockage de persistance. Une machine virtuelle Java (JVM) 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 les 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.

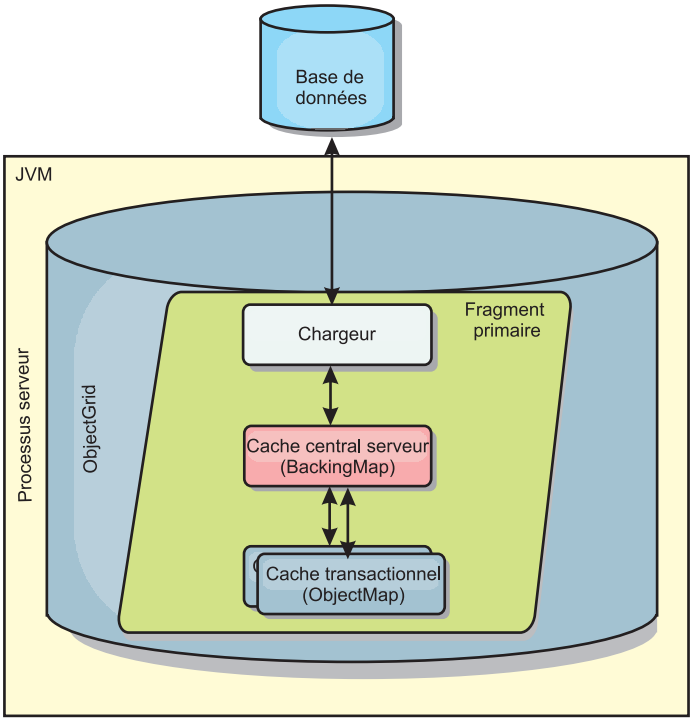


Figure 25. Chargeur

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 au démarrage. 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. En mode synchrone, toutes les transactions client sont bloquées, ce qui empêche tout accès incohérent à la grille. Il est également possible de recourir à un préchargeur client pour charger l'ensemble de la grille.

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, consultez les informations relatives aux chargeurs JPA dans le *Présentation du produit*.

Utilisation d'un 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.

Pour plus d'informations, reportez-vous à la rubrique relative à l'écriture d'un chargeur dans le *Présentation du produit*.

Approche XML de la connexion d'un chargeur

Un chargeur fourni par l'application peut être connecté par le biais d'un fichier XML. L'exemple suivant illustre comment connecter le chargeur MyLoader dans la mappe de sauvegarde map1. 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 recourir à la même structure XML si vous utilisez uniquement un préchargeur en spécifiant le nom de classe du préchargeur et non un nom de classe de chargeur complet.

Configuration du chargeur avec XML

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
    <objectGrid name="grid">
        <backingMap name="map1" 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>
```

Ajout d'un chargeur à l'aide d'un programme

Vous pouvez utiliser une configuration à l'aide d'un programme avec les grilles locales en mémoire. L'extrait de code suivant illustre comment introduire un chargeur fourni par l'application dans la mappe de sauvegarde pour map1, par le biais de l'interface de programme d'application ObjectGrid :

Configuration à l'aide d'un programme du chargeur

```
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 chargeur 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 chargeur 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 chargeur.

- L'utilisateur bob est authentifié en tant qu'utilisateur eXtreme Scale. L'application accède à une grille mygrid à l'aide du nom de l'unité de persistance DB2Hibernate. Le serveur de conteneur s'appelle XS_Server1. Le texte résultant doit ressembler au suivant :
  - **Utilisateur**=bob
  - **Nom de poste de travail**=XS_Server1,192.168.1.101
  - **Nom d'application**=mygrid,DB2Hibernate
  - **Informations comptabilité**=1, DEFAULT,FE7954BD-0126-4000-E000-2298094151DB,com.ibm.db2.jcc.t4.b@71787178
- L'utilisateur bob est authentifié à l'aide d'un jeton WAS. L'application accède à une grille mygrid à l'aide du nom de l'unité de persistance DB2OpenJPA. Le serveur de conteneur s'appelle XS_Server2. Le texte résultant doit ressembler au suivant :
  - **Utilisateur**
    =acme.principal.UserPrincipal[Bob],acme.principal.GroupPrincipal[admin]
  - **Nom de poste de travail**=XS_Server2,192.168.1.102
  - **Nom d'application**=mygrid,DB2OpenJPA
  - **Informations comptabilité**=188,DEFAULT,FE72BC63-0126-4000-E000-851C092A4E33,com.ibm.ws.rsadapter.jdbc.WSJccSQLJConnection@2b432b43

Pour savoir comment surveiller l'accès à la base de données, informez-vous sur DB2 Performance Expert.

Préchargement et préremplissage des données

Dans un grand nombre de scénarios qui intègrent l'utilisation d'un loader, il est possible de préparer la grille en la préchargeant de données.

Une grille, utilisée comme cache complet, doit contenir la totalité des données et elle doit être chargée avant que des clients puissent se connecter à elle. Utilisé comme cache partiel, le cache doit être prérempli avec les données pour permettre aux clients d'accéder immédiatement à ces données dès qu'ils se connectent à la grille.

Il existe deux approches pour le préchargement des données dans la grille : un plug-in Loader ou un loader client. Ces deux approches sont expliquées dans les sections qui suivent.

Plug-in Loader

Le plug-in Loader est associée à chacune des mappes et il est 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, ce qui fait que, si l'on a 100 partitions, l'on aura 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.

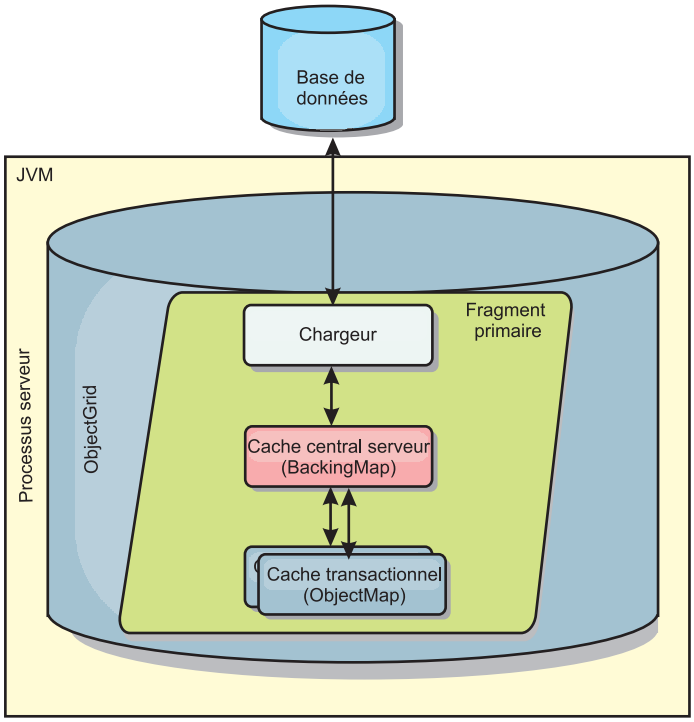


Figure 26. Plug-in Loader

Pour plus d'informations, voir dans le *Guide de programmation* les explications sur l'utilisation d'un 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. Les loaders clients peuvent être invoqués manuellement ou automatiquement au démarrage de la grille. Ils peuvent éventuellement utiliser StateManager pour définir en mode préchargement l'état de la grille, de manière à ce que les clients ne puissent accéder à celle-ci tant que les données sont en cours de préchargement. WebSphere eXtreme Scale comporte un loader JPA (Java Persistence API) utilisable pour charger automatiquement la grille à l'aide d'OpenJPA ou d'Hibernate JPA.

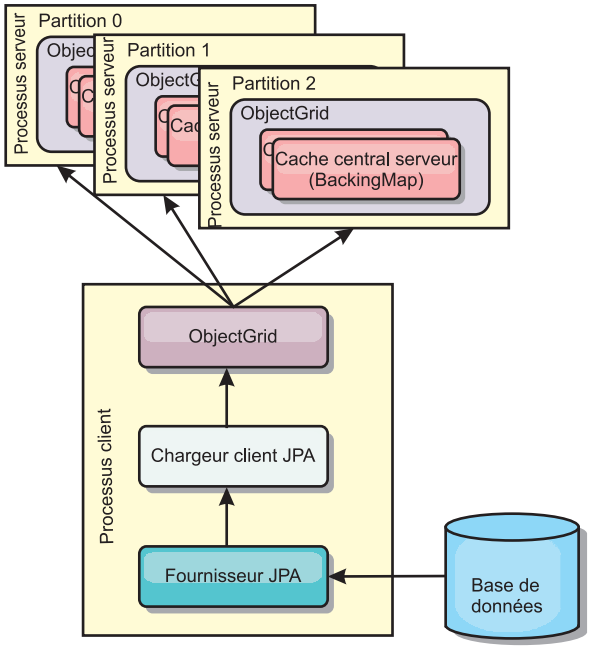


Figure 27. Loader client

Préchargement des mappes

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 répliques. 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.

Vous trouverez dans le *Guide de programmation* les explications sur la manière d'écrire un exemple sur la manière d'implémenter un loader pour un eXtreme Scale répliqué.

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 répliques 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 4. 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 répliques. 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 répliques 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 répliques 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 5. 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
Envoi des modifications vers les fragments répliques 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 répliques 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 6. 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 répliques, 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.

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.

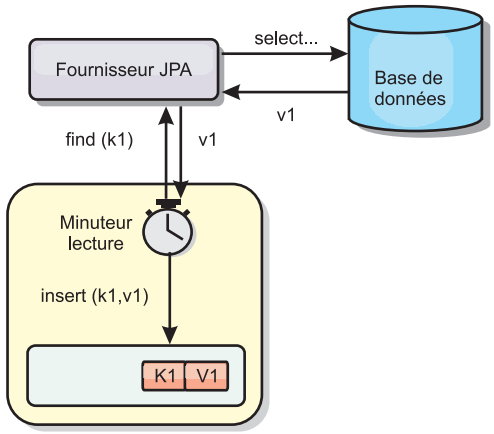


Figure 28. Actualisation régulière

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.

Invalidation des données en cache obsolètes

Pour réduire la durée de visualisation des données obsolètes par les clients, vous pouvez utiliser un mécanisme d'invalidation basé sur les événements ou par 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. eXtreme Scale contient un plug-in JMS ObjectGridEventListener qui informe les clients des éventuelles modifications du 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 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

MapIndexPlugin permet de générer un ou plusieurs index dans une mappe de sauvegarde afin de permettre l'accès aux données dépourvues de clés.

Types d'indexation et configuration d'index

L'indexation est représentée par le plug-in MapIndexPlugin (Index en abrégé). 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.

L'indexation permet de générer un ou plusieurs index dans une mappe de sauvegarde. 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. Cette durée dépendant de la quantité de données indexées, l'interface DynamicIndexCallback est fournie pour permettre aux applications de recevoir des notifications lorsque se produisent certains événements pendant 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'objet proxy de l'index est retourné par l'appel à la méthode getIndex à laquelle on passe le nom du plug-in d'indexation. 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.

Concernant la configuration de HashIndex, voir Configuration de HashIndex.

Pour plus d'informations sur la manière d'écrire son propre plug-in d'indexation, voir dans le *Guide de programmation* les explications concernant l'écriture de plug-in d'indexation.

Pour savoir comment utiliser l'indexation, voir dans le *Guide de programmation* et dans la section HashIndex composite du présent document les explications concernant l'utilisation de l'indexation.

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.

Concepts de mise en cache d'objets Java

WebSphere eXtreme Scale est principalement utilisé comme une grille et un cache de données pour les objets Java. Plusieurs API peuvent permettre d'interagir avec la grilleeXtreme Scale pour accéder et stocker ces objets.

Cette rubrique décrit quelques API courantes et certains concepts utiles pour choisir une API et une topologie de déploiement. Voir la rubrique «Architecture de la mise en cache : mappes, conteneurs, clients et catalogues», à la page 11 pour obtenir une description des divers services et topologies mis à disposition par eXtreme Scale.

Le principal composant de WebSphere eXtreme Scale est l'interface ObjectGrid. Il s'agit de l'espace de noms qui stocke les données liées et contient des ensembles de mappes de hachage, comprenant chacune des paires clé-valeur. Ces mappes peuvent être regroupées et partitionnées ainsi qu'être rendues hautement disponibles et évolutives.

Etant donné que la grille contient par nature des objets Java, il faut réfléchir avec soin à la conception d'une application pour que la grille puisse stocker des données et y accéder avec efficacité. Certains facteurs susceptibles de fragiliser l'évolutivité, les performances et l'utilisation de la mémoire son traités ci-dessous.

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

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

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

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

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

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

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.

L'utilisation de relations dans une grille oblige les données à être organisées en arborescence soumise à contraintes. Il ne doit y avoir qu'un seul type de racine 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 root pour choisir une partition.

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 comme CountryName, par exemple. On a là un cas très particulier où les données doivent exister dans chaque partition. Ici, les données sont accessibles à n'importe quelle clé root et les résultats retournés seront tous identiques. Des données de référence de ce type ne doivent être utilisées que dans les cas où les données sont statiques car leur actualisation pourra s'avérer très onéreuses, devant s'effectuer dans

chacune des partitions. L'API DataGrid est une technique usuellement employée pour conserver à jour les données de référence.

Coûts et avantages de la normalisation

Normaliser les données à l'aide de relations peut contribuer à réduire la quantité de mémoire utilisée par la grille puisqu'il y a moins de duplication des données à opérer. 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. En conséquence de quoi, si vous avez un grand nombre de relations pour une instance d'arborescence, la grille risque de devenir déséquilibrée, avec une partition détenant davantage de données que les autres.

Lorsque les données sont dénormalisées ou mises à plat sans relations hiérarchiques, les données qui, normalement, sont partagées entre deux objets sont au contraire dupliquées et chaque table peut être partitionnée de manière indépendante, ce qui donne une grille beaucoup plus équilibrée. 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. Elle permet aux clients d'utiliser des opérations CRUD (create, read, update et delete) usuelles pour gérer des paires clé/valeur d'objets dans la grille 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, le graphe des relations est reconstruit, comme dans l'exemple qui suit :

```
@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 i est une grille eXtreme Scale grid, 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.

Performances de sérialisation

WebSphere eXtreme Scale utilise plusieurs processus Java pour héberger les données. Ces processus sérialisent les données, c'est-à-dire les convertissent (sous la forme d'instances d'objets Java) en octets, puis si nécessaire de nouveau en objets pour permettre le déplacement des données entre les processus client et serveur. La conversion des paramètres de données est l'opération la plus coûteuse et doit être effectuée par le développeur d'applications lors de la conception du schéma, la configuration de la grille et l'interaction avec les API d'accès aux données.

La sérialisation Java par défaut et les routines de copie sont relativement lentes et peuvent consommer 60 à 70 % de la capacité du processeur dans une configuration classique. Les sections ci-dessous présentent des options d'amélioration des performances de la sérialisation.

Ecriture d'un ObjectTransformer pour chaque mappe de sauvegarde

Un ObjectTransformer peut être associé à une mappe de sauvegarde. Votre application peut comporter une classe qui implémente l'interface ObjectTransformer et offre des implémentations pour les opérations suivantes :

- Copie des valeurs
- Sérialisation et inflation des clés vers et à partir des flux
- Sérialisation et inflation des valeurs vers et à partir des flux

L'application ne doit pas copier des clés car celles-ci sont considérées comme étant non modifiables.

Pour plus d'informations, voir Plug-in de sérialisation et de copie d'objets mis en cache et Pratiques recommandées pour l'interface ObjectTransformer.

Remarque : L'interface ObjectTransformer est uniquement appelée lorsque la grille d'objets connaît les données en cours de transformation. Par exemple, les agents de l'API DataGrid sont utilisés, les agents et les données d'instance des agents ou les données renvoyées par l'agent doivent être optimisées à l'aide de techniques de sérialisation personnalisées. L'interface ObjectTransformer n'est pas appelée pour les agent de l'API DataGrid.

Utilisation d'entités

Lors de l'utilisation de l'API EntityManager avec les entités, la grille ne stocke pas les objets d'entité directement dans les mappes de sauvegarde. L'API EntityManager convertit l'objet d'entité en objets de bloc de données. Pour plus d'informations, voir Pour plus d'informations, consultez la rubrique relative à l'utilisation d'un chargeur avec les mappes d'entité et les blocs de données dans le *Guide de programmation*. Les mappes d'entité sont automatiquement associées à un ObjectTransformer hautement optimisé. Dès que l'API ObjectMap ou EntityManager est utilisée pour interagir avec les mappes d'entité, l'entité ObjectTransformer est appelée.

Sérialisation personnalisée

Dans certains cas, les objets doivent être modifiés de façon à utiliser la sérialisation personnalisée, telle que l'implémentation de l'interface java.io.Externalizable ou l'implémentation des méthodes writeObject et readObject pour les classes qui mettent en oeuvre l'interface java.io.Serializable. Les techniques de sérialisation personnalisée doivent être employées lorsque les objets sont sérialisés à l'aide de mécanismes autres que les méthodes de l'API ObjectGrid ou EntityManager.

Par exemple, lorsque les objets ou entités sont stockés en tant que données d'instance dans un agent d'API DataGrid ou lorsque l'agent renvoie des objets ou des entités, ces objets ne sont transformés à l'aide d'un ObjectTransformer. Toutefois, l'agent fait automatiquement appel à l'ObjectTransformer lors de l'utilisation de l'interface `EntityMixin`. Pour plus de détails, voir Agents DataGrid et mappes basées sur les entités.

Tableaux d'octets

Lors de l'utilisation des API ObjectMap ou DataGrid, les objets de clé et de valeur sont sérialisés dès que le client interagit avec la grille et lorsque les objets sont répliqués. Pour limiter les frais liés à la sérialisation, utilisez des tableaux d'octets et non les objets Java. Le stockage en mémoire des tableaux d'octets revient nettement moins cher car le kit JDK doit rechercher moins d'objets lors de la récupération de place et les tableaux d'octets peuvent être agrandis à la demande. Les tableaux d'octets doivent être uniquement utilisés si vous ne devez pas accéder aux objets utilisant des requêtes ou des index. Les données étant stockées sous la forme d'octets, leur accès n'est autorisé qu'avec la clé correspondante.

WebSphere eXtreme Scale peut automatiquement stocker les données sous la forme de tableaux d'octets à l'aide l'option de configuration de mappes CopyMode.COPY_TO_BYTES ou ce mode de stockage peut être traité manuellement par le client. Cette option permet de stocker les données en mémoire de façon efficace et peut, à la demande, agrandir automatiquement les objets au sein du tableau d'octets pour une utilisation des requêtes et des index.

Voir les meilleures pratiques de la méthode CopyMode dans le *Guide de programmation* pour plus d'informations.

Insertion de données pour différents fuseaux horaires

Lors de l'insertion de données associées à des attributs de calendrier, java.util.Date et d'horodatage dans un objet ObjectGrid, vous devez vous assurer que ces attributs de date et heure sont créés en fonction du même fuseau horaire, surtout en cas de déploiement sur des serveurs correspondant à des fuseaux horaires différents. L'utilisation d'objets de date et heure basés sur le même fuseau horaire permet d'éviter les problèmes liés aux fuseaux horaires divergents. En outre, les données peuvent être interrogées à l'aide de prédicats de calendrier, java.util.Date et d'horodatage.

Si un fuseau horaire n'est pas explicitement indiqué lors de la création d'objets de date et heure, Java utilise le fuseau horaire local, ce qui peut entraîner l'utilisation de valeurs de date et heure incohérentes sur les clients et les serveurs.

Prenons l'exemple d'un déploiement réparti dans lequel client1 correspond au fuseau horaire [GMT-0] et client2 au fuseau horaire [GMT-6]. Ces derniers veulent créer un objet java.util.Date avec la valeur "1999-12-31 06:00:00". client1 crée l'objet java.util.Date avec la valeur "1999-12-31 06:00:00 [GMT-0]" et client2 l'objet java.util.Date avec la valeur "1999-12-31 06:00:00 [GMT-6]". Les deux objets java.util.Date ne correspondent pas car leurs fuseaux horaires sont divergents. Un problème similaire survient lors du pré-chargement de données dans des partitions résidant sur des serveurs correspondants à des fuseaux horaires différents, si le fuseau horaire local est utilisé pour créer les objets de date et heure.

Pour éviter ce problème, l'application peut sélectionner un fuseau horaire de base, tel que [GMT-0], pour la création des objets de calendrier, java.util.Date et d'horodatage.

Pour de plus amples informations, voir la rubrique sur l'interrogation des données situées dans plusieurs fuseaux horaires le *Guide de programmation*

Chapitre 3. Intégration du cache : mise en cache des JPA et des sessions et mise en cache dynamique

L'élément crucial qui donne à WebSphere eXtreme Scale son extrême adaptabilité et fiabilité réside dans sa conception des caches, qui lui permet d'optimiser la persistance et la collecte des données dans quasiment n'importe quel environnement de déploiement.

Loaders JPA

La Java Persistence API (JPA) est une spécification de mappage des objets Java à des bases de données relationnelles. JPA contient une spécification ORM (Object-Relational Mapping) complète utilisant des annotations de métadonnées de langage Java, des descripteurs XML ou les deux pour définir le mappage entre les objets Java et une base de données relationnelle. Un certain nombre d'implémentations commerciales et de code source ouvert sont disponibles.

Vous pouvez utiliser une implémentation de plug-in de chargeur Java Persistence API (JPA) avec eXtreme Scale pour interagir avec les bases de données prises en charge par le chargeur choisi. Pour utiliser JPA, vous devez disposer d'un fournisseur JPA pris en charge, tel qu'OpenJPA ou Hibernate, des fichiers JAR et un fichier `META-INF/persistence.xml` dans votre chemin d'accès aux classes.

Les plug-in JPALoader com.ibm.websphere.objectgrid.jpa.JPALoader et JPAEntityLoader com.ibm.websphere.objectgrid.jpa.JPAEntityLoader sont deux plug-in pré-intégrés de chargeur JPA qui permettent de synchroniser les mappes ObjectGrid avec une base de données. Pour utiliser cette fonction, vous devez disposer d'une implémentation JPA, comme Hibernate ou OpenJPA. La base de données peut correspondre à tout programme d'arrière plan prise en charge par le fournisseur JPA choisi.

Vous pouvez utiliser le plug-in JPALoader lorsque vous stockez des données à l'aide de l'API ObjectMap. Utilisez le plug-in JPAEntityLoader lorsque vous stockez des données à l'aide de l'API EntityManager.

Architecture du chargeur JPA

Le chargeur JPA est utilisé pour les mappes eXtreme Scale qui stockent des objets Java simples (Plain Old Java Objects - POJO).

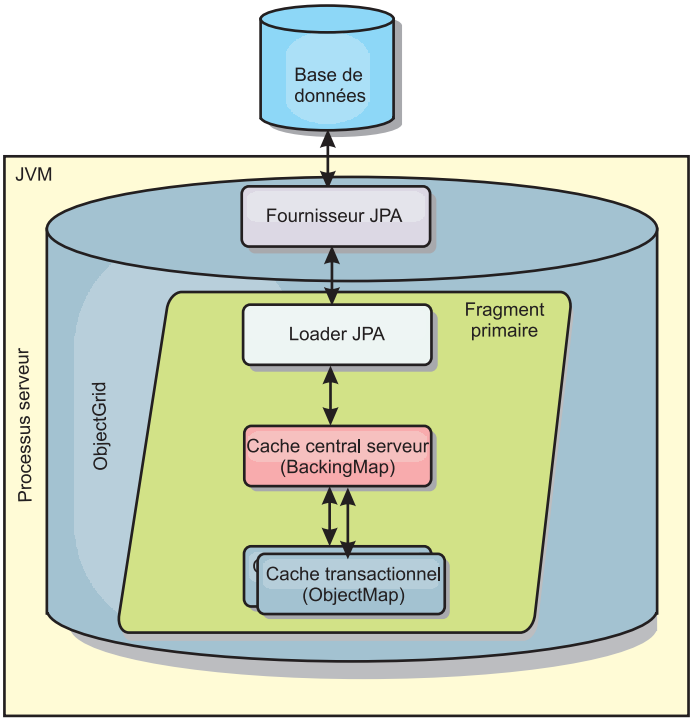


Figure 29. Architecture du chargeur JPA

Lorsqu'une méthode ObjectMap.get(Object key) est appelée, l'exécution eXtreme Scale vérifie tout d'abord si l'entrée est contenue dans la couche de l'ObjectMap. Si ce n'est pas le cas, l'exécution délègue la demande au chargeur JPA. Sur demande de chargement de la clé, JPALoader appelle la méthode JPA EntityManager.find(Object key) pour trouver les données à partir du chargeur JPA. Si les données sont contenues dans le gestionnaire d'entités JPA, elles sont renvoyées ; dans le cas contraire, le fournisseur JPA interagit avec la base de données pour obtenir la valeur.

Lors d'une mise à jour de l'ObjectMap, par exemple à l'aide de la méthode ObjectMap.update(Object key, Object value), l'exécution eXtreme Scale crée un élément LogElement pour cette mise à jour et l'envoie au JPALoader. Le JPALoader appelle la méthode JPA EntityManager.merge(Object value) pour mettre à jour la valeur dans la base de données.

Le plug-in JPAEntityLoader implique les quatre mêmes couches. Toutefois, étant donné que le plug-in JPAEntityLoader est utilisé pour les mappes qui stockent des entités eXtreme Scale, les relations entre les entités peuvent compliquer le scénario d'usage. Une entité eXtreme Scale se distingue d'une entité JPA. Pour plus d'informations, consultez la rubrique relative au plug-in JPAEntityLoader dans le *Guide de programmation*.

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

Pour plus d'informations sur la configuration des chargeurs JPA, consultez les informations relatives à la configuration des chargeurs JPA dans le *Guide d'administration*. Pour plus d'informations sur la programmation des chargeurs JPA, reportez-vous au *Guide de programmation*.

Plug-in de cache JPA

WebSphere eXtreme Scale inclut des plug-in de mémoire cache de niveau 2 pour les fournisseurs OpenJPA et Hibernate Java Persistence API (JPA).

L'utilisation d'eXtreme Scale en tant que fournisseur de cache de niveau 2 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 aux implémentations de cache pré-intégrées car le cache est automatiquement répliqué entre tous les processus. Lorsqu'un client met une valeur en cache, tous les autres clients peuvent utiliser la valeur mise en cache en local.

Avec les plug-in OpenJPA et Hibernate de cache ObjectGrid, vous pouvez créer trois types de topologies : imbriquée, imbriquée et partitionnée, et distante.

Topologie imbriquée

Une topologie imbriquée crée un serveur eXtreme Scale dans l'espace de traitement de chaque application. Les plug-in OpenJPA et Hibernate lisent directement la copie en mémoire du cache et écrivent dans toutes les autres copies. Vous pouvez améliorer les performances d'écriture à l'aide de la réplication asynchrone. Cette topologie par défaut produit un résultat optimal lorsque la quantité de données mises en cache est suffisamment réduite pour être traitée par un seul processus.

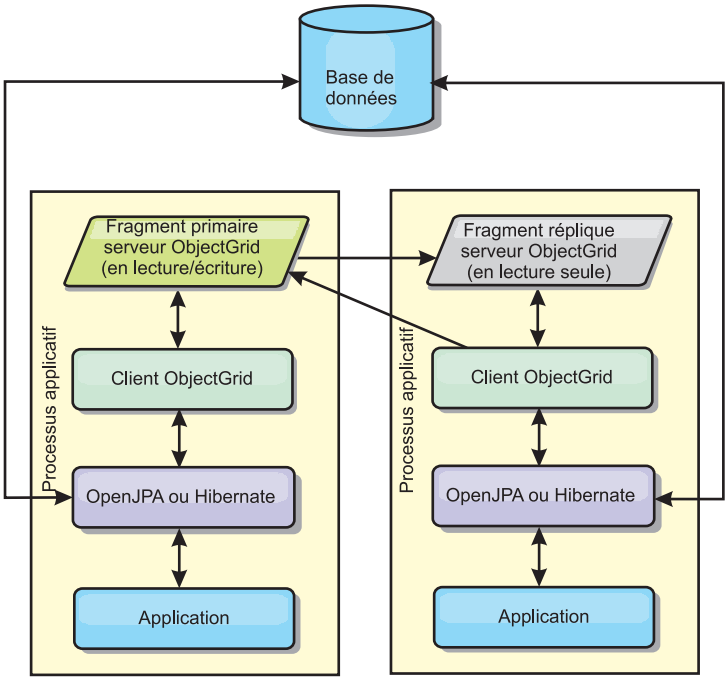


Figure 30. Topologie imbriquée JPA

Avantages :

- Toutes les lectures de cache sont très rapides, en accès local.
- Simple à configurer.

Limitations :

- La quantité de données est limitée à la taille du processus.
- Toutes les mises à jour de cache sont envoyées à un processus.

Topologie imbriquée et partitionnée

Lorsque les données mises en cache sont trop volumineuses pour être comprises dans un seul processus, la topologie imbriquée et partitionnée utilise les partitions ObjectGrid pour diviser les données en plusieurs processus. Les performances obtenues ne sont pas aussi élevées que celles de la topologie imbriquée car la plupart des lectures de cache sont distantes. Toutefois, cette option est utile en cas de temps d'attente élevé de la base de données.

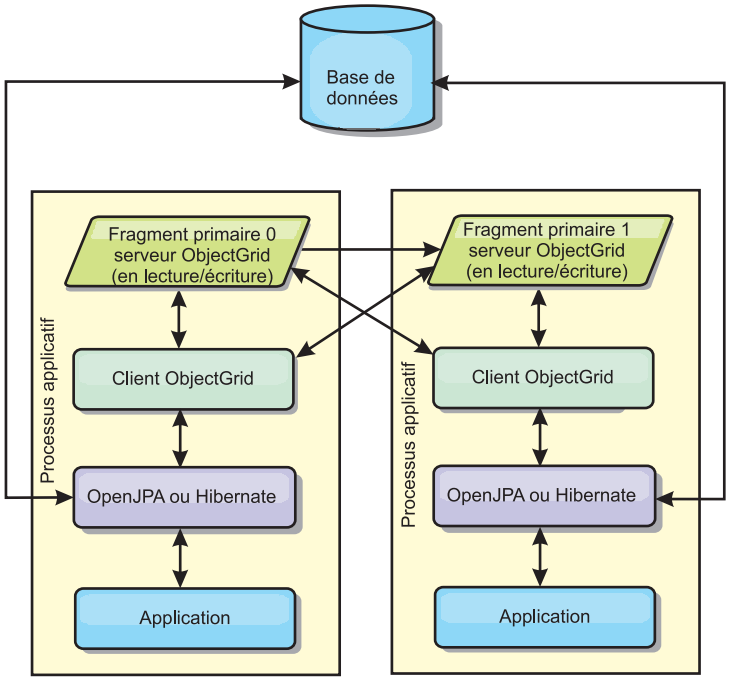


Figure 31. Topologie imbriquée et partitionnée JPA

Avantages :

- Stocke de grandes quantités de données.
- Simple à configurer.
- Les mises à jour de cache sont réparties sur plusieurs processus.

Limitation :

- La plupart des lectures et des mises à jour de cache sont distantes.

Par exemple, pour mettre en cache 10 Go de données avec 1 Go maximum par machine virtuelle Java, dix machines virtuelles Java sont requises. Le nombre de partitions doit par conséquent être défini sur 10 ou plus. De façon idéale, le nombre de partitions doit être défini sur un premier nombre dans lequel chaque fragment stocke une quantité de mémoire raisonnable. Le paramètre numberOfPartitions est généralement égal au nombre de machines virtuelles Java. Chaque machine virtuelle Java stocke une partition à l'aide de ce paramètre. Si vous activez la réplication, vous devez augmenter le nombre de machines virtuelles Java dans le système. Dans le cas contraire, chaque machine virtuelle Java stocke également une réplique de partition qui consomme autant de mémoire que la partition principale.

Consultez la rubrique relative à la définition de la taille de la mémoire et au calcul du nombre de partitions dans le *Guide d'administration* pour optimiser les performances de la configuration choisie.

Par exemple, dans un système comportant 4 machines virtuelles Java et dans lequel la valeur de paramètre numberOfPartitions est égale à 4, chaque machine virtuelle Java héberge une partition principale. Une opération de lecture a 25 pourcents de chances d'extraire des données d'une partition disponible en local, ce qui est sensiblement plus rapide qu'à partir d'une machine virtuelle Java distante. Si une opération de lecture, telle que l'exécution d'une requête, doit extraire une collection de données impliquant une répartition égale de quatre partitions, 75

pourcents des appels sont distants et 25 pourcents sont locaux. Si le paramètre ReplicaMode est défini sur `SYNC` ou `ASYNC` et si le paramètre ReplicaReadEnabled est défini sur `true`, quatre répliques de partitions sont créées et réparties entre quatre machines virtuelles Java. Chaque machine virtuelle Java héberge une partition principale et une réplique. L'opération de lecture a désormais à 50 pourcents de chances de s'exécuter en local. L'opération de lecture qui extrait une collection de données impliquant une répartition égale de quatre partitions comporte 50 pourcents d'appels distants et 50 pourcents d'appels locaux. Les appels locaux sont considérablement plus rapides que les appels distants. Dès que des appels distants sont effectués, les performances chutent.

Topologie distante

Une topologie distante stocke toutes les donnés mises en cache dans un ou plusieurs processus, ce qui réduit la sollicitation de la mémoire par les processus applicatifs. Vous pouvez tirer parti de la distribution de vos données sur des processus distincts en déployant une grille eXtreme Scale partitionnée et répliquée. Contrairement aux configurations imbriquée et imbriquée et partitionnée décrites précédemment, si vous voulez gérer la grille distante, vous devez l'effectuer indépendamment de l'application et du fournisseur JPA. Pour plus d'informations sur la gestion d'un déploiement de grille eXtreme Scale, consultez la rubrique relative à la surveillance de votre environnement de déploiement.

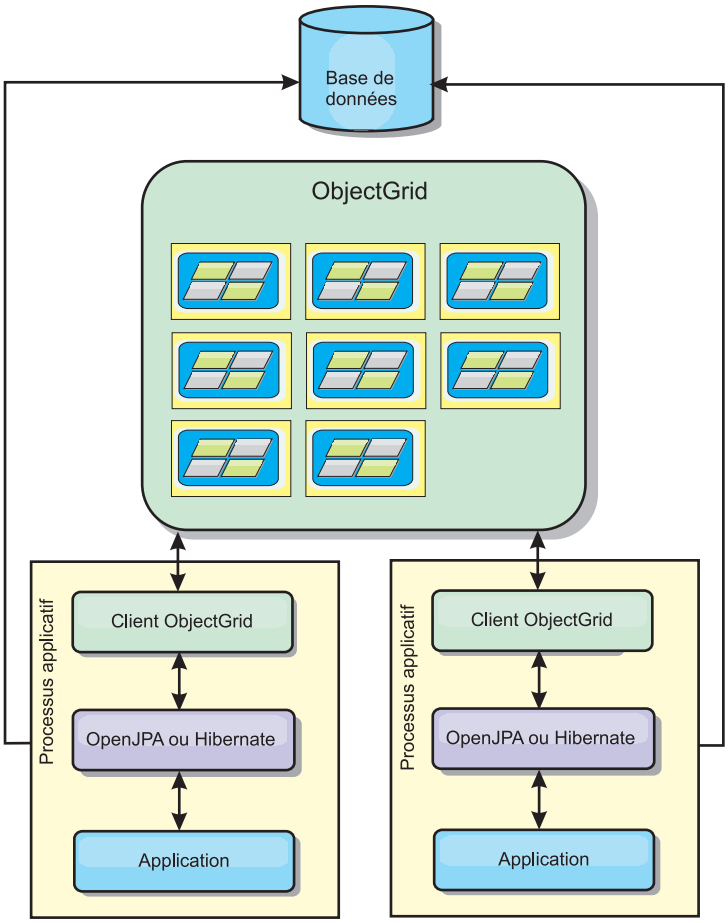


Figure 32. Topologie distante JPA

Avantages :

- Stocke de grandes quantités de données.
- Le processus applicatif est exempt de données en cache.
- Les mises à jour de cache sont réparties sur plusieurs processus.
- Options de configuration extrêmement souples.

Limitation :

- Toutes lectures et mises à jour de cache sont distantes.

Configuration

Pour plus d'informations sur la configuration des plug-in de cache JPA, reportez-vous à la section relative aux plug-in dans le *Guide de programmation*.

Gestion des sessions HTTP

Le gestionnaire de réplication de session livré avec WebSphere eXtreme Scale peut collaborer avec le gestionnaire de sessions par défaut sur le serveur d'applications pour répliquer des données de session d'un processus vers un autre et permettre une haute disponibilité des données de sessions utilisateurs.

Caractéristiques

Le gestionnaire de sessions a été conçu pour être exécuté dans tout conteneur Java Platform, Enterprise Edition version 1.4. Le gestionnaire de sessions ne dépendant en aucune façon des API WebSphere, il peut prendre en charge les diverses versions de WebSphere Application Server ainsi que les environnement de serveurs d'applications du commerce.

Le gestionnaire de sessions HTTP offre des fonctions de réplication de sessions pour une application associée. Le gestionnaire de réplication de session collabore avec celui des conteneurs Web pour créer des sessions HTTP et gérer les cycles de vie des sessions HTTP associées à l'application. La gestion du cycle de vie comprend : l'invalidation des sessions en fonction d'un délai d'attente, d'un servlet explicite ou d'un appel à des JSP (JavaServer Pages). Autre activité de gestion du cycle de vie : l'invocation de programmes d'écoute des sessions associées à la session ou à l'application Web. Le gestionnaire de sessions conserve ses sessions dans une instance ObjectGrid. Cette instance peut être une instance interne locale ou une instance entièrement répliquée, mise en cluster et partitionnée. L'utilisation de cette dernière topologie permet au gestionnaire de sessions de permettre le basculement de sessions HTTP lors de l'arrêt ou de l'interruption inopinée des serveurs d'applications. Le gestionnaire de sessions peut également être exécuté dans des environnements qui ne prennent pas en charge l'affinité lorsque cette dernière n'est pas appliquée par un groupe de serveurs d'équilibrage de charge qui diffusent les demandes au groupe de serveurs d'applications.

Scénarios d'utilisation

Le gestionnaire de sessions est particulièrement utile dans les cas suivants :

- Dans les environnements qui utilisent des serveurs d'applications à différentes versions de WebSphere Application Server, tel que dans un scénario de migration classique.
- Dans les déploiements qui utilisent des serveurs d'application provenant de différentes fournisseurs. Par exemple, application en cours de développement sur des serveurs d'applications source ouverte et hébergée sur WebSphere

Application Server. Une application promue de la phase de transfert à la phase de produit en est un autre exemple. Une migration transparente de ces versions de serveur d'applications est possible alors que toutes les sessions HTTP sont opérationnelles et en service.

- Dans des environnement qui nécessitent que l'utilisateur conserve les sessions avec des niveaux de qualité of service (QoS) plus élevés et une meilleure garantie de disponibilité de session lors de la reprise en ligne du serveur que les niveaux QoS offerts par le serveur par défaut WebSphere Application Server.
- Dans un environnement où l'affinité de session ne peut pas être garantie ou dans des environnements où l'affinité est maintenue par un équilibreur de charge du commerce et où le mécanisme d'affinité doit être personnalisé pour cet équilibreur de charge.
- Dans un environnement pour alléger les frais généraux liés à la gestion des sessions et le stockage vers un processus Java externe.
- Dans plusieurs cellules pour activer le basculement de session entre les cellules.
- Dans plusieurs centres de données ou plusieurs zones.

Fonctionnement du gestionnaire de sessions

Le gestionnaire de réplication de session utilise l'écouteur de session standard pour surveiller les modifications apportées aux données de session et conserve les données de session dans une instance ObjectGrid localement ou à distance. Les données de session sont rechargées dans le chemin d'accès à la demande via le servlet standard depuis l'instance ObjectGrid localement ou à distance. Des outils livrés avec WebSphere eXtreme Scale vous permettent d'ajouter l'écouteur de session et le filtre de servlet à chacun des modules Web de votre application. Vous pouvez également ajouter manuellement ces écouteurs et ces filtres au descripteur de déploiement Web de votre application.

Ce gestionnaire de réplication de session collabore avec chacun des gestionnaires de sessions de conteneurs Web tiers pour pour répliquer les données de session entre les machines virtuelles Java. Lorsque le serveur d'origine expire, les utilisateurs peuvent extraire des données de session d'autres serveurs.

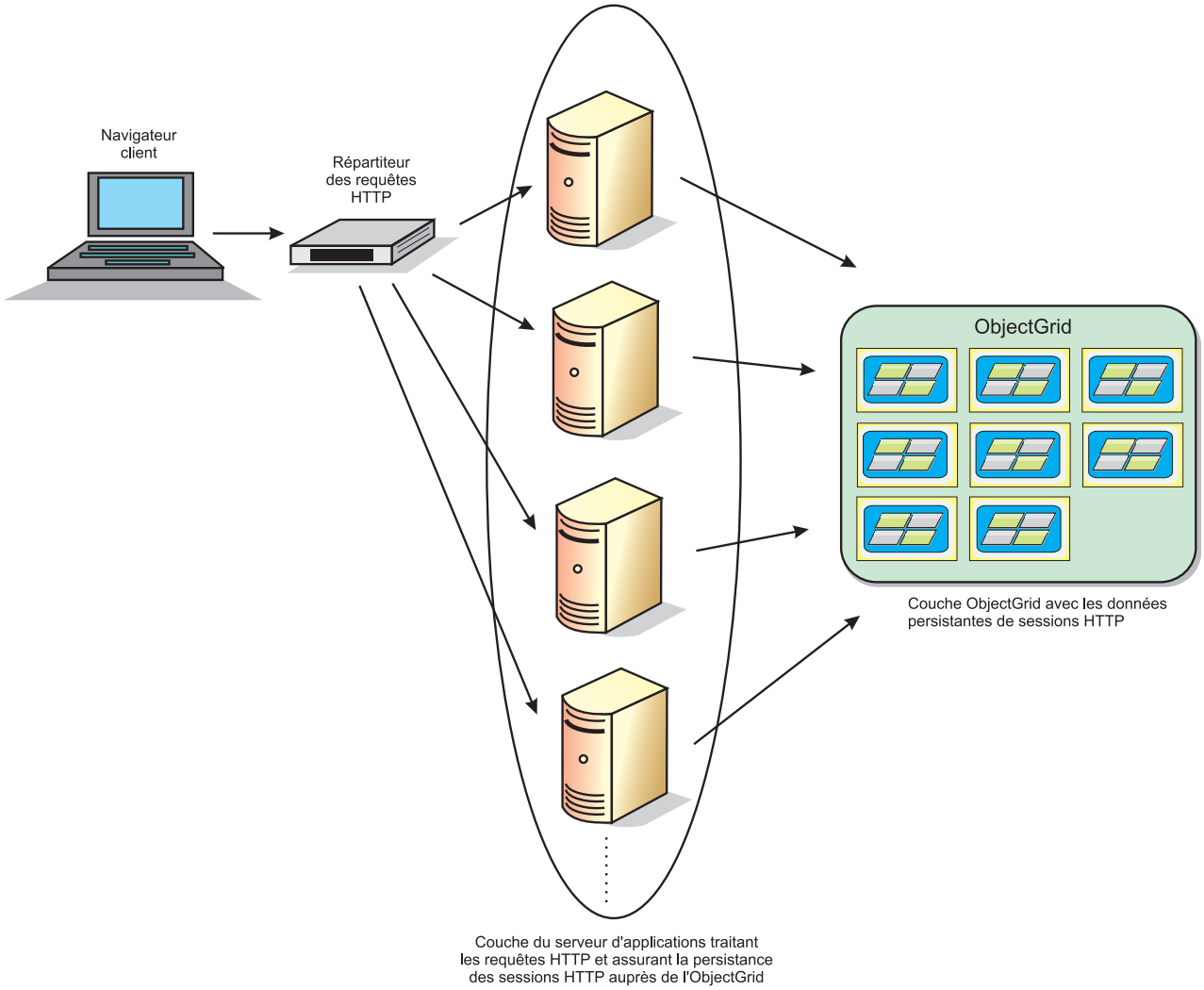


Figure 33. Topologie de gestion de session HTTP avec configuration de conteneur à distance

Topologies de déploiement

Le gestionnaire de sessions peut être configuré à l'aide de deux scénarios de déploiement dynamiques :

- **Conteneurs eXtreme Scale réseau intégrés**

  Dans ce scénario, les serveurs eXtreme Scale sont regroupés dans les mêmes processus que les servlets. Le gestionnaire de sessions peut communiquer directement avec l'instance ObjectGrid locale, pour éviter les retards coûteux du réseau. Ce scénario est préférable dans une exécution avec affinité où les performances sont vitales.
- **Conteneurs eXtreme Scale réseaux distants**

  Dans ce scénario, les serveurs eXtreme Scale exécutent dans des processus externes le processus dans lequel les servlets sont exécutés. Le gestionnaire de sessions communique avec une grille du serveur eXtreme Scale distant. Ce scénario est préférable lorsque le groupe de serveurs conteneurs Web ne dispose pas de la mémoire pour stocker les données de session. Les données de session seront déchargées vers un groupe distinct, ce qui donnera lieu à une moindre

consommation de la mémoire sur le groupe de serveurs conteneurs Web, mais au prix de temps d'attente plus importants en raison de l'emplacement distant des données.

Démarrage du conteneur intégré générique

eXtreme Scale démarre automatiquement un conteneur ObjectGrid intégré dans tout processus de serveur d'applications lorsque le conteneur Web initialise le programme d'écoute de session ou le filtre de servlet, si la propriété objectGridType est définie comme EMBEDDED. Voir Paramètres d'initialisation du contexte de servlet.

Avec la version 7.1 d'eXtreme Scale, vous n'êtes pas obligé de packager un fichier ObjectGrid.xml et un fichier objectGridDeployment.xml dans le fichier WAR ou EAR de votre application Web. Les fichiers ObjectGrid.xml et objectGridDeployment.xml par défaut sont packagés dans le fichier JAR du produit. Des mappes dynamiques sont créées par défaut pour les différents contextes de l'application Web. Les mappes eXtreme Scale statiques n'en sont pas moins toujours prises en charge.

Ce démarrage des conteneurs ObjectGrid intégrés s'applique à tous les types de serveurs d'applications. L'utilisation de composant WebSphere Application Server ou d'un GBean WebSphere Application Server Community Edition GBean est abandonnée.

Gestionnaire de réplication de session basé sur écouteur

Le gestionnaire de réplication de session eXtreme Scale livré avec WebSphere eXtreme Scale peut s'associer au gestionnaire de sessions par défaut du serveur d'applications pour répliquer des données de session d'un processus vers un autre pour une prise en charge de la haute disponibilité des données de session utilisateur.

Le gestionnaire de sessions a été conçu pour être exécuté dans tout conteneur Java™ Platform, Enterprise Edition version 1.4. Le gestionnaire de sessions ne présente aucune dépendance des API WebSphere, et peut donc prendre en charge diverses versions de WebSphere Application Server et des environnement de serveur d'applications de fournisseurs.

Le gestionnaire de sessions HTTP offre des fonctions de réplication de sessions pour une application associée. Le gestionnaire de réplication de session s'associe au gestionnaire de sessions du conteneur Web pour créer des sessions HTTP et gérer les cycles de vie des sessions HTTP associées à l'application. Ces activités de gestion du cycle de vie sont notamment : l'invalidation des sessions en fonction d'un délai d'attente, un servlet explicite ou l'appel aux JavaServer Pages (JSP) et l'invocation de programmes d'écoute de sessions associés à la session ou à l'application Web. Le gestionnaire de sessions conserve ses sessions dans une instance ObjectGrid. Cette instance peut être une instance interne locale ou une instance entièrement répliquée, mise en cluster et partitionnée. L'utilisation de la dernière topologie permet au gestionnaire de sessions de basculer des sessions HTTP lors de l'arrêt ou de l'interruption inopinée des serveurs d'applications. Le gestionnaire de sessions peut également être exécuté dans des environnements qui ne prennent pas en charge l'affinité lorsque cette dernière n'est pas appliquée par un groupe de serveurs d'équilibrage de charge qui diffusent les demandes au groupe de serveurs d'applications.

Scénarios d'utilisation

Le gestionnaire de sessions est particulièrement utile dans les cas suivants :

- Dans les environnements qui utilisent des serveurs d'applications de différentes versions de WebSphere Application Server, tel que dans un scénario de migration classique.
- Dans les déploiements qui utilisent des serveurs d'application provenant de différentes fournisseurs. Par exemple, application développée sur des serveurs d'applications de code source ouvert et hébergée sur WebSphere Application Server. Une application promue de la phase de transfert à la phase de produit en est un autre exemple. Une migration transparente de ces versions de serveur d'applications est possible alors que toutes les sessions HTTP sont opérationnelles et en service.
- Dans des environnement qui nécessitent que l'utilisateur conserve les sessions avec des niveaux de qualité de service (QoS) plus élevés et une meilleure garantie de disponibilité de session lors des basculements de serveurs que les niveaux QoS offerts par le serveur par défaut WebSphere Application Server.
- Dans un environnement où l'affinité de session ne peut pas être garantie ou des environnements où l'affinité est maintenue par un équilibreur de charge de fournisseur et où le mécanisme d'affinité doit être personnalisé en fonction de cet équilibreur de charge.
- Dans un environnement pour alléger les frais généraux liés à la gestion des sessions et le stockage vers un processus Java externe.
- Dans plusieurs cellules pour activer le basculement de session entre les cellules.
- Dans plusieurs centres de données ou plusieurs zones.

Détails du gestionnaire de sessions

Le gestionnaire de réplication de session utilise l'écouteur de session standard pour surveiller les modifications apportées aux données de session et conserve les données de session dans une instance ObjectGrid localement ou à distance. Les données de session sont rechargées dans le chemin d'accès à la demande via le servlet standard depuis l'instance ObjectGrid localement ou à distance. Vous pouvez ajouter l'écouteur de session et le filtre de servlet à chaque module Web de votre application avec des outils livrés avec WebSphere eXtreme Scale. Vous pouvez également ajouter manuellement ces écouteurs et filtres au descripteur de déploiement Web de votre application.

Le gestionnaire de réplication de session s'associe au gestionnaire de sessions de base de chaque fournisseur pour répliquer les données de session d'application. Points à prendre en considération :

- Choisissez des conteneurs ObjectGrid imbriqués ou des conteneurs ObjectGrid distants en fonction de vos besoins en performances et de la taille de vos données. Les conteneurs imbriqués sont plus faciles à configurer et donnent de meilleures performances.
- Choisissez le nom des conteneurs ObjectGrid distants par conteneur Web en fonction de la taille des données des utilisateurs.
- Choisissez la manière de stocker les données de session, la totalité ensemble ou chaque attribut séparément en fonction du nombre et de la taille des données utilisateur d'attributs et de la fréquence de leurs modifications.
- Choisissez l'intervalle de réplication. Des intervalles de réplication moins agressifs donnent de meilleures performances.

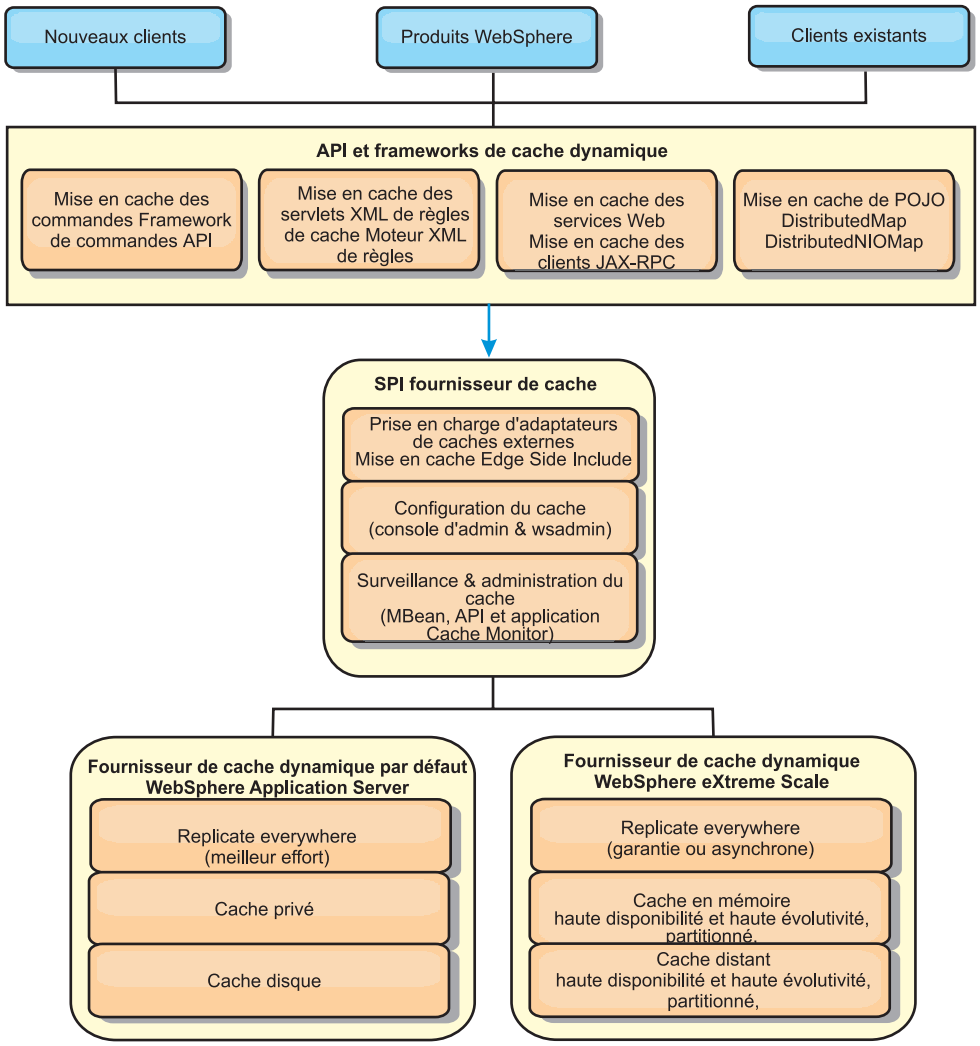
- Choisissez la taille de la table des sessions de manière à équilibrer la taille de la mémoire locale et les performances. Le produit décharge les données utilisateur de sessions lorsque la taille maximale du cache local des sessions est atteinte.
- Le produit prend en charge les sessions HTTP au sein du contexte d'application, conformément à la spécification du servlet, afin d'éviter les problèmes de sécurité et de conflits de noms d'attributs. Vous pouvez partager des sessions entre plusieurs contextes d'application d'après leur classe singleton.
- Le gestionnaire de réplication de session réplique les données de session afin d'assurer la haute disponibilité en écoutant les modifications intervenues dans les données de session et en rechargeant à la demande les données des sessions stockées. Pour ce faire, il réutilise le gestionnaire de sessions de base des conteneurs Web fourni par chaque vendeur. La session est liée ensemble par l'intermédiaire des divers ID natifs de session. Les données de session sont basculées d'un conteneur Web dans un autre grâce au rechargement des données de session à partir de l'instance ObjectGrid. L'ID de session et l'heure de création ne sont pas forcément identiques avant et après le basculement.

Fournisseur de cache dynamique

L'API de cache dynamique est disponible pour les applications Java EE qui sont déployées dans WebSphere Application Server. Le fournisseur de cache dynamique peut être optimisé pour mettre en cache les données métier et les fichiers HTML générés ou pour synchroniser les données en cache de la cellule à l'aide du service de réplication des données.

Présentation

Le seul fournisseur de services auparavant disponible était le moteur de cache dynamique par défaut intégré à WebSphere Application Server. Les clients peuvent utiliser l'interface du fournisseur de cache dynamique de WebSphere Application Server pour connecter eXtreme Scale au cache dynamique. En configurant cette fonction, vous permettez aux applications écrites avec l'API de cache dynamique ou aux applications utilisant la mise en cache au niveau des conteneurs (par exemple les servlets) d'optimiser les fonctions et les performances de WebSphere eXtreme Scale.



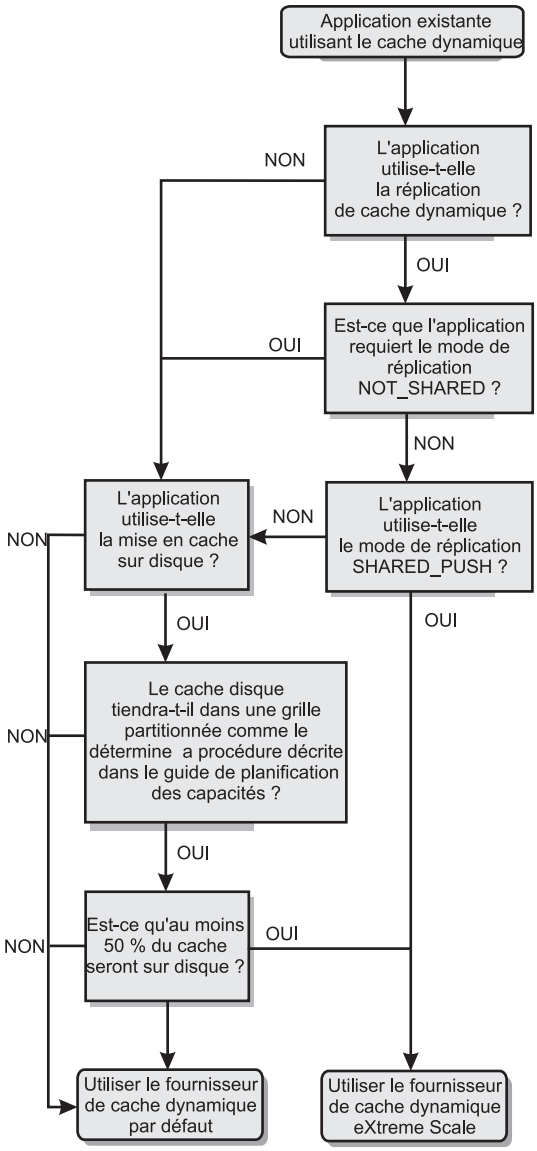
Pour installer et configurer le fournisseur de cache dynamique, reportez-vous aux instructions fournies à la section Configuration du fournisseur de cache dynamique pour WebSphere eXtreme Scale.

Différents modes d'optimisation de WebSphere eXtreme Scale

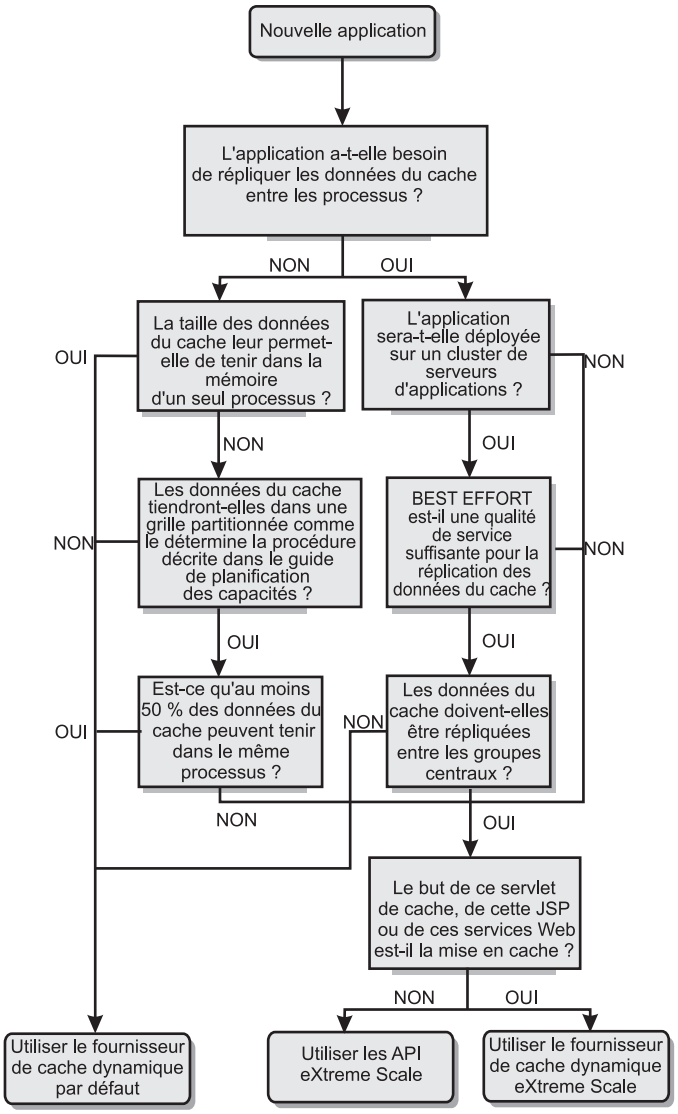
Les différentes fonctions de WebSphere eXtreme Scale permettent d'augmenter de manière significative les fonctions réparties de l'API de cache dynamique au-delà des possibilités offertes par le moteur de cache dynamique et par le service de réplication des données par défaut. Avec eXtreme Scale, vous pouvez créer des mémoires cache véritablement réparties entre plusieurs serveurs et non simplement répliquées et synchronisées d'un serveur à l'autre. Par ailleurs, les mémoires cache eXtreme Scale sont transactionnelles et hautement disponibles : chaque serveur voit ainsi le même contenu pour le service de cache dynamique. WebSphere eXtreme Scale offre une qualité de service en matière de réplication supérieure à celle proposée par DRS.

Tous ces avantages ne signifient cependant pas que le fournisseur de cache dynamique eXtreme Scale constitue la meilleure solution pour toutes les applications. Pour identifier la technologie la mieux adaptée à votre application, utilisez l'arbre de décision et la matrice de comparaison des fonctions.

Arbre de décision permettant de faire migrer des applications existantes de cache dynamique



Arbre de décision permettant de choisir un fournisseur de cache pour les nouvelles applications.



Comparaison des fonctionnalités

Tableau 7. Comparaison des fonctionnalités

Fonctionnalités de cache	Fournisseur par défaut	Fournisseur eXtreme Scale	API eXtreme Scale
Mise en cache local en mémoire	x	x	x
Mise en cache réparti	Imbriqué	Imbriqué, partitionné imbriqué et partitionné distant	Multiple
Evolutivité linéaire		x	x
Réplication fiable (synchrone)		ORB	ORB
Dépassement de capacité des disques	x		

Tableau 7. Comparaison des fonctionnalités (suite)

Fonctionnalités de cache	**Fournisseur par défaut**	**Fournisseur eXtreme Scale**	**API eXtreme Scale**
Expulsion	LRU/TTL/en fonction des segments	LRU/TTL (par partition)	Multiple
Invalidation	x	x	x
Relations	ID de dépendance, modèles	ID de dépendance, modèles	x
Recherches non-clés			Requête et index
Intégration dorsale			Loaders
Transactionnel		Implicite	x
Stockage à base de clés	x	x	x
Evénements et programmes d'écoute	x	x	x
Intégration à WebSphere Application Server	Une seule cellule	Plusieurs cellules	Indépendant des cellules
Prise en charge de Java Standard Edition		x	x
Surveillance et statistiques	x	x	x
Sécurité	x	x	x

Tableau 8. Intégration transparente à la technologie

Fonctionnalités de cache	**Fournisseur par défaut**	Fournisseur **eXtreme Scale**	**API eXtreme Scale**
Mise en cache des résultats servlets/JSP WebSphere Application Server	V5.1+	V6.1.0.25+	
Mise en cache des résultats (JAX-RPC) WebSphere Application Server Web Services	V5.1+	V6.1.0.25+	
Mise en cache des sessions HTTP			x
Fournisseur de cache pour OpenJPA et Hibernate			x
Synchronisation de la base de données à l'aide d'OpenJPA et Hibernate			x

Tableau 9. Interfaces de programmation

Fonctionnalités de cache	Fournisseur par défaut	Fournisseur eXtreme Scale	API eXtreme Scale
API à base de commandes	API de structure des commandes	API de structure des commandes	API DataGrid
API à base de mappes	API DistributedMap	API DistributedMap	API ObjectMap
API EntityManager			x

Pour une explication détaillée du fonctionnement des caches eXtreme Scale répartis, voir dans le *Guide d'administration* les explications sur la configuration des déploiements.

Remarque : Le cache eXtreme Scale réparti peut uniquement stocker des entrées dont la clé et la valeur implémentent toutes les deux l'interface java.io.Serializable.

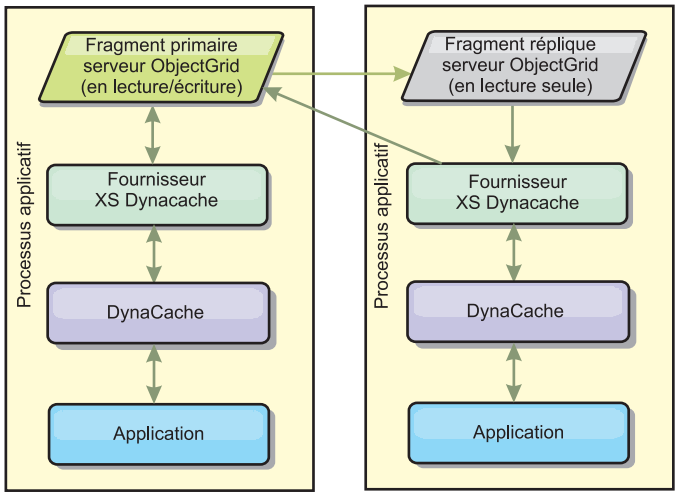
Types de topologie

Un service de cache dynamique créé à l'aide du fournisseur eXtreme Scale peut être déployé dans l'une des trois topologies disponibles. Vous pouvez ainsi personnaliser cette mémoire en fonction de vos besoins en matière de performances, de ressources et d'administration. Ces topologies sont les suivantes : topologie imbriquée, topologie partitionnée imbriquée et topologie distante.

Topologie imbriquée

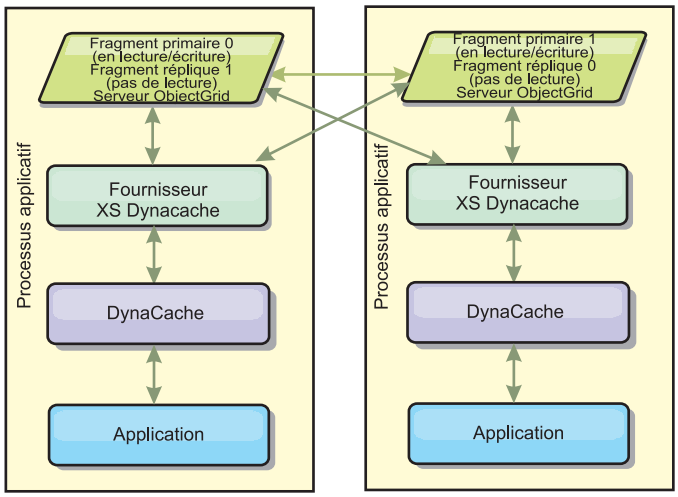
La topologie imbriquée est semblable au cache dynamique et au fournisseur de DRS par défaut. Les instances du cache réparti créées à l'aide de la topologie imbriquée conservent une copie complète de ce cache dans chaque processus eXtreme Scale qui accède au service de cache dynamique, ce qui permet à toutes les opérations de lecture de se faire localement. Toutes les opérations d'écriture passent par un processus serveur unique, au cours duquel les verrous transactionnels sont gérés. Elles sont ensuite répliquées sur les serveurs restants. Cette topologie convient par conséquent mieux aux charges de travail comptant davantage d'opérations de lecture que d'opérations d'écriture.

Avec la topologie imbriquée, les entrées nouvelles ou mises à jour ne sont pas immédiatement visibles sur tous les processus serveur unique. Une entrée de cache n'est pas visible, même pour le serveur l'ayant générée, tant qu'elle ne s'est pas propagée via les services de réplication asynchrones de WebSphere eXtreme Scale. Ces services sont aussi rapides que le matériel le permet, mais il existe toujours un léger décalage. Le graphique suivant présente une topologie imbriquée :



Topologie partitionnée imbriquée

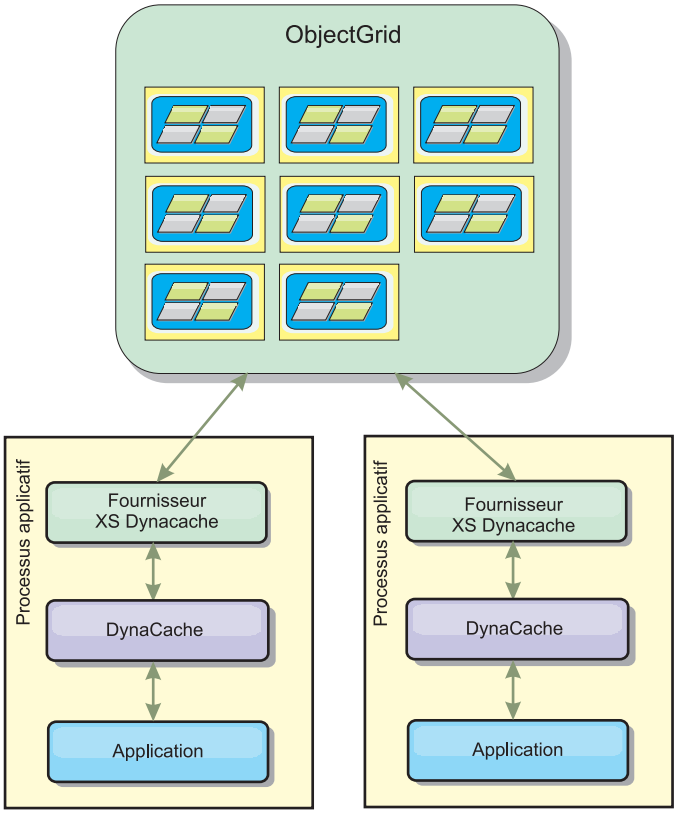
Pour les charges de travail dans lesquelles les opérations d'écriture en cache sont plus fréquentes que les opérations de lecture, une topologie partitionnée imbriquée ou une topologie distante est recommandée. La première conserve toutes les données du cache dans les processus WebSphere Application Server qui accèdent à ce cache. Toutefois, chaque processus ne stocke qu'une partie de ces données. Toutes les opérations de lecture et d'écriture relatives aux données situées sur cette "partition" passent par ce processus, ce qui signifie que la plupart des demandes adressées au cache sont traitées à l'aide d'un appel de procédure distant. Il en découle un temps d'attente plus élevé pour les opérations de lecture qu'avec la topologie imbriquée, mais la capacité du cache réparti à gérer les opérations de lecture et d'écriture évolue de manière linéaire avec le nombre de processus WebSphere Application Server qui accèdent au cache. Par ailleurs, dans cette topologie, la taille maximale du cache n'est pas liée à la taille complète d'un processus WebSphere. Chaque processus en effet ne détenant qu'une partie du cache, la taille maximale de ce dernier est égale à la somme de la taille de tous les processus, moins la taille du processus lui-même. Le graphique suivant représente une topologie partitionnée imbriquée :



Supposons par exemple que vous disposiez d'une grille de processus serveur dont chacun est associé à 256 mégaoctets de segments de mémoire disponibles pour héberger un service de cache dynamique. S'ils utilisaient une topologie imbriquée, le fournisseur de cache dynamique par défaut et le fournisseur eXtreme Scale seraient limités à une taille de cache interne égale à 256 mégaoctets moins la mémoire utilisée par eux-mêmes. Reportez-vous à la section relative à la planification de la capacité et à la haute disponibilité du présent manuel. S'il utilisait une topologie partitionnée imbriquée, le fournisseur eXtreme Scale serait limité à une taille de cache égale à un gigaoctet moins la mémoire utilisée par lui-même. Le fournisseur WebSphere eXtreme Scale permet ainsi de disposer d'un cache dynamique en mémoire plus grand qu'un simple processus serveur. Le fournisseur de cache dynamique par défaut utilise le cache-disque pour permettre aux instances du cache de croître au-delà de la taille d'un simple processus. Dans de nombreux cas, le fournisseur WebSphere eXtreme Scale rend superflue l'utilisation du cache-disque et des systèmes coûteux de stockage sur disque pour exécuter ces instances.

Topologie distante

La topologie distante rend également superflue l'utilisation du cache-disque. La seule différence entre la topologie distante et la topologie partitionnée imbriquée est qu'avec la première, toutes les données du cache sont stockées hors des processus WebSphere Application Server. WebSphere eXtreme Scale prend en charge les processus conteneur autonomes pour les données en cache. Ces processus utilisent moins de mémoire qu'un processus WebSphere Application Server et ils ne sont pas limités à l'utilisation d'une machine virtuelle Java donnée. Les données associées à un service de cache dynamique auquel un processus WebSphere Application Server 32 bits accède peuvent par exemple se trouver sur un processus conteneur s'exécutant sur une machine virtuelle Java eXtreme Scale. Les utilisateurs peuvent ainsi exploiter la capacité mémoire accrue des processus 64 bits pour la mise en cache, sans supporter la mémoire supplémentaire nécessaire aux processus serveur de l'application. Le graphique suivant représente une topologie distante :



Compression des données

La compression est une autre fonction du fournisseur de cache dynamique WebSphere eXtreme Scale pouvant aider les utilisateurs à gérer l'excédent de mémoire généré par le cache. Le fournisseur de cache dynamique ne permet pas la compression en mémoire des données du cache. Avec le fournisseur eXtreme Scale, cette opération est possible. La compression du cache à l'aide de l'algorithme deflate peut être activée dans les trois topologies. Elle génère une charge supplémentaire pour les opérations de lecture et d'écriture, mais permet d'améliorer considérablement la densité de la mémoire cache pour les applications telles que les applications de mise en cache des servlets et de JSP.

Cache interne locale

Le fournisseur de cache dynamique de WebSphere eXtreme Scale est utilisable également pour les instances de cache dynamique dans lesquelles la **réplication est désactivée**. Tout comme avec le fournisseur par défaut, ces caches peuvent stocker des données non sérialisables. Ils offrent également de meilleures performances sur les serveurs d'entreprise contenant multiprocesseurs car le codepath eXtreme Scale est conçu pour optimiser les accès simultanés en mémoire cache interne.

Moteur de cache dynamique et différences fonctionnelles avec eXtreme Scale

Lorsque la réplication est désactivée pour les caches locaux en mémoire, il n'existe aucune différence fonctionnelle réelle entre les caches du fournisseur de cache dynamique par défaut et WebSphere eXtreme Scale. Les utilisateurs ne constatent

aucune différence, sinon que les caches WebSphere eXtreme Scale ne supportent pas le déchargement sur disque ni les statistiques ou opérations en rapport avec la taille du cache en mémoire.

Lorsque la réplication est activée, aucune différence notable ne pourra être observée dans les résultats renvoyés par la plupart des appels de l'API de cache dynamique lorsque le client utilise le fournisseur de cache dynamique ou lorsqu'il utilise le fournisseur eXtreme Scale. Pour certaines opérations, il est impossible d'émuler le comportement du moteur de cache dynamique à l'aide d'eXtreme Scale.

Statistiques du cache dynamique

Les statistiques de la mémoire cache dynamique sont générées via l'application CacheMonitor ou via le bean géré de cache dynamique. Lorsque le fournisseur de cache dynamique eXtreme Scale est utilisé, les statistiques sont générées via ces interfaces, mais le contexte des valeurs statistiques est différent.

Si une instance de cache dynamique est partagée entre trois serveurs nommés A, B et C, l'objet des statistiques renvoie des statistiques uniquement pour la copie du cache se trouvant sur le serveur à partir duquel l'appel a été lancé. Si les statistiques sont extraites à partir du serveur A, elles reflètent uniquement l'activité du serveur A.

Avec eXtreme Scale, il n'existe qu'un seul cache réparti partagé entre tous les serveurs : il est donc impossible d'effectuer un suivi de la plupart des statistiques serveur par serveur, contrairement à ce qui est le cas avec le fournisseur de cache dynamique par défaut. Vous trouverez ci-dessous une liste des statistiques générées par l'API Cache Statistics, ainsi que les explications correspondantes lorsque vous utilisez le fournisseur WebSphere eXtreme Scale de cache dynamique. Tout comme le fournisseur par défaut, ces statistiques ne sont pas synchronisées. Elles peuvent donc varier jusqu'à 10 % pour des charges de travail simultanées.

- **Réussites en cache** : les réussites en cache font l'objet d'un suivi par serveur. Si le trafic enregistré sur le serveur A génère 10 réussites en cache et que le trafic enregistré sur le serveur B génère 20 réussites en cache, cette statistique fait état de 10 réussites en cache sur le serveur A et de 20 réussites en cache sur le serveur B.
- **Echecs en cache** : comme les réussites, les échecs en cache font l'objet d'un suivi par serveur.
- **Entrées en cache** : cette statistique renvoie le nombre d'entrées présentes dans le cache réparti. Chaque serveur qui accède au cache renvoie la même valeur, qui correspond au nombre total d'entrées en cache en mémoire pour tous les serveurs.
- **Taille du cache en Mo** : cette mesure n'est actuellement prise en charge que pour les caches utilisant les topologies distantes, imbriquées ou partitionnées imbriquées. Elle indique combien de mégaoctets du segment de mémoire Java sont utilisés par le cache dans l'ensemble de la grille. Cette statistique ne porte que sur l'utilisation des segments de mémoire par les partitions primaires. C'est à vous de prendre en compte les fragments répliques. Le paramètre par défaut des topologies distantes et partitionnées imbriquées étant le fragment réplique asynchrone, il convient en effet de doubler ce chiffre pour connaître l'utilisation effective de la mémoire par le cache.
- **Suppressions dans le cache** : nombre total d'entrées supprimées du cache par quelque méthode que ce soit. Agrégat des suppressions de l'ensemble du cache

réparti. Si le trafic enregistré sur le serveur A génère 10 invalidations et que le trafic enregistré sur le serveur B génère 20 invalidations, la valeur correspondant à la somme des deux serveurs est 30.

- **Suppression LRU (Least Recently Used) du cache** : tout comme les suppressions dans le cache, il s'agit d'un agrégat. Il représente le nombre d'entrées ayant été supprimées pour maintenir la taille du cache en deçà de sa taille maximale.
- **Invalidation pour dépassement du délai d'attente** : agrégat du nombre des entrées qui ont été supprimées en raison d'un dépassement du délai d'attente.
- **Invalidations explicites** : agrégat du nombre des entrées qui ont été supprimées du fait d'une invalidation directe par clé, ID dépendance ou modèle.
- **Statistiques étendues** : le fournisseur de cache dynamique eXtreme Scale exporte les chaînes de clé de statistiques étendues suivantes.
  - **com.ibm.websphere.xs.dynacache.remote_hits** : nombre total de réussites en mémoire faisant l'objet d'un suivi dans le conteneur eXtreme Scale. Il s'agit d'une statistique agrégée. Sa valeur dans la mappe des statistiques étendues est une valeur `longue`.
  - **com.ibm.websphere.xs.dynacache.remote_misses** : nombre total d'échecs en mémoire faisant l'objet d'un suivi dans le conteneur eXtreme Scale. Il s'agit d'une statistique agrégée. Sa valeur dans la mappe des statistiques étendues est une valeur `de type long`.

Rapport sur la réinitialisation des statistiques

Le fournisseur de cache dynamique vous permet de réinitialiser les statistiques du cache. Avec le fournisseur par défaut, l'opération de réinitialisation efface uniquement les statistiques du serveur concerné. Le fournisseur de cache dynamique eXtreme Scale suit la plupart de ses données statistiques sur les conteneurs de cache distant. Ces données ne sont ni effacées ni modifiées lors de la réinitialisation des statistiques. Au contraire, le comportement du cache dynamique par défaut est simulé sur le client en signalant la différence entre la valeur actuelle d'une statistique donnée et sa valeur au moment du dernier appel de réinitialisation de ce serveur.

Si, par exemple, le trafic enregistré sur le serveur A génère 10 suppressions dans le cache, les statistiques concernant le serveur A et le serveur B indiquent 10 suppressions. Si les statistiques du serveur B sont réinitialisées et que 10 suppressions supplémentaires sont enregistrées sur le serveur A, les statistiques du serveur A indiqueront 20 suppressions et celles du serveur B indiqueront 10 suppressions.

Evénements du cache dynamique

L'API de cache dynamique permet aux utilisateurs d'enregistrer des programmes d'écoute d'événements. Lorsque vous utilisez eXtreme Scale en tant que fournisseur de cache, ces programmes fonctionnent comme prévu pour les mémoires cache internes.

Pour les mémoires cache réparties, le comportement des événements dépend de la topologie utilisée. Si la topologie est imbriquée, les événements sont générés sur le serveur qui gère les opérations d'écriture, également appelé fragment primaire. Ceci signifie qu'un seul serveur reçoit les notifications d'événements, mais qu'il recevra toutes les notifications normalement attendues par le fournisseur de cache

dynamique. Comme WebSphere eXtreme Scale choisit le fragment primaire au moment de l'exécution, il est impossible de s'assurer qu'un processus serveur donné reçoit toujours ces événements.

Les mémoires cache partitionnées imbriquées génèrent des événements sur les serveurs hébergeant une partition du cache. Ainsi, si un cache contient 11 partitions et que chaque serveur d'une grille WebSphere Network Deployment constituée de 11 serveurs héberge l'une des partitions, chaque serveur reçoit les événements du cache dynamique relatifs aux entrées de cache qu'il héberge. Aucun processus serveur ne peut voir à lui tout seul tous les événements, à moins que les 11 partitions ne soient hébergées dans ce processus serveur. Comme dans le cas de la topologie imbriquée, il est impossible de s'assurer qu'un processus serveur donné va recevoir un ensemble donné d'événements ou aucun événement.

Les mémoires cache utilisant une topologie distante ne prennent pas en charge les événements du cache dynamique.

Appels des beans gérés

Le fournisseur de cache dynamique WebSphere eXtreme Scale ne prend pas en charge la mise en cache sur un disque. Les appels de beans gérés relatifs à une mise en cache sur un disque ne fonctionnent pas.

Mappage des règles de réplication du cache dynamique

Le fournisseur de cache dynamique pré-intégré de WebSphere Application Server prend en charge plusieurs règles de réplication du cache. Vous pouvez configurer ces règles de manière globale ou pour chaque entrée du cache. Pour obtenir une description de ces règles, consultez la documentation relative au cache dynamique.

Le fournisseur de cache dynamique eXtreme Scale n'honore pas directement ces règles. Les caractéristiques de réplication du cache sont déterminées par le type de topologie répartie eXtreme Scale configuré. Elles s'appliquent à toutes les valeurs placées dans cette mémoire, quelles que soient les règles de réplication définies par le service de cache dynamique pour l'entrée. Vous trouverez ci-dessous une liste de toutes les règles de réplication prises en charge par le service de cache dynamique et une description des topologies eXtreme Scale offrant des caractéristiques de réplication semblables.

Remarquez que le fournisseur de cache dynamique eXtreme Scale ignore les règles de réplication DRS sur un cache ou sur une entrée de cache. Les utilisateurs doivent choisir la topologie la mieux adaptée à leurs besoins en réplication.

- NOT_SHARED – aucune des topologies actuellement proposées par le fournisseur de cache dynamique eXtreme Scale ne se rapproche de ces règles. Cela signifie que toutes les données stockées en cache doivent être associées à des clés et à des valeurs qui implémentent java.io.Serializable.
- SHARED_PUSH : la topologie imbriquée se rapproche de ces règles de réplication. Lorsqu'une entrée de cache est créée, elle est répliquée vers l'ensemble des serveurs. Ceux-ci recherchent les entrées de cache localement uniquement. Si une entrée n'est trouvée localement, elle est supposée ne pas exister et les autres serveurs ne sont pas interrogés à son sujet.
- SHARED_PULL et SHARED_PUSH_PULL : les topologies partitionnées imbriquées et les topologies distantes se rapprochent de ces règles de réplication. L'état réparti du cache est parfaitement cohérent sur tous les serveurs.

Ces informations sont destinées à vous aider à vérifier que la topologie choisie correspond à vos besoins en matière de cohérence répartie. Si, par exemple, la topologie imbriquée correspond parfaitement à vos besoins en matière de déploiement et de performances, mais que vous souhaitiez bénéficier du niveau de cohérence du cache apporté par SHARED_PUSH_PULL, vous pouvez envisager d'utiliser la topologie partitionnée imbriquée en dépit de ses performances légèrement inférieures.

Sécurité

Vous pouvez sécuriser les instances de cache dynamique qui s'exécutent dans des topologies imbriquées ou dans des topologies partitionnées imbriquées grâce à la fonction de sécurité intégrée à WebSphere Application Server. Consultez la documentation relative à Sécurisation des applications dans leur environnement dans le WebSphere Application Servercentre de documentation.

Lorsqu'un cache s'exécute dans une topologie distante, un client eXtreme Scale autonome peut se connecter au cache et affecter le contenu de l'instance de cache dynamique. Le fournisseur de cache dynamique eXtreme Scale est doté d'une fonction légère de chiffrement pouvant empêcher la lecture ou la modification des données du cache par des clients autres que les clients WebSphere Application Server. Pour l'activer, associez le paramètre facultatif **com.ibm.websphere.xs.dynacache.encryption_password** à la même valeur pour chaque instance WebSphere Application Server qui accède au fournisseur de cache dynamique. La valeur et les métadonnées utilisateur de l'entrée de cache sont ainsi chiffrés à l'aide de la fonction de chiffrement AES 128 bits. Il est très important de définir la même valeur sur tous les serveurs, car ils ne parviendront pas à lire les données mises en cache par d'autres serveurs avec une valeur différente.

Si le fournisseur eXtreme Scale détecte différentes valeurs pour cette variable dans le même cache, il génère un avertissement dans le journal du processus conteneur eXtreme Scale.

Voir la documentation eXtreme Scale sur WebSphere eXtreme Scale la sécurité si l'authentification SSL ou client est requise.

Informations supplémentaires

- Redbook relatif au cache dynamique
- Documentation relative au cache dynamique
  - WebSphere Application Server 7.0
  - WebSphere Application Server 6.1
- Documentation relative à DRS
  - WebSphere Application Server 7.0
  - WebSphere Application Server 6.1

Planification de la capacité et haute disponibilité (mise en cache dynamique)

L'API de cache dynamique est disponible pour les applications Java EE qui sont déployées dans WebSphere Application Server. Ce cache dynamique peut être optimisé pour mettre en cache les données métier et les fichiers HTML générés ou pour synchroniser les données en cache de la cellule à l'aide du service DRS de réplication des données.

Présentation

Toutes les instances du cache dynamique créées avec le fournisseur de cache dynamique de WebSphere eXtreme Scale sont par défaut à disponibilité élevée. Le niveau et le coût de la mémoire de la haute disponibilité dépendent de la topologie utilisée.

Si vous utilisez une topologie imbriquée, la taille du cache est limitée à la quantité de mémoire disponible dans un processus serveur unique et chaque processus serveur stocke une copie complète du cache. Tant que l'exécution de ce processus serveur unique se poursuit, le cache survit. Les données de cette mémoire sont perdues uniquement en cas d'arrêt de tous les serveurs qui y accèdent.

Dans le cas d'une topologie partitionnée imbriquée, la taille du cache est limitée à l'ensemble de l'espace disponible dans tous les processus serveur. Le fournisseur de cache dynamique eXtreme Scale par défaut utilise une réplique pour chaque fragment primaire, de sorte que chaque donnée mise en cache est stockée deux fois.

Utilisez la formule A pour déterminer la capacité du cache partitionné imbriqué.

Formule A

D * C / (1 + S) = M

Où :

- D = Mémoire disponible par processus conteneur
- C = nombre de conteneurs
- S = nombre de fragments répliques
- M = taille totale du cache

Prenons le cas d'une grille WebSphere Network Deployment dans laquelle chaque processus dispose de 256 Mo d'espace disponible et qui compte un total de 4 processus serveur : une instance du cache de tous ces serveurs peut stocker jusqu'à 512 mégaoctets de données. Dans ce mode, le cache peut survivre à une panne de serveur sans perte de données. De la même manière, deux serveurs au maximum peuvent être arrêtés séquentiellement sans perte de données. Pour l'exemple suivant, la formule est la suivante :

256 Mo * 4 conteneurs/ (1 primaire + 1 réplique) = 512 Mo.

Les caractéristiques de taille des mémoires cache utilisant une topologie distante sont similaires à celles qui utilisent une topologie partitionnée imbriquée, mais les premières sont limitées à la quantité d'espace disponible dans tous les processus conteneur eXtreme Scale.

Dans une topologie distante, il est possible d'augmenter le nombre de fragments répliques pour atteindre un niveau de disponibilité plus élevé, au prix d'une augmentation de la quantité de mémoire. Dans la plupart des applications dont le cache est dynamique, cette opération n'est pas nécessaire, mais vous pouvez éditer le fichier dynacache-remote-deployment.xml pour augmenter le nombre de fragments répliques.

Utilisez les formules B et C suivantes pour déterminer l'effet de l'ajout de fragments répliques sur la haute disponibilité du cache.

Formule B

N = Minimum(T -1, S)

Où :

- N = nombre d'échecs simultanés de processus
- T = nombre total de conteneurs
- S = nombre total de fragments répliques

Formule C

Ceiling(T/ (1+N)) = m

Où :

- T = nombre total de conteneurs
- N = nombre total de fragments répliques
- m = nombre minimal de conteneurs nécessaires pour prendre en charge les données en cache.

Sur l'optimisation des performances avec le fournisseur de cache dynamique, voir Optimisation du fournisseur de cache dynamique.

Définition de la taille du cache

Avant de pouvoir déployer une application utilisant le fournisseur de cache dynamique WebSphere eXtreme Scale, vous devez combiner les principes généraux décrits à la section précédente et les données environnementales des systèmes de production. Les premiers chiffres à identifier sont le nombre total de processus conteneur et la quantité de mémoire disponible dans chaque processus pouvant contenir les données du cache. Dans une topologie imbriquée, les conteneurs du cache sont aussi localisés dans les processus serveur WebSphere Application, de sorte qu'il existe un conteneur par serveur partageant le cache. Le meilleur moyen de définir l'espace disponible dans le processus est d'identifier la quantité de mémoire supplémentaire de l'application sans activer la mise en cache et WebSphere Application Server. Pour cela, vous pouvez analyser les données détaillées de récupération de place. Lorsque la topologie utilisée est distante, vous trouvez ces informations dans la sortie détaillée de la récupération de place d'un conteneur autonome ayant démarré récemment et dans lequel aucune donnée de cache n'a encore été entrée. Dernier élément auquel vous devez penser : vous devez réserver des segments de la mémoire pour la récupération de place. La somme de l'espace restant dans le conteneur WebSphere Application Server ou dans le conteneur autonome et de la taille réservée pour le cache ne doit pas dépasser 70 % du total des segments de mémoire.

Une fois ces informations collectées, les valeurs peuvent être entrées dans la formule A décrite précédemment pour déterminer la taille maximale du cache partitionné. Une fois cette taille connue, l'étape suivante consiste à déterminer le nombre total d'entrées du cache pouvant être prises en charge, ce qui suppose d'identifier la taille moyenne de chaque entrée. Il suffit pour cela d'ajouter 10% à la taille de l'objet client. Pour obtenir des informations plus détaillées sur la définition de la taille des entrées du cache lors d'une utilisation du cache dynamique, voir le Guide d'optimisation du cache dynamique et du service de réplication des données.

Lorsque la compression est activée, elle affecte la taille de l'objet client et non l'espace restant dans le système de mise en cache. Utilisez la formule suivante pour déterminer la taille d'un objet mis en cache lorsque la compression est utilisée :

T = O * C + O * 0.10

Où :

- T = Taille moyenne de l'objet mis en cache
- O = Taille moyenne de l'objet client non compressé
- C = Rapport de compression exprimé sous la forme d'une fraction.

Un rapport de compression de 2 à 1 est égal à 1/2 = 0.50. Une valeur moins élevée est recommandée. Si l'objet stocké est un objet Java simple normal principalement rempli de types primitifs, vous pouvez supposer que le rapport de compression est de l'ordre de 0,60 à 0,70. Si l'objet mis en cache est un objet servlet, JSP ou WebServices, la meilleure méthode pour déterminer le rapport de compression consiste à compresser un exemple représentatif avec un utilitaire de compression ZIP. Si cette opération est impossible, considérez qu'un rapport de compression compris entre 0,2 et 0,35 est fréquent pour ce type de données.

Ensuite, utilisez ces informations pour déterminer le nombre total d'entrées du cache qui peuvent être prises en charge. Utilisez la formule D suivante :

Formule D

T = S / M

Où :

- T = Nombre total d'entrées du cache
- S = Taille totale disponible pour les données du cache, calculée à l'aide de la formule A
- M = Taille moyenne de chaque entrée du cache

Enfin, vous devez définir la taille de l'instance de cache dynamique pour appliquer cette limite. A cet égard, le fournisseur de cache dynamique WebSphere eXtreme Scale est différent du fournisseur de cache dynamique par défaut. Utilisez la formule suivante pour déterminer la valeur de la taille de l'instance de cache dynamique. Cette formule est la suivante :

Formule E

Ct = Tt / Np

Où :

- Tt = Taille cible totale du cache
- Ct = Paramètre de la taille du cache à définir dans l'instance de cache dynamique
- Np = Nombre de partitions. La valeur par défaut est 47.

Associez la taille de l'instance de cache dynamique à une valeur calculée par la formule E pour chaque serveur partageant l'instance du cache.

Chapitre 4. Extensibilité

L'extensibilité est ce qui permet aux données d'un déploiement de WebSphere eXtreme Scale d'être réparties dans un ensemble de serveurs (conteneurs) en fonction des choix de configurations que vous avez opérés.

Evolutivité

WebSphere eXtreme Scale est évolutif grâce aux données partitionnées. Il peut s'adapter à des milliers de conteneurs si besoin est, car les conteneurs sont indépendants les uns des autres.

WebSphere eXtreme Scale divise les ensembles de données en partitions distinctes pouvant être déplacées d'un processus à un autre, voire même d'une machine à une autre au cours de l'exécution. Vous pouvez par exemple commencer le déploiement de quatre serveurs, puis passer à un déploiement de dix serveurs au fur et à mesure que les demandes de cache augmentent. De la même façon que vous pouvez ajouter des machines physiques et des unités de traitement pour l'évolutivité verticale, vous pouvez augmenter la capacité d'évolutivité de eXtreme Scale horizontalement grâce au partitionnement. Il s'agit d'une différence majeure avec les bases de données en mémoire (IMDB), par opposition à eXtreme Scale (qui est une grille de données), étant donné que les IMDB permettent uniquement une évolutivité verticale.

Avec WebSphere eXtreme Scale, vous pouvez également utiliser un ensemble d'interfaces de programme d'application (API) pour disposer d'un accès transactionnel aux données partitionnées et réparties. En termes de performances, les choix que vous faites pour interagir avec le cache sont tout aussi importants que les fonctions permettant de gérer la disponibilité du cache.

Remarque : L'évolutivité n'est pas disponible lorsque les conteneurs communiquent entre eux. Le protocole de gestion de la disponibilité, ou groupage central, est un algorithme de maintenance des signaux de présence et de l'affichage $O(N^2)$. Le nombre de membres du groupe central est limité à 20. Seule la réplication d'égal à égal entre les fragments existe.

Clients répartis

Le protocole client WebSphere eXtreme Scale prend en charge un grand nombre de clients. La stratégie de partitionnement suppose que tous les clients ne sont pas forcément intéressés par toutes les partitions, car les connexions peuvent s'étendre à plusieurs conteneurs. Les clients sont connectés directement aux partitions, de sorte que la latence est limitée à une connexion transférée.

Grilles, partitions et fragments

Une grille répartie eXtreme Scale est divisée en partitions. Une partition contient un sous-ensemble unique de données. Elle est composée d'un ou plusieurs fragments : un fragment primaire et des fragments répliques. Des fragments répliques ne doivent pas nécessairement être contenus dans une partition mais ils offrent une haute disponibilité. Que votre déploiement soit une grille de données

indépendante stockée en mémoire ou un espace de traitement de base de données en mémoire, l'accès aux données dans eXtreme Scale dépend étroitement des concepts de fragmentation.

Les données pour une partition sont stockées à l'exécution dans un ensemble de fragments. Cet ensemble de fragments inclut un fragment primaire et une ou plusieurs éventuels fragments répliques. Un fragment est la plus petite unité que eXtreme Scale puisse ajouter ou supprimer d'une machine virtuelle Java.

Il existe deux stratégies de positionnement : FIXED_PARTITIONS (par défaut) et PER_CONTAINER. La discussion ci-dessous traite de l'utilisation de la stratégie FIXED_PARTITIONS.

Nombre de fragments

Si votre environnement incluait dix partitions contenant un million d'objets sans réplique, dix fragments stockant 100 000 objets existeraient. Si vous ajoutez une réplique à ce scénario, un fragment supplémentaire existe dans chaque partition. Dans ce cas, 20 fragments existent : dix fragments primaires et dix répliques. Cette fois encore, chaque fragment stocke 100 000 objets. Chaque partition est composée d'un fragment primaire et d'une ou plusieurs répliques (N). La définition du nombre optimal de fragments s'avère déterminante. Si vous configurez une petite quantité de fragments, les données ne sont pas réparties de façon homogène entre les fragments, ce qui entraîne des erreurs de mémoire insuffisante et de surcharge du processeur. Dix fragments doivent au moins être configurés pour chaque JVM lors de la mise à l'échelle. Lors du déploiement initial de la grille, il est conseillé d'utiliser une grande quantité de partitions.

Nombre de fragments par JVM

Scénario : petite quantité de fragments pour chaque JVM

Les données sont ajoutées et supprimées d'une JVM à partir d'unités de fragments. Les fragments ne sont jamais divisés en plusieurs éléments. Pour 10 Go de données et 20 fragments qui les contiennent, chaque fragment contient en moyenne 500 Mo de données. Si neuf machines virtuelles Java hébergent la grille, chaque JVM comporte en moyenne deux fragments. Etant donné que le nombre 20 n'est pas divisible par 9, quelques machines virtuelles Java comportent trois fragments selon la distribution suivante :

- 7 machines virtuelles Java avec 2 fragments
- 2 machines virtuelles Java avec 3 fragments

Etant donné que chaque fragment contient 500 Mo de données, les données ne sont pas réparties de façon égale. Les sept machines virtuelles Java dotées de deux fragments hébergent chacune 1 Go de données. Les deux machines virtuelles Java dotées de trois fragments contiennent 50 % de données en plus, soit 1,5 Go, ce qui représente une charge nettement supérieure pour la mémoire. Puisque ces deux machines virtuelles Java hébergent trois fragments, elles reçoivent également 50 % de demandes de données en plus. Par conséquent, la configuration d'une petite quantité de fragments pour chaque JVM provoque un déséquilibre. Pour optimiser les performances, vous pouvez augmenter le nombre de fragments pour chaque JVM.

Scénario : quantité accrue de fragments pour chaque JVM

Dans ce scénario, augmentez nettement le nombre de fragments. Notre exemple illustre la configuration de 101 fragments avec 9 machines virtuelles Java hébergeant 10 Go de données. Dans ce cas, chaque fragment contient 99 Mo de données. Les machines virtuelles Java présentent le distribution de fragments suivante :

- 7 machines virtuelles Java avec 11 fragments
- 2 machines virtuelles Java avec 12 fragments

Les deux machines virtuelles Java dotées de 12 fragments contiennent 99 Mo de données en plus que les autres fragments, ce qui constitue une différence de 9 %. Ce scénario montre une distribution nettement plus égale que le scénario impliquant une petite quantité de fragments qui présente une différence de 50 %. Du point de vue de l'utilisation du processeur, une différence de traitement de 9 % seulement existe entre les deux machines virtuelles Java dotées de 12 fragments et les sept machines virtuelles Java dotées de 11 fragments. En augmentant le nombre de fragments dans chaque JVM, l'utilisation des données et du processeur est répartie de façon égale et juste.

Lorsque vous créez votre système, utilisez 10 fragments pour chaque JVM dans le scénario de taille maximale correspondant ou lorsque le système exécute le nombre maximal de machines virtuelles Java dans le cadre de votre planification.

Facteurs supplémentaires de positionnement

Le nombre de partitions, la stratégie de positionnement et le type des fragments répliques sont définis dans la règle de déploiement. Le nombre de fragments positionnés dépendra de la règle définie pour le déploiement. Les paramètres numInitialContainers, minSyncReplicas, developmentMode, maxSyncReplicas et maxAsyncReplicas affecteront l'emplacement et le moment où seront positionnés des partitions et des fragments répliques. Si le démarrage initial des serveurs ne permet pas le positionnement de maxSyncReplicas et de maxAsyncReplicas, vous pourrez faire positionner des fragments répliques supplémentaires en démarrant ultérieurement des serveurs supplémentaires. Lorsque vous planifiez le nombre de fragments par machine virtuelle Java, le nombre maximum de fragments, répliques comprises, dépendra s'il existe suffisamment de machines virtuelles Java pour permettre le nombre maximum de fragments répliques demandés. Un fragment réplique n'est jamais positionné dans le même processus que son fragment primaire, car en cas de perte du processus, ce sont les deux fragments qui seraient perdus. Lorsque developmentMode est false, le fragment primaire et ses fragments répliques ne seront pas positionnés sur la même machine.

Partitionnement

Le partitionnement permet de stocker d'importantes quantités de données dans la machine virtuelle Java (JVM). Pour partitionner les données, utilisez un schéma spécifié par une application pour diviser les données. Avec WebSphere eXtreme Scale, le partitionnement augmente à la fois l'extensibilité et la disponibilité.

Le partitionnement est le mécanisme utilisé par WebSphere eXtreme Scale pour monter en charge une application. Le partitionnement consiste en la séparation de l'application en parties contenant chacune un ensemble de données d'instance complètes. Le partitionnement n'est pas similaire à la technologie RAID, qui divise chaque instance de chaque segment. Chaque partition héberge les données complètes d'entrées individuelles. Le partitionnement est un moyen très efficace d'extension, mais il n'est pas valable pour toutes les applications. Les applications

requérant des garanties transactionnelles dans des ensembles de données volumineux ne sont pas évolutives et ne permettent pas un partitionnement efficace. eXtreme Scale ne prend actuellement pas en charge la validation en deux phases dans les partitions.

Important : Sélectionnez le nombre de partitions avec précaution. Le nombre de partitions défini dans la règle de déploiement influe sur le nombre de conteneurs à partir duquel une application peut être mise à l'échelle. Chaque partition est composée d'un fragment primaire et du nombre configuré de fragments de réplique. La formule `(Nombre_Partitions*(1 + Nombre_Répliques))` correspond au nombre de conteneurs qui peuvent être utilisés pour étendre une application.

Utiliser des partitions

Une grille peut avoir un grand nombre de partitions, des milliers, si nécessaire. Une grille peut monter jusqu'au produit du nombre de partitions multiplié par le nombre de fragments par partition. Si, par exemple, vous avez 16 partitions dont chacune comporte un fragment primaire et un fragment réplique, soit deux fragments, vous pouvez monter jusqu'à 32 machines virtuelles Java. Dans ce cas, il est défini un seul fragment pour chaque machine virtuelle Java. Vous devez choisir un nombre raisonnable de partitions en fonction du nombre de machines virtuelles Java que vous êtes censé utiliser. Pour le système, chaque fragment augmente l'utilisation de processeurs et de mémoire. Le système est conçu pour monter en puissance afin de gérer cette charge en fonction du nombre de machines virtuelles Java de serveurs qui sont disponibles.

Les applications ne doivent pas utiliser des milliers de partitions si elles s'exécutent dans une grille de quatre machines virtuelles Java conteneurs. Elles doivent être configurées de manière à n'avoir qu'un nombre raisonnable de fragments pour chaque machine virtuelle Java conteneur. Exemple de configuration déraisonnable : 2 000 partitions de deux fragments chacune, qui s'exécutent sur quatre machines virtuelles Java conteneurs. Résultat d'une telle configuration : 4 000 fragments placés dans quatre machines virtuelles Java conteneurs, soit 1 000 fragments par machine virtuelle Java conteneur.

Il serait plus intelligent d'avoir 10 fragments pour chacune des machines virtuelles Java attendues. Cette configuration n'empêche pas une évolutivité élastique de dix fois la configuration initiale tout en conservant une quantité raisonnable de fragments par machine virtuelle Java conteneur.

Prenons l'exemple suivant : vous avez actuellement six ordinateurs avec deux machines virtuelles Java conteneurs par ordinateur. Vous prévoyez de monter à vingt ordinateurs dans les trois années à venir. Avec vingt ordinateurs, vous avez quarante machines virtuelles Java conteneurs et, pour être pessimiste, vous choisissez un nombre de soixante. Vous voulez quatre fragments par machine virtuelle Java conteneur. Cela vous fait soixante conteneurs potentiels, soit un total de deux cent quarante fragments. Si vous avez un fragment primaire et un fragment réplique par partition, vous voudrez cent vingt partitions. Cet exemple vous donne 240 divisé par 12 machines virtuelles Java conteneurs, soit vingt fragments par machine virtuelle Java conteneur pour le déploiement initial avec le potentiel pour monter ultérieurement à vingt ordinateurs.

ObjectMap et partitionnement

Avec la stratégie FIXED_PARTITION de positionnement par défaut, les mappes sont fractionnées entre des partitions et des clés hachent vers les différentes

partitions. Le client n'a pas besoin de savoir à quelles partitions appartiennent les clés. Si un groupe de mappes comporte plusieurs mappes, ces mappes doivent être validées dans des transactions distinctes.

Entités et partitionnement

Les entités EntityManager sont dotées d'une optimisation qui aide les clients utilisant des entités sur un serveur. Le schéma d'entités sur le serveur pour le groupe de mappes peut spécifier une seule entité racine. Le client doit accéder à toutes les entités via cette entité racine. Le gestionnaire d'entités peut alors trouver dans la même partition les entités liées à partir de cette racine sans avoir besoin que les mappes liées aient une clé commune. C'est l'entité racine qui établit l'affinité avec la partition. Cette partition sert à toutes les recherches d'entités au sein de la transaction une fois que l'affinité a été établie. Cette affinité peut économiser de la mémoire car les mappes liées ne nécessitent pas de clé commune. L'entité racine doit être spécifiée avec une annotation d'entité modifiée, comme dans l'exemple suivant :

```
@Entity(schemaRoot=true)
```

Utilisez l'entité pour trouver la racine du graphe d'objets. Le graphe d'objets définit les relations entre une ou plusieurs entités. Chaque entité liée par une même relation doit se résoudre vers la même partition. Toutes les entités enfants sont censées se trouver dans la même partition que la racine. Les entités enfants présentes dans le graphe d'objets ne sont accessibles à un client qu'à partir de leur entité racine. Les entités racines sont toujours requises dans les environnements partitionnés lorsqu'on utilise un client eXtreme Scale pour communiquer avec le serveur. Il n'est possible de définir qu'une seule entité racine par clients. Les entités racines ne sont pas requises lorsqu'on utilise des ObjectGrids de style XTP (Extreme Transaction Processing) car toute la communication avec la partition s'effectue alors par accès direct en local et non via un mécanisme de client/serveur.

Placement et partitions

Deux stratégies s'offrent à vous dans WebSphere eXtreme Scale pour le positionnement : sur partition fixe et par conteneur. Le choix que vous ferez affectera la manière dont la configuration de votre déploiement place les partitions dans la grille distante.

Placement sur partition fixe

Vous pouvez définir dans le fichier XML de règles de déploiement la stratégie de positionnement. La stratégie par défaut est la stratégie de positionnement sur partition fixe, qui est activée avec le paramètre `FIXED_PARTITION`. Le nombre de fragments primaires qui sont placés dans les conteneurs disponibles est égal au nombre de partitions que vous avez configuré avec l'élément numberOfPartitions. Si vous avez configuré des fragments répliques, le nombre total minimum de fragments placés est défini par la formule suivante : `((1 fragment primaire + minimum de fragments synchrones)*partitions définies)`. Le nombre total maximum de fragments placés est défini par la formule suivante : `((1 fragment primaire + maximum de fragments synchrones + maximum de fragments asynchrones)*partitions définies)`. Votre déploiement WebSphere eXtreme Scale dissémine ces fragments dans les conteneurs disponibles. Les clés de chaque mappe sont hachées en partitions attribuées en fonction du nombre total de partitions que vous avez définies. Ces clés hachent vers la même partition même si celle-ci est déplacée pour cause de basculement ou de changement de serveur.

Si, par exemple, la valeur de numberPartitions est de 6 et celle de minSync est de 1 pour MapSet1, le nombre total de fragments pour ce groupe de mappes sera de 12 car chacune des 6 partitions requiert un fragment réplique synchrone. Si trois conteneurs sont démarrés, WebSphere eXtreme Scale placera quatre fragments par conteneur pour MapSet1.

Placement par conteneur

L'autre stratégie de positionnement est le positionnement par conteneur, laquelle s'active avec le paramètre `PER_CONTAINER` de placementStrategy dans l'élément mapset du fichier XML de déploiement. Avec cette stratégie, le nombre de fragments primaires qui sont placés dans chaque nouveau conteneur est égal au nombre *P* de partitions que vous avez configuré. L'environnement de déploiement de WebSphere eXtreme Scale place *P* répliques de chaque partition pour chaque conteneur restant. Le paramètre numInitialContainers est ignoré lorsqu'on utilise le positionnement par conteneur. Les partitions grandissent en même temps que les conteneurs. Dans cette stratégie, les clés des mappes ne sont pas fixées à une partition données. C'est le client qui route vers une partition en utilisant une partition primaire aléatoire. Pour pouvoir se reconnecter à une session qu'il a déjà utilisée pour trouver une clé, le client doit utiliser un descripteur de session.

Pour de plus amples informations, voir dans le *Guide de programmation* la session consacrée à l'utilisation d'un SessionHandle pour le routage.

En cas de basculement ou d'arrêt de serveur, dans la stratégie de positionnement par conteneur, l'environnement WebSphere eXtreme Scale déplace les fragments primaires si ces fragments contiennent encore des données. S'ils sont vides, ils sont éliminés. Dans la stratégie par conteneur, les anciens fragments primaires ne sont pas conservés car de nouveaux fragments primaires sont placés pour chaque conteneur.

WebSphere eXtreme Scale permet une autre stratégie de positionnement, le positionnement par conteneur, qui peut être utilisée à la place de ce que l'on pourrait appeler la stratégie "normale", à savoir l'approche partition fixe dans laquelle la clé de mappe est hachée sur l'une des partitions. Dans une approche par conteneur (définie alors à l'aide de PER_CONTAINER), le déploiement place les partitions dans l'ensemble des serveurs conteneurs en ligne en en augmentant ou diminuant le nombre au fur et à mesure que des conteneurs sont ajoutés à la grille ou en sont retirés. Dans une approche partition fixe, une grille fonctionne très bien si elle se base sur des clés, c'est-à-dire si l'application utilise un objet key pour repérer les données dans la grille. Ici, nous allons voir l'autre approche.

Exemple de grille par conteneur

Les grilles PER_CONTAINER se présentent différemment. La spécification d'une grille PER_CONTAINER se fait par l'attribut placementPolicy du fichier XML de déploiement. Au lieu de configurer le nombre total de partitions que l'on veut dans la grille, l'on spécifie le nombre de partitions que l'on veut par conteneur démarré.

Par exemple, si vous définissez à 5 le nombre de partitions par conteneur, lorsque vous démarrez un conteneur, eXtreme Scale créera 5 partitions primaires anonymes sur ce conteneur et autant de fragments répliques que nécessaires sur les autres conteneurs déjà déployés.

Voici un exemple d'enchaînement potentiel lors de la croissance d'une grille dans un environnement par conteneur.

1. L'on démarre le conteneur C0 qui héberge 5 partitions primaires (P0-P4).
   - C0 héberge P0, P1, P2, P3, P4.
2. L'on démarre le conteneur C1 qui héberge 5 autres partitions primaires (P5-P9). Des fragments répliques sont répartis de manière équilibrée sur les conteneurs.
   - C0 héberge P0, P1, P2, P3, P4, R5, R6, R7, R8, R9.
   - C1 héberge P5, P6, P7, P8, P9, R0, R1, R2, R3, R4.
3. L'on démarre le conteneur C2 qui héberge 5 autres partitions primaires (P10-P14). Là aussi, des fragments répliques sont répartis de manière équilibrée sur les conteneurs.
   - C0 héberge P0, P1, P2, P3, P4, R7, R8, R9, R10, R11, R12.
   - C1 héberge P5, P6, P7, P8, P9, R2, R3, R4, R13, R14.
   - C2 héberge P10, P11, P12, P13, P14, R5, R6, R0, R1.

Le schéma continue au fur et à mesure que d'autres conteneurs sont ajoutés, en créant 5 partitions primaires à chaque fois et redistribuant les fragments répliques sur les conteneurs disponibles dans la grille.

Remarque : Dans le cadre de la stratégie PER_CONTAINER, WebSphere eXtreme Scale ne déplace pas de partitions primaires, mais uniquement des fragments répliques.

Ne perdez pas de vue que, le nombre des partitions étant arbitraire et n'ayant rien à voir avec des clés, il est impossible d'utiliser un routage à base de clés. Si un conteneur s'arrête, les ID de partitions créés pour ce conteneur ne sont plus utilisés, ce qui fait qu'il y a un trou dans les ID de partitions. Dans notre exemple, en cas de défaillance du conteneur C2, il n'y aurait plus de partitions P5-P9 et il ne resterait plus que les partitions P0-P4 et P10-P14, ce qui rend impossible tout hachage à base de clés.

L'utilisation de chiffres comme 5, voire 10, pour le nombre de partitions par conteneur fonctionne mieux si l'on envisage les conséquences de la défaillance d'un conteneur. Pour répartir de manière égale dans la grille la charge des fragments, l'on a besoin de davantage qu'une seule partition pour chaque conteneur. Si l'on n'a qu'une seule partition par conteneur, en cas de défaillance de l'un des conteneurs, c'est un seul conteneur (celui qui héberge le fragment réplique correspondant) qui devra porter toute la charge du fragment primaire perdu. Dans ce cas, la charge est immédiatement doublée pour le conteneur. Au contraire, si l'on a 5 partitions par conteneur, ce sont 5 conteneurs qui recueilleront la charge du conteneur perdu, d'où, pour chacun, un impact moindre de 80 %. L'utilisation de plusieurs partitions par conteneur diminue de manière substantielle l'impact potentiel sur chacun des conteneurs. Plus directement, l'on peut envisager le cas où un conteneur connaît de manière inopinée des pics d'utilisation. La charge de la réplication sur ce conteneur sera alors répartie sur 5 autres conteneurs et non uniquement sur un seul.

Utiliser la stratégie par conteneur

Plusieurs scénarios font de la stratégie par conteneur la configuration idéale, pour la réplication de sessions HTTP, par exemple, ou pour l'état de session d'application. Dans un pareil cas, un routeur HTTP affecte une session à un conteneur de servlet. Ce dernier a besoin de créer une session HTTP et il choisit l'une des 5 partitions primaires locales. L'"ID" de la partition choisie est alors stocké dans un cookie. Le conteneur de servlet peut désormais accéder en local à l'état de la session, ce qui signifie un temps d'attente nul pour l'accès aux données

dans le cadre de cette demande aussi longtemps que l'on maintient l'affinité de session. Une grille eXtreme Scale réplique toutes les modifications apportées à la partition.

Dans la pratique, rappelez-vous ce que nous disions plus haut des conséquences avantageuses entraînées par le fait d'avoir plusieurs partitions (nous disions 5) par conteneur. Naturellement, chaque nouveau conteneur qui démarre ajoute 5 nouvelles partitions et 5 autres fragments répliques. Au fil du temps, d'autres partitions doivent normalement être créées et elles ne doivent ni être déplacées ni être détruites. Mais ce n'est pas ainsi que se comportent en fait les conteneurs. Lorsqu'un conteneur démarre, il héberge 5 fragments primaires, que l'on peut appeler des fragments primaires "home", et qui existent dans les conteneurs respectifs qui les ont créés. En cas de défaillance du conteneur, les fragments répliques deviennent des fragments primaires et eXtreme Scale crée 5 autres fragments répliques afin de conserver la haute disponibilité (à moins que l'on n'ait désactivé la réparation automatique). Les nouveaux fragments primaires se trouvant sur un conteneur autre que celui qui les a créés, on peut les appeler des fragments primaires "externes". L'application ne doit en aucun cas placer un nouvel état ou de nouvelles sessions dans un fragment primaire externe. Au final, le fragment primaire externe ne comporte aucune entrée et eXtreme Scale le supprime automatiquement ainsi que les fragments répliques qui lui sont associés. Les fragments primaires externes n'ont de raison d'être que de permettre aux sessions existantes d'être toujours disponibles (aux sessions existantes, mais non à de nouvelles sessions).

Un client peut toujours interagir avec une grille qui ne s'appuie pas sur des clés. Le client commence juste une transaction et il stocke les données dans la grille indépendamment de quelque clé que ce soit. Il réclame à la session un objet SessionHandle, qui est un descripteur sérialisable lui permettant d'interagir avec la même partition lorsque c'est nécessaire. Pour plus d'informations, voir dans le *Guide de programmation* la rubrique consacrée à l'utilisation d'un SessionHandle pour le routage. WebSphere eXtreme Scale choisit une partition pour le client dans la liste des partitions primaires home. Il ne retourne jamais de partition primaire externe. SessionHandle peut être sérialisé en cookie HTTP, par exemple, et le cookie peut ultérieurement être reconverti en SessionHandle. Les API WebSphere eXtreme Scale peuvent alors, à l'aide du SessionHandle, obtenir à nouveau une session liée à la même partition.

Remarque : L'on ne peut utiliser d'agents pour interagir avec une grille PER_CONTAINER.

Avantages

Ce que nous venons d'expliquer diffère d'une grille FIXED_PARTITION normale (grille avec hachage) parce que le client par conteneur stocke les données à un endroit de la grille, obtient un descripteur pour ces données et utilise ce descripteur pour accéder à nouveau à ces données. Il n'existe aucune clé fournie par application comme c'est le cas dans la partition fixe.

Le déploiement ne crée pas de nouvelle partition pour chaque session. De ce fait, dans un déploiement par conteneur, les clés qui servent à stocker les données dans la partition doivent exister en un seul exemplaire au sein de cette partition. L'on peut, par exemple, faire générer par le client un SessionID unique que l'on utilisera comme clé afin de trouver les informations dans les mappes de cette partition. De multiples sessions clients interagissent avec la même partition ; c'est pourquoi

l'application a besoin d'utiliser des clés uniques pour stocker les données de session dans chacune des partitions concernées.

Les exemples donnés plus haut utilisaient 5 partitions, mais le paramètre numberOfPartitions du fichier XML objectgrid permet de spécifier autant de partitions que l'on veut. Simplement, au lieu d'être par grille, ce nombre est par conteneur (le nombre de fragments répliques est spécifié de la même manière qu'avec la stratégie de partition fixe).

La stratégie par conteneur est également utilisable avec des zones multiples. Dans la mesure du possible, eXtreme Scale retourne un SessionHandle à une partition dont le fragment primaire est situé dans la même zone que le client. Celui-ci peut spécifier la zone au conteneur comme paramètre ou à 'aide d'une API. L'ID de zone du client peut être défini à l'aide de `serverproperties` ou de `clientproperties`.

La stratégie PER_CONTAINER convient aux applications qui stocke un état de type conversationnel plutôt que des données orientées base de données. La clé permettant d'accéder aux données sera un ID de conversation et non un enregistrement spécifique de base de données. Cette stratégie permet des performances plus élevées (car les partitions primaires peuvent cohabiter, avec des servlets, par exemple) et une configuration plus facile (pas besoin de calculer les partitions et les conteneurs).

Transactions dans une partition unique et transactions dans plusieurs partitions de la grille

La différence majeure entre WebSphere eXtreme Scale et les solutions classiques de stockage de données (bases de données relationnelles ou bases de données en mémoire) consiste en l'utilisation du partitionnement, qui permet au cache d'évoluer de manière linéaire. Les principaux types de transactions sont les transactions impliquant une seule partition et les transactions impliquant toutes les partitions (d'une grille).

Les interactions avec le cache se rangent dans deux catégories : les transactions impliquant une seule partition et les transactions impliquant l'ensemble de la grille. Ces deux types de transactions sont décrits ci-dessous.

Transactions dans une partition unique

Les transactions dans une partition unique constituent le mode préférentiel d'interaction avec les mémoires cache hébergées par WebSphere eXtreme Scale. Lorsqu'une transaction est limitée à une partition, elle se limite par défaut à une seule machine virtuelle Java et donc à un seul serveur. Un serveur peut exécuter un nombre *M* de transactions par seconde. Si votre système contient *N* ordinateurs, vous pouvez exécuter `M*N` transactions par seconde. Si votre activité augmente et que vous ayez besoin de doubler le nombre de transactions par seconde, vous pouvez doubler *N* en installant davantage d'ordinateurs. Vous pouvez alors répondre à vos besoins en capacité sans modifier l'application, mettre à niveau le matériel ni utiliser l'application hors ligne.

Outre qu'elles permettent au cache d'évoluer de manière significative, les transactions s'exécutant dans une seule partition permettent aussi d'optimiser la disponibilité de ce dernier. Chaque transaction est liée à un seul ordinateur. Une défaillance peut se produire sur l'un des autres ordinateurs `(N-1)` sans que la

réussite ou le temps de réponse de la transaction en soit affecté. Si 100 ordinateurs sont en cours d'exécution et que l'un d'eux échoue, 1 % des transactions en cours au moment de l'échec sont annulées. Après cet échec, WebSphere eXtreme Scale relocalise les partitions qui sont hébergées par le serveur défaillant vers les 99 autres ordinateurs. Pendant cette courte période, ces ordinateurs continuent à exécuter des transactions. Seules les transactions impliquant les partitions qui sont en cours de relocalisation sont bloquées. Une fois le basculement terminé, le cache peut continuer à s'exécuter en étant pleinement opérationnel, c'est-à-dire à 99 % de sa capacité de traitement d'origine. Une fois le serveur défaillant remplacé et revenu dans la grille, le cache retrouve 100 % de sa capacité de traitement.

Transactions dans l'ensemble de la grille

En termes de performances, de disponibilité et d'évolutivité, les transactions s'exécutant dans l'ensemble de la grille sont l'opposé des transactions impliquant une seule partition. Elles accèdent à toutes les partitions et donc à chaque ordinateur de la configuration. Chaque ordinateur de la grille est sollicité pour rechercher des données, puis renvoyer le résultat. La transaction n'est pas terminée tant que tous les ordinateurs n'ont pas répondu. La capacité de traitement de la grille est donc limitée par l'ordinateur le plus lent. L'ajout d'ordinateurs ne permet pas d'améliorer la vitesse de l'ordinateur le plus lent ni d'augmenter la capacité de traitement du cache.

Les transactions impliquant l'ensemble de la grille ont des effets semblables sur la disponibilité. Reprenons l'exemple précédent : si 100 serveurs sont en cours d'exécution et que l'un d'eux échoue, 100 % des transactions en cours sont annulées. Après cet échec, WebSphere eXtreme Scale commence à relocaliser les partitions qui sont hébergées par ce serveur vers les 99 autres ordinateurs. En attendant la fin du basculement, la grille ne parvient à traiter aucune de ces transactions. Une fois le basculement terminé, le cache continue à s'exécuter, mais à un niveau de capacité réduit. Si chaque ordinateur de la grille gère 10 partitions, ces 10 ordinateurs reçoivent au moins une partition supplémentaire dans le cadre du basculement. L'ajout d'une partition supplémentaire augmente la charge de travail de cet ordinateur d'au moins 10 %. La capacité de la grille de traitement étant limitée à la capacité de traitement de l'ordinateur le plus lent, cette capacité est limitée en moyenne de 10 %.

Les transactions s'exécutant dans une partition unique sont préférables aux partitions impliquant l'ensemble de la grille pour garantir l'évolutivité d'un cache objet réparti hautement disponible comme WebSphere eXtreme Scale. Pour optimiser les performance de ces systèmes, vous devez utiliser des techniques autres que les méthodologies relationnelles traditionnelles, mais vous pouvez transformer les transactions impliquant l'ensemble de la grille en transactions s'exécutant dans une seule partition.

Méthodes recommandées en matière de génération de modèles de données évolutifs

Les méthodes recommandées pour générer des applications évolutives avec des produits tels que WebSphere eXtreme Scale sont au nombre de deux : les principes fondamentaux et les conseils relatifs à l'implémentation. Les principes fondamentaux sont les grandes idées que vous devez capturer lors de la conception des données. Une application n'observant pas ces principes aura peu de chance de pouvoir évoluer de manière satisfaisante, même pour les transactions principales. Les conseils d'implémentation s'appliquent aux transactions posant problème dans une application par ailleurs bien conçue et observant les principes

généraux relatifs aux modèles de données évolutifs.

Principes fondamentaux

Certains concepts ou principes de base à ne pas oublier constituent les éléments clés pour optimiser l'évolutivité.

Duplication plutôt que normalisation

Il est essentiel de bien comprendre que les produits tels que WebSphere eXtreme Scale sont conçus pour répartir des données dans un grand nombre d'ordinateurs. Si votre objectif est d'exécuter la plupart ou l'ensemble des transactions sur un même ordinateur, le modèle de données doit s'assurer que toutes les données nécessaires à la transaction sont situées dans cette partition. Dans la plupart des cas, la seule manière d'y parvenir consiste à dupliquer les données.

Prenons l'exemple d'un forum électronique. Deux transactions très importantes pour ce forum présentent tous les messages publiés par un utilisateur donné et tous les messages publiés sur un sujet donné. Imaginez tout d'abord comment ces transactions se comporteraient dans le cadre d'un modèle de données normalisé contenant un enregistrement utilisateur, un enregistrement relatif au sujet et un enregistrement relatif au message et contenant le texte réel. Si les messages publiés sont partitionnés avec les enregistrements utilisateur, l'affichage du sujet devient une transaction impliquant l'ensemble de la grille, et inversement. Il est impossible de partitionner les sujets et les utilisateurs car leur relation est de type many-to-many.

La meilleure méthode pour permettre à ce forum électronique d'évoluer est de dupliquer les messages publiés, c'est-à-dire de copier chaque message avec l'enregistrement relatif au sujet et avec l'enregistrement utilisateur. L'affichage des messages publiés à partir d'un utilisateur constitue alors une transaction dans une partition unique, de même que l'affichage des messages publiés dans un sujet, tandis que la mise à jour ou la suppression d'un message publié est une transaction impliquant deux partitions. Ces trois transactions évoluent de manière linéaire au fur et à mesure que le nombre d'ordinateurs de la grille augmente.

L'évolutivité plutôt que les ressources

Le principal obstacle à surmonter en matière de modèles de données dénormalisés est l'impact de ces modèles sur les ressources. La conservation de deux ou trois copies, voire plus, de certaines données risque d'entraîner la consommation d'une trop grande quantité de ressources pour être pratique. Lorsque vous êtes confronté à un tel scénario, gardez ceci en mémoire : les coûts liés à l'achat de matériel diminuent d'année en année. Autre considération, et non des moindres : WebSphere eXtreme Scale permet de supprimer la plupart des coûts associés au déploiement de ressources supplémentaires.

Mesurez les ressources en termes de coût plutôt qu'en termes purement informatiques comme les mégaoctets et les processeurs. Les magasins de données utilisant des données relationnelles normalisées doivent généralement être situés sur le même ordinateur. Cela signifie que vous devez acheter un seul ordinateur d'entreprise puissant, plutôt que plusieurs machines modestes. Il n'est pas rare qu'un ordinateur d'entreprise

pouvant exécuter un million de transactions par seconde soit bien plus onéreux que dix ordinateurs pouvant traiter 100 000 transactions par seconde.

L'ajout de ressources a également un coût. Une entreprise en pleine croissance finit par manquer de capacité. Lorsque vous vous trouvez dans cette situation, vous devez soit arrêter votre système lors du passage à un ordinateur plus rapide et plus puissant, soit créer un second environnement de production. Quelle que soit l'option choisie, vous devrez prendre en charge des coûts supplémentaires liés à la réduction du volume de traitement ou au maintien de la capacité de traitement, pratiquement doublée pendant la durée de la transaction.

Avec WebSphere eXtreme Scale, il n'est pas nécessaire de fermer l'application lors de l'accroissement de la capacité. Si les prévisions relatives à votre entreprise indiquent que vous devez augmenter de 10 % votre capacité pour l'année à venir, augmentez d'autant le nombre d'ordinateurs de la grille. Ce pourcentage peut augmenter sans entraîner d'indisponibilité de l'application et sans que vous ayez à accroître la capacité.

Des transformations de données inutiles

Lorsque vous utilisez WebSphere eXtreme Scale, vous devez stocker les données dans un format directement utilisable par la logique métier. La répartition des données dans un format plus primitif consomme davantage de ressources. La transformation doit se faire lors de l'écriture et lors de la lecture des données. Dans le cas d'une base de données relationnelle, cette transformation est nécessaire car les données sont enregistrées fréquemment sur le disque, mais avec WebSphere eXtreme Scale, elle n'est pas obligatoire. La plupart des données sont stockées en mémoire et peuvent donc être stockées au format requis par l'application.

L'observation de cette règle simple vous aide à dénormaliser les données conformément au premier principe. Le type de transformation des données métier le plus commun est l'opération JOIN nécessaire pour transformer des données normalisées en un ensemble de résultats correspondant aux besoins de l'application. Le stockage des données au format correct permet implicitement d'éviter d'avoir à exécuter ces opérations de type JOIN et génère un modèle de données dénormalisé.

Abandon des requêtes illimitées

Quelle que soit la structure de vos données, les requêtes illimitées évoluent mal. Nous ne vous recommandons par exemple pas d'exécuter une transaction visant à obtenir une liste de tous les articles triés par valeur. Elle peut renvoyer un résultat correct au début, lorsque le nombre d'articles est égal à 1 000, mais lorsque celui-ci atteint 10 millions, la transaction renvoie ces 10 millions d'articles. L'exécution d'une telle transaction risque d'entraîner un délai d'inactivité de celle-ci ou une erreur liée à une insuffisance de mémoire du client.

La meilleure solution consiste à modifier la logique métier de façon que seuls les 10 ou 20 vingt premiers articles soient renvoyés. Cette modification permet de faire en sorte que la transaction soit gérable quel que soit le nombre d'articles présents en cache.

Définition d'un schéma

Le principal avantage lié à la normalisation des données est que la base de données peut prendre en charge la cohérence des données en arrière-plan.

Lorsque les données sont dénormalisées à des fins d'évolutivité, la gestion automatique de la cohérence des données disparaît. Pour la rétablir, vous devez implémenter un modèle de données pouvant fonctionner dans la couche d'applications ou en tant que plug-in de la grille répartie.

Prenons l'exemple du forum électronique. Si une transaction supprime un message publié d'un sujet, sa copie dans l'enregistrement utilisateur doit également être supprimée. Sans modèle de données, il est possible que le développeur prévoie la suppression du message publié dans le code de l'application, mais qu'il oublie de le supprimer de l'enregistrement utilisateur. Toutefois, s'il avait utilisé un modèle de données au lieu d'interagir directement avec le cache, la méthode removePost aurait permis d'extraire l'ID utilisateur du message, de rechercher l'enregistrement utilisateur et de supprimer la copie du message en arrière-plan.

Vous pouvez également implémenter un programme d'écoute s'exécutant sur la partition et capable de détecter la modification apportée au sujet et de modifier automatiquement l'enregistrement utilisateur. Un tel programme peut se révéler avantageux car la modification de l'enregistrement utilisateur est effectuée localement si celui-ci se trouve sur la partition ou, s'il se trouve sur une autre partition, la transaction est exécutée entre deux serveurs et non entre le client et le serveur. La connexion réseau entre des serveurs est probablement plus rapide que la connexion existant entre le client et le serveur.

Disparition des conflits

Evitez les scénarios contenant par exemple un compteur global. La grille ne parvient pas à évoluer si un enregistrement unique est utilisé un nombre disproportionné de fois par rapport aux autres enregistrements. Les performances de la grille seront limitées par les performances de l'ordinateur détenant cet enregistrement.

Dans une telle situation, essayez de fractionner l'enregistrement afin qu'il soit géré au niveau de la partition. Prenons le cas d'une transaction renvoyant le nombre total d'entrées présentes dans le cache réparti. Plutôt que d'exécuter une opération d'insertion et de suppression accédant à un enregistrement unique qui s'incrémente, installez un programme d'écoute sur chaque partition pour suivre ces opérations. Grâce à ce suivi, les opérations d'insertion et de suppression deviennent des transactions s'exécutant dans une partition unique.

La lecture du compteur devient une opération impliquant l'ensemble de la grille, mais elle était déjà en grande partie aussi inefficace qu'une telle opération car ses performances étaient liées à celles de l'ordinateur hébergeant l'enregistrement.

Conseils relatifs à l'implémentation

Pour optimiser l'évolutivité, prenez en compte les conseils suivants.

Utilisation des index de recherche inversée

Prenons le cas d'un modèle de données dénormalisé dans lequel les enregistrements client sont partitionnés en fonction du numéro d'ID du client. Cette méthode de partitionnement est un choix logique car pratiquement toutes les opérations métier exécutées avec l'enregistrement client utilisent cet ID. Toutefois, la transaction de connexion, qui est essentielle, ne l'utilise pas. Les données utilisées pour la connexion sont plus fréquemment le nom d'utilisateur ou l'adresse électronique.

L'approche la plus simple consiste alors à utiliser une transaction impliquant l'ensemble de la grille pour rechercher l'enregistrement client. Comme expliqué précédemment, cette approche n'est pas évolutive.

Autre option : procéder à un partitionnement en fonction du nom d'utilisateur ou de l'adresse électronique. Cette option n'est pas pratique car toutes les opérations liées à l'ID client deviendraient alors des transactions impliquant l'ensemble de la grille. De plus, les clients de votre site pourraient souhaiter modifier leur nom d'utilisateur ou leur adresse électronique. Dans les produits tels que WebSphere eXtreme Scale, la valeur utilisée pour partitionner les données doit rester constante.

La bonne solution consiste alors à utiliser un index de recherche inversée. Avec WebSphere eXtreme Scale, il est possible de créer un cache dans la même grille répartie que le cache contenant tous les enregistrements utilisateur. Ce cache est à haute disponibilité, partitionnée et évolutif. Il permet de mapper un nom d'utilisateur ou une adresse électronique vers un ID client. Plutôt que d'avoir une opération impliquant l'ensemble de la grille, il transforme la procédure de connexion en transaction s'exécutant sur deux partitions. Ce scénario n'est pas aussi optimal qu'une transaction impliquant une seule partition, mais la capacité de traitement augmente cependant de manière linéaire par rapport au nombre d'ordinateurs.

Calcul des valeurs lors de l'écriture

Les calculs les plus fréquents, tels que les moyennes ou les totaux, peuvent consommer une grande quantité de ressources car ils supposent la lecture d'un grand nombre d'entrées. Les lectures étant plus fréquentes que les écritures dans la plupart des applications, il est plus efficace de calculer ces valeurs lors de l'écriture, puis de stocker le résultat dans le cache. Les opérations gagnent ainsi en rapidité et en évolutivité.

Zones facultatives

Prenons l'exemple d'un enregistrement utilisateur contenant un numéro de téléphone professionnel, un numéro de téléphone personnel et un numéro de téléphone portable. Tous ces numéros ou une combinaison d'entre eux (ou aucun) peuvent être définis pour un utilisateur. Si les données ont été normalisées, une table utilisateur et une table contenant les numéros de téléphone existent. Vous pouvez alors identifier les numéros de téléphone d'un utilisateur donné à l'aide d'une opération JOIN entre les deux tables.

Pour dénormaliser cet enregistrement, aucune duplication des données n'est nécessaire, car la plupart des utilisateurs ne partagent pas leurs numéros de téléphone. Au contraire, les emplacements vides doivent être autorisés dans l'enregistrement utilisateur. Au lieu de constituer une table contenant les numéros de téléphone, ajoutez trois attributs à l'enregistrement utilisateur, chacun correspondant à un type de numéro de téléphone. L'ajout de ces attributs rend superflue l'opération JOIN et permet d'effectuer la recherche des numéros de téléphone d'un utilisateur en tant qu'opération impliquant une seule partition.

Positionnement des relations many-to-many

Prenons l'exemple d'une application qui assure le suivi de certains produits et des magasins dans lesquels ceux-ci sont commercialisés. Un même produit est vendu dans plusieurs magasins et un même magasin vend plusieurs produits. Supposons que cette application assure le suivi de 50 revendeurs importants. Chaque produit est vendu dans 50 magasins au maximum, chaque magasin commercialisant des milliers de produits.

Constituez une liste des magasins dans l'entité produit (organisation A) plutôt qu'une liste des produits dans chaque entité magasin (organisation B). Une simple observation des transactions que cette application devrait exécuter permet de comprendre pourquoi l'organisation A est la plus évolutive.

Considérons d'abord les mises à jour. Dans l'organisation A, la suppression d'un produit du stock d'un magasin verrouille l'entité produit. Si la grille contient 10 000 produits, seul 1/10 000è de la grille doit être verrouillé lors de la mise à jour. Dans l'organisation B, la grille contient 50 magasins, si bien qu'1/50è de la grille doit être verrouillé pendant la mise à jour. Même si les deux opérations peuvent être considérées comme des opérations impliquant une seule partition, l'organisation A permet une meilleure évolutivité.

Considérons maintenant les opérations de lecture avec l'organisation A : la recherche des magasins dans lesquels un produit est commercialisé est une transaction impliquant une seule partition, pouvant évoluer et rapide car elle transmet une faible quantité de données. Avec l'organisation B, cette transaction devient une transaction impliquant plusieurs grilles car chaque entité de magasin doit faire l'objet d'un accès permettant d'identifier si le produit est vendu dans ce magasin, ce qui donne un avantage certain à l'organisation A en termes de performances.

Evolutivité avec les données normalisées

Les transactions impliquant l'ensemble de la grille sont utilisées à juste titre pour faire évoluer le traitement des données. Si une grille contient cinq ordinateurs et qu'une transaction visant à trier environ 100 000 enregistrements dans chaque ordinateur est lancée sur l'ensemble de la grille, cette transaction trie 500 000 enregistrements. Si l'ordinateur le plus lent peut traiter 10 de ces transactions par seconde, la grille peut trier 5 millions d'enregistrements par seconde. Si les données de la grille doublent, chaque ordinateur doit trier 200 000 enregistrements et chaque transaction trie 1 million d'enregistrements. Cette progression des données ramène la capacité de traitement de l'ordinateur le plus lent à cinq transactions par seconde et celle de la grille à cinq transactions par seconde. La grille trie toutefois 5 millions d'enregistrements par seconde.

Dans un tel scénario, le fait de doubler le nombre d'ordinateurs permet à chaque ordinateur de revenir à sa charge précédente et à l'ordinateur le plus lent de traiter 10 de ces transactions par seconde. La capacité de traitement de la grille reste la même (10 demandes par seconde), mais chaque transaction traite désormais 1 million d'enregistrements, si bien que la grille a doublé sa capacité en termes de traitement d'enregistrements, atteignant les 10 millions d'enregistrements par seconde.

Avec les applications telles que les moteurs de recherche, qui ont besoin d'évoluer en termes de traitement des données pour s'adapter à la taille croissante de l'Internet et en termes de capacité afin de s'adapter à la croissance du nombre d'utilisateurs, vous devez créer plusieurs grilles avec permutation circulaire des demandes entre les grilles. Si vous avez besoin de faire évoluer la capacité de traitement, ajoutez des ordinateurs et ajoutez une grille pour gérer les demandes. Si le traitement des données doit évoluer, ajoutez des ordinateurs et conservez le même nombre de grilles.

Mise à l'échelle en unités ou capsules

Cette rubrique aborde le problème de l'évolutivité d'une nouvelle façon, autre que celle qui consiste à étendre une grille à un nombre x de machines virtuelles Java. L'évolutivité est abordée ici en termes d'opérations, de planification et de gestion des risques. Bien que ces facteurs soient généralement plus importants que les considérations habituelles d'évolutivité d'un produit, ils sont malheureusement souvent ignorés. La génération de systèmes hautement disponibles requiert de prendre en compte ces deux types de considérations pour déployer eXtreme Scale dans un processus de déploiement fiable.

Déploiement d'une grille unique de grande taille

Les tests ont montré que eXtreme Scale peut s'étendre à plus de 1 000 machines virtuelles Java. Ces tests encouragent la création d'applications permettant de déployer des grilles uniques sur un grand nombre de sous-systèmes de stockage. Bien que cela soit possible, cela n'est pas recommandé pour les raisons suivantes.

1. **Problème de budget :** il n'est pas réaliste d'envisager que votre environnement teste une grille de 1 000 serveurs. Cependant, il peut tester une grille beaucoup plus petite en prenant compte du budget. Il est donc inutile d'acheter deux fois le matériel, en particulier pour un nombre si important de serveurs.
2. **Différentes versions d'application :** l'utilisation d'un grand nombre de sous-systèmes de stockage pour chaque unité d'exécution de test n'est pas pratique. Le risque est de ne pas tester les mêmes facteurs que si vous étiez dans un environnement de production.
3. **Perte de données :** l'exécution d'une base de données sur un lecteur de disque dur n'est pas fiable. Tout problème avec le disque dur peut entraîner la perte de données. L'exécution d'une application dont la taille augmente sur une grille unique est similaire. Il est probable que vous rencontriez des bogues dans votre environnement et vos applications. Le positionnement toutes les données sur un seul grand système mène souvent à la perte d'un grand nombre de données.

Fractionnement de la grille

Le fractionnement de la grille de l'application en capsules (unités) est plus fiable. Une capsule est un groupe de serveurs exécutant une pile d'applications homogène. Les capsules peuvent avoir n'importe quelle taille, mais elles doivent idéalement se composer de 20 sous-systèmes de stockage. Plutôt que d'avoir 500 sous-systèmes de stockage dans une seule grille, vous pouvez avoir 25 capsules de 20 sous-systèmes de stockage. Une seule version d'une pile d'applications doit s'exécuter sur une capsule donnée, mais les différentes capsules peuvent avoir leur propre version d'une pile d'applications.

Généralement, une pile d'applications prend en compte les niveaux des composants suivants.

- système d'exploitation
- matériel
- machines virtuelles Java
- version d'eXtreme Scale
- application
- autres composants nécessaires

Une capsule est une unité de déploiement de taille adaptée aux tests. Dans le cadre des tests, il est plus pratique d'avoir 20 serveurs plutôt que plusieurs centaines.

Vous continuez néanmoins de tester la même configuration que vous auriez en production. La production utilise des grilles de 20 serveurs maximum, chacune constituant une capsule. Vous pouvez effectuer des tests de contrainte sur une capsule, puis déterminer sa capacité, le nombre d'utilisateurs, la quantité de données et la capacité de traitement. Cela facilite la planification et permet une évolutivité et des coûts prévisibles.

Configuration d'un environnement basé sur capsule

Selon les cas, la capsule ne contient pas forcément 20 serveurs. La taille de la capsule doit être adaptée aux tests. La taille d'une capsule doit être telle que, si la capsule rencontre un problème en production, la quantité des transactions affectées est tolérable.

Idéalement, un bogue n'a de conséquence que sur une seule capsule. Dans l'exemple précédent, un bogue n'aurait un impact que sur 4 % des transactions de l'application, au lieu de 100 %. De plus, les mises à niveau sont facilitées, car elles peuvent être appliquées à une seule capsule à la fois. Cela est utile dans le cas où une mise à niveau crée un problème sur une capsule : l'utilisateur peut revenir à la version antérieure de la capsule. Les mises à niveau incluent toutes les modifications apportées à l'application, la pile d'applications ou les mises à jour système. Dans la mesure du possible, les mises à niveau doivent modifier un seul élément de la pile à la fois, afin de permettre un diagnostic plus précis des problèmes.

Pour implémenter un environnement, vous avez besoin d'une couche de routage au-dessus des capsules qui soit compatible avec les versions antérieures et ultérieures si les capsules reçoivent des mises à niveau de logiciel. Vous devez également créer un répertoire contenant les informations sur le contenu de chaque capsule. Pour cela, vous pouvez utiliser, de préférence en écriture différée, une autre grille eXtreme Scale appuyée sur une base de données. Cela génère une solution à deux niveaux. Le niveau 1 est le répertoire, et est utilisé pour déterminer quelle capsule gère quelle transaction. Le niveau 2 se compose des capsules. Lorsque le niveau 1 identifie une capsule, la configuration achemine chaque transaction vers le serveur approprié dans la capsule, qui est généralement le serveur contenant la partition pour les données utilisées par la transaction. Le cas échéant, vous pouvez également utiliser un cache proche sur le niveau 1 afin de minimiser l'impact généré par la recherche de la capsule adéquate.

L'utilisation des capsules est légèrement plus complexe que l'utilisation d'une grille unique, mais les améliorations que cela suppose en termes de capacité opérationnelle, de test et de fiabilité en font un élément essentiel du test d'évolutivité.

Chapitre 5. Disponibilité : présentation générale

Haute disponibilité

Grâce à sa haute disponibilité, WebSphere eXtreme Scale garantit une grande fiabilité dans la redondance des données et la détection des échec.

WebSphere eXtreme Scale organise des grilles de machines virtuelles Java en une arborescence fédérée de manière souple : le service de catalogue se trouve à la racine et les groupes centraux contenant les conteneurs représentent les ramifications. Pour plus d'informations, voir «Architecture de la mise en cache : mappes, conteneurs, clients et catalogues», à la page 11.

Chaque groupe central est automatiquement créé par le service de catalogue en groupes d'environ 20 serveurs. Les membres de ce groupe assurent la surveillance de la santé des autres membres. Chaque groupe central choisit un membre qui aura la responsabilité de communiquer les informations relatives au groupe au service de catalogue. Lorsque la taille du groupe central est limitée, la surveillance de la santé et l'évolutivité de l'environnement s'améliorent.

Remarque : Dans un environnement WebSphere Application Server, dans lequel la taille du groupe central peut être modifiée, eXtreme Scale ne prend pas en charge plus de 50 membres par groupe central.

Echecs

Il existe plusieurs types d'échecs des processus. Un processus peut échouer car la limite des ressources (par exemple la taille maximale des segments de mémoire) a été atteinte ou parce qu'une logique de contrôle de processus a mis fin à un processus. Ceci risque alors d'entraîner un échec du système d'exploitation et la perte des processus en cours d'exécution. Moins fréquemment, une défaillance du matériel, par exemple de la carte d'interface réseau, peut se produire : le système d'exploitation est alors déconnecté du réseau. D'autres échecs peuvent survenir et entraîner l'indisponibilité du processus. Dans ce contexte, les échecs peuvent être classés en deux types : échec de processus et perte de connectivité.

Echec de processus

WebSphere eXtreme Scale réagit très rapidement à un échec de processus. Lorsqu'un processus échoue, le système d'exploitation est responsable du nettoyage des ressources utilisées par celui-ci. Ce nettoyage comprend l'allocation de port et la connectivité. Lorsqu'un processus échoue, un signal est envoyé via les connexions utilisées par ce processus pour fermer celles-ci. Ce signal permet aux autres processus connectés au processus ayant échoué de détecter instantanément cet échec.

Perte de connectivité

Une perte de connectivité se produit lorsque le système d'exploitation est déconnecté. Il ne parvient alors plus à envoyer des signaux à d'autres processus. Plusieurs raisons peuvent expliquer la perte de connectivité. Elles peuvent être classées en deux catégories : échec de l'hôte ou îlotage.

Echec de l'hôte

Si la prise d'alimentation est débranchée, la machine cesse immédiatement de fonctionner.

Ilotage

Ce scénario constitue l'échec le plus complexe à gérer par le logiciel car le processus est censé être indisponible, alors qu'il ne l'est pas. Plus simplement, un serveur ou un autre processus semble avoir échoué alors qu'il s'exécute correctement.

Echec d'un conteneur eXtreme Scale

L'échec d'un conteneur est généralement identifié par des conteneurs homologues via le mécanisme du groupe central. Lorsqu'un conteneur ou un jeu de conteneurs échoue, le service de catalogue migre les fragments hébergés par ce ou ces conteneurs. Le service de catalogue commence par rechercher une réplique synchrone avant de procéder à la migration vers une réplique asynchrone. Une fois les fragments primaires migrés vers les nouveaux conteneurs, le service de catalogue recherche de nouveaux conteneurs hôtes pour les fragments répliques manquants.

Remarque : Ilotage de conteneurs - Le service de catalogue fait migrer les fragments de certains conteneurs lorsque ceux-ci s'avèrent indisponibles. S'ils redeviennent disponibles, le service de catalogue considèrent que des fragments peuvent à nouveau y être positionnés, comme dans le flux de démarrage normal.

Temps d'attente de détection des basculements

Les échecs peuvent être classés en deux catégories : les échecs liés aux logiciels et les échecs liés au matériel. Les échecs liés aux logiciels se produisent généralement lorsqu'un processus échoue. Ils sont détectés par le système d'exploitation, qui parvient très rapidement à récupérer les ressources utilisées, par exemple les sockets réseau. Cette détection se fait en général en moins d'une seconde. La détection des échecs liés au matériel se fait à l'aide de la fonction d'optimisation du signal de présence par défaut. Elle peut prendre jusqu'à 200 secondes. Ces échecs peuvent prendre la forme suivante : panne d'une machine physique, déconnexion d'un câble réseau ou échec du système d'exploitation. eXtreme Scale se fie donc au signal de présence pour détecter les échecs liés au matériel qui peuvent être configurés. Pour plus de détails sur les possibilités de réduction du temps nécessaire à la détection d'un échec lié au matériel, consultez la section «Types de détection de reprise en ligne», à la page 117.

Echec du service de catalogue

La grille du service de catalogue étant aussi une grille eXtreme Scale, elle utilise le mécanisme de regroupement central de la même manière que le processus d'échec des conteneurs, à cette différence près : le domaine de services de catalogue utilise une sélection d'homologue pour définir le fragment primaire plutôt que l'algorithme de service de catalogue utilisé pour les conteneurs.

Il est à noter que le service de positionnement et le service de regroupement central sont des services Un sur N. Un service Un sur N ne s'exécute que sur un seul membre du groupe haute disponibilité. Le service de localisation et l'administration s'exécutent, quant à eux, sur tous les membres de ce groupe. Le

service de positionnement et le service de regroupement central sont des singletons car ils sont responsables de l'agencement du système. Le service de localisation et d'administration sont des services en lecture seule qui existent partout à des fins d'évolutivité.

Le service de catalogue utilise la réplication pour garantir sa tolérance aux erreurs. Si un processus de service de catalogue échoue, il doit redémarrer pour rétablir le niveau de disponibilité du système. Si tous les processus hébergeant le service de catalogue échouent, des données critiques sont perdues. Vous devez alors impérativement redémarrer tous les conteneurs. Comme le service de catalogue peut s'exécuter sur plusieurs processus, cet échec est improbable. Toutefois, si vous exécutez tous les processus sur un même sous-système de stockage, sur un même châssis lame ou à partir d'un même commutateur réseau, un échec est très probable. Essayez de supprimer les modes d'échecs les plus communs des sous-systèmes de stockage qui hébergent le service de catalogue pour limiter les risques d'échec.

Echec de plusieurs conteneurs

Un fragment réplique n'est jamais placé dans le même processus que le fragment primaire, car en cas de perte du processus, les deux fragments seraient perdus. La règle de déploiement définit un attribut booléen de mode de développement utilisé par le service de catalogue pour déterminer si une réplique doit être placée sur la même machine que le fragment primaire. Dans un environnement de développement ne comprenant qu'une machine, vous pouvez prévoir deux conteneurs afin de procéder à des fragments répliques. Dans un environnement de production, l'utilisation d'une seule machine est cependant insuffisante car une éventuelle perte de cette machine hôte entraînerait la perte des deux conteneurs. Pour passer d'un environnement de développement comptant une seule machine à un environnement de production comprenant plusieurs machines, désactivez le mode de développement dans le fichier de configuration de la règle de déploiement.

Disponibilité grâce à la réplication

La réplication fournit la tolérance aux pannes et augmente les performances d'une topologie eXtreme Scale répartie.

La réplication est activée par l'association de mappes de sauvegarde à un groupe de mappes.

Un groupe de mappes (MapSet) est une collection de mappes qui sont catégorisées par une partition-key. Cette partition-key est dérivée de la clé des mappes individuelles par une opération consistant à prendre son hachage modulo le nombre de partitions. Ainsi, si un groupe de mappes au sein du MapSet a une partition-key X, ces mappes seront stockées dans une partition X correspondante dans la grille, toutes les mappes seront stockées dans la partition Y, et ainsi de suite. Egalement, les données contenues dans les mappes sont répliquées en fonction des règles définies dans le MapSet, règles qui ne sont utilisées que pour les topologies eXtreme Scale réparties (l'on n'en a pas besoin pour des instances locales).

Voir «Partitionnement», à la page 93 pour plus de détails.

Aux MapSets sont attribués le nombre de partitions qu'ils auront ainsi qu'une règle de réplication. La configuration de la réplication d'un MapSet consiste simplement

à identifier le nombre de fragments synchrones et asynchrones que doit avoir le MapSet en plus du fragment primaire. Par exemple, s'il doit y avoir un fragment réplique synchrone et un fragment réplique asynchrone, toutes les mappes de sauvegarde attribuées au MapSet auront chacune un fragment réplique automatiquement réparti au sein de l'ensemble des conteneurs disponibles pour eXtreme Scale. La configuration de la réplication peut également permettre aux clients de lire les données depuis les serveurs répliqués de manière synchrone. Cela peut étaler la charge des demandes de lecture sur d'autres serveurs de la grille eXtreme Scale. La réplication a un impact sur le modèle de modèle de programmation lorsqu'on précharge les mappes de sauvegarde.

Nous allons rentrer dans le détail des diverses options de configuration :

Préchargement des mappes

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 répliques. 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.

Vous trouverez dans le *Guide de programmation* les explications sur la manière d'écrire un exemple sur la manière d'implémenter un loader pour un eXtreme Scale répliqué.

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 répliques 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 10. 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.

Tableau 10. Valeur du statut et réponse (suite)

Valeur de statut retournée	Réponse d'eXtreme Scale
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 répliques. 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 répliques 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 répliques 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 11. 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
Envoi des modifications vers les fragments répliques 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 répliques 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 12. 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 répliques, 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 reprise par basculement est terminée.

Equilibrage de la charge entre les fragments répliques

eXtreme Scale, sauf configuration différente, envoie au serveur primaire toutes les demandes de lecture et d'écriture concernant un groupe de réplication donné. Ce serveur primaire doit servir toutes les demandes émanant des clients. Vous voudrez peut-être autoriser l'envoi des demandes de lecture aux répliques du fragment primaire. L'envoi des demandes de lecture aux fragments répliques permet à la charge de ces demandes d'être partagées par plusieurs machines virtuelles Java. Cela dit, il faut savoir que l'utilisation des fragments répliques pour les demandes de lecture peut donner des réponses incohérentes.

Equilibrer la charge entre les fragments répliques s'utilise en général uniquement lorsque les clients mettent en cache des données qui changent en permanence ou lorsque les clients utilisent le verrouillage pessimiste.

Si les données changent en permanence et qu'elles sont ensuite invalidées dans les caches locaux du client, le fragment primaire devrait constater en résultat un taux relativement élevé de demandes get provenant des clients. De même, en mode de

verrouillage pessimiste, il n'existe aucun cache local, c'est pourquoi toutes les demandes sont envoyées au fragment primaire.

Si les données sont relativement statiques ou si le mode pessimiste n'est pas utilisé, l'envoi des demandes au fragment réplique n'a pas un énorme impact sur les performances. La fréquence des demandes get émanant des clients avec des caches pleins de données n'est pas élevée.

Lors du premier démarrage d'un client, son cache local est vide. Les demandes adressées au cache vide sont transmises au fragment primaire. Au fil du temps, le cache client obtient des données, ce qui fait tomber la charge des demandes. Si un grand nombre de clients démarrent simultanément, la charge peut être importante et la lecture par les fragments répliques peut être un choix approprié pour les performances.

Réplication côté client

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.

Types de détection de reprise en ligne

WebSphere eXtreme Scale peut détecter les échecs de manière fiable.

Signaux de présence

1. Les sockets restent ouverts entre les machines virtuelles Java et si un socket se ferme de manière inattendue, cette fermeture est détectée comme un incident de la machine virtuelle Java homologue. Cette détection intercepte les incidents tels que l'arrêt très rapide d'une machine virtuelle Java. Une telle détection permet généralement une reprise en ligne de moins d'une seconde après ces types d'incident.
2. Les autres types d'incident incluent : un blocage du système d'exploitation, un incident physique du serveur ou une panne réseau. Ces incidents sont détectés par l'intermédiaire des signaux de présence.

Des signaux de présence sont envoyés de manière périodique entre des paires de processus : Si un nombre donné de signaux de présence sont manquants, un incident est supposé. Cette approche détecte les incidents en N*M secondes, N

représentant le nombre de signaux de présence manquants et M, l'intervalle auquel les signaux de présence doivent être définis. Il n'est pas possible de spécifier directement M et N ; à la place, un mécanisme de règle est utilisé pour autoriser une plage de combinaisons M et N testées à utiliser.

Echecs

Il existe plusieurs types d'échecs des processus. Un processus peut échouer car la limite des ressources (par exemple la taille maximale des segments de mémoire) a été atteinte ou parce qu'une logique de contrôle de processus a mis fin à un processus. Ceci risque alors d'entraîner un échec du système d'exploitation et la perte des processus en cours d'exécution. Moins fréquemment, une défaillance du matériel, par exemple de la carte d'interface réseau, peut se produire : le système d'exploitation est alors déconnecté du réseau. Dans ce contexte, les échecs peuvent être classés en deux types : échec de processus et perte de connectivité.

Echec de processus

WebSphere eXtreme Scale réagit très rapidement à un échec de processus. Lorsqu'un processus échoue, le système d'exploitation est responsable du nettoyage des ressources utilisées par celui-ci. Ce nettoyage comprend l'allocation de port et la connectivité. Lorsqu'un processus échoue, un signal est immédiatement envoyé via les connexions utilisées par ce processus pour fermer celles-ci.

Perte de connectivité

Une perte de connectivité se produit lorsque le système d'exploitation est déconnecté. Il ne parvient alors plus à envoyer des signaux à d'autres processus. Plusieurs raisons peuvent expliquer la perte de connectivité. Elles peuvent être classées en deux catégories : échec de l'hôte ou îlotage.

Echec de l'hôte

Si une machine hôte n'est plus alimentée, elle devient instantanément non disponible.

Ilotage

Ce scénario constitue l'échec le plus complexe à gérer par le logiciel car le processus est censé être indisponible, alors qu'il ne l'est pas.

Echec de conteneur

L'échec d'un conteneur est généralement identifié par des conteneurs homologues via le mécanisme du groupe central. Lorsqu'un conteneur ou un jeu de conteneurs échoue, le service de catalogue migre les fragments hébergés par ce ou ces conteneurs. Le service de catalogue commence par rechercher un fragment réplique synchrone avant de procéder à la migration vers un fragment réplique asynchrone. Une fois les fragments primaires migrés vers les nouveaux conteneurs, le service de catalogue recherche de nouveaux conteneurs hôtes pour les fragments répliques manquants.

Remarque : Ilotage de conteneurs - Le service de catalogue fait migrer les fragments de certains conteneurs lorsque ceux-ci s'avèrent indisponibles. S'ils redeviennent disponibles, le service de catalogue considèrent que des fragments peuvent à nouveau y être positionnés, comme dans le flux de démarrage normal.

Temps d'attente de détection des basculements

Les échecs peuvent être classés en deux catégories : les échecs liés aux logiciels et les échecs liés au matériel. Les échecs liés aux logiciels se produisent généralement lorsqu'un processus échoue. Ils sont détectés par le système d'exploitation, qui parvient très rapidement à récupérer les ressources utilisées, par exemple les sockets réseau. Cette détection se fait en général en moins d'une seconde. La détection des échecs liés au matériel se fait à l'aide de la fonction d'optimisation du signal de présence par défaut. Elle peut prendre jusqu'à 200 secondes. Ces échecs peuvent prendre la forme suivante : panne d'une machine physique, déconnexion d'un câble réseau ou échec du système d'exploitation. eXtreme Scale se fie donc au signal de présence pour détecter les échecs liés au matériel qui peuvent être configurés.

Echec de plusieurs conteneurs

Un fragment réplique n'est jamais placé dans le même processus que le fragment primaire, car en cas de perte du processus, les deux fragments seraient perdus. La règle de déploiement définit un attribut utilisé par le service de catalogue pour déterminer si un fragment réplique doit être placé sur la même machine que le fragment primaire. Dans un environnement de développement ne comprenant qu'une machine, vous pouvez prévoir deux conteneurs afin de procéder à des fragments répliques. Dans un environnement de production, l'utilisation d'une seule machine est cependant insuffisante car une éventuelle perte de cette machine hôte entraînerait la perte des deux conteneurs. Pour passer d'un environnement de développement comptant une seule machine à un environnement de production comprenant plusieurs machines, désactivez le mode de développement dans le fichier de configuration de la règle de déploiement.

Echec du service de catalogue

La grille du service de catalogue étant aussi une grille eXtreme Scale, elle utilise le mécanisme de regroupement central de la même manière que le processus d'échec des conteneurs, à cette différence près : le domaine de services de catalogue utilise une sélection d'homologue pour définir le fragment primaire plutôt que l'algorithme de service de catalogue utilisé pour les conteneurs.

Il est à noter que le service de positionnement et le service de regroupement central sont des services Un sur N. Un service Un sur N ne s'exécute que sur un seul membre du groupe haute disponibilité. Le service de localisation et l'administration s'exécutent, quant à eux, sur tous les membres de ce groupe. Le service de positionnement et le service de regroupement central sont des singletons car ils sont responsables de l'agencement du système. Le service de localisation et d'administration sont des services en lecture seule qui existent partout à des fins d'évolutivité.

Le service de catalogue utilise la réplication pour garantir sa tolérance aux erreurs. Si un processus de service de catalogue échoue, il doit redémarrer pour rétablir le niveau de disponibilité du système. Si tous les processus hébergeant le service de catalogue échouent, des données critiques sont perdues. Vous devez alors impérativement redémarrer tous les conteneurs. Comme le service de catalogue peut s'exécuter sur plusieurs processus, cet échec est improbable. Toutefois, si vous exécutez tous les processus sur un même sous-système de stockage, sur un même chassis lame ou à partir d'un même commutateur réseau, un échec est très probable. Essayez de supprimer les modes d'échecs les plus connus des

sous-systèmes de stockage qui hébergent le service de catalogue pour limiter les risques d'échec.

Tableau 13. Récapitulatif de la reconnaissance d'échec et de la reprise en ligne

Type de perte	Mécanisme de reconnaissance (détection)	Méthode de récupération
Perte de processus	E-S	Redémarrage
Perte de serveur	Signal de présence	Redémarrage
Indisponibilité du réseau	Signal de présence	Rétablissement du réseau et de la connexion
Blocage côté serveur	Signal de présence	Arrêt et redémarrage du serveur
Serveur occupé	Signal de présence	Attendre que le serveur soit disponible

Service de catalogue à haute disponibilité

Le domaine de services de catalogue est la grille des serveurs de catalogues que vous utilisez. Il contient les informations relatives à la topologie de tous les conteneurs de votre environnement eXtreme Scale. Le service de catalogue contrôle les fonctions d'équilibrage et de routage pour tous les clients. Pour déployer eXtreme Scale en tant qu'espace de traitement de la base de données en mémoire, vous devez regrouper le service de catalogue en un cluster de domaine de services de catalogue afin de garantir la haute disponibilité.

Composants du domaine de services de catalogue

Lorsque plusieurs serveurs de catalogues démarrent, l'un d'eux est choisi comme serveur maître acceptant les signaux de présence IIOP (Internet Inter-ORB Protocol) et gérant les modifications des données du système consécutives aux modifications apportées aux services de catalogue ou aux conteneurs.

Lorsque des clients contactent l'un des serveurs de catalogues, la table de routage du domaine de services de catalogue est propagée vers ces clients via le contexte de service CORBA (Common Object Request Broker Architecture).

Configurez au moins trois serveurs de catalogues. Si votre configuration contient des zones, vous pouvez configurer un serveur de catalogues par zone.

Lorsqu'un serveur conteneur eXtreme Scale contacte l'un des serveurs de catalogues, la table de routage du domaine de services de catalogue est également propagée au serveur conteneur eXtreme Scale via le contexte de service CORBA. En outre, si le serveur contacté n'est pas le serveur maître, la demande est automatiquement redirigée vers le serveur maître actuel et la table de routage du serveur de catalogues est mise à jour.

Remarque : La grille des serveurs de catalogues et la grille des serveurs conteneurs sont très différentes. La première, le domaine de services de catalogue, permet de garantir la haute disponibilité de vos données système. La seconde, la grille des conteneurs, permet d'assurer la haute disponibilité et l'extensibilité de vos données ainsi que la gestion des charges de travail. De ce fait, il existe deux différentes tables de routage : la table de routage du domaine de services de catalogue et celle des fragments de la grille des serveurs.

Les responsabilités du catalogue se répartissent en plusieurs sortes de services. Le gestionnaire du groupe central gère le regroupement homologue en vue de la surveillance de la santé, le service de positionnement prend en charge les allocations, le service d'administration fournit l'accès à l'administration et le service de localisation gère les localités.

Déploiement du domaine de services de catalogue

Gestionnaire du groupe central

Le service de catalogue utilise le gestionnaire de haute disponibilité (gestionnaire HA) pour regrouper des processus en vue de la surveillance de la disponibilité. Chaque regroupement est un groupe central. Avec eXtreme Scale, le gestionnaire du groupe central regroupe les processus de manière dynamique. Ces processus doivent être de petite taille afin de permettre l'évolutivité. Chaque groupe central choisit un responsable qui sera chargé d'envoyer un statut au gestionnaire du groupe central en cas de défaillance de certains membre. Le même mécanisme de statut est utilisé pour identifier une éventuelle défaillance de tous les membres d'un groupe, qui pourrait entraîner un échec de la communication avec le responsable.

Le gestionnaire du groupe central est un service entièrement automatique responsable de l'organisation des conteneurs en petits groupes de serveurs qui sont alors automatiquement fédérés de manière souple pour constituer une grille d'objets. Lorsqu'un conteneur contacte le service de catalogue pour la première fois, il attend d'être affecté à un nouveau groupe ou à un groupe existant. Un déploiement eXtreme Scale consiste en plusieurs groupes de ce type et ce regroupement est une tâche d'activation essentielle pour l'évolutivité. Chaque groupe est un groupe de machines virtuelles Java utilisant les signaux de présence pour surveiller la disponibilité des autres groupes. L'un des membres de ce groupe est choisi comme responsable. Il est chargé de relayer les informations relatives à la disponibilité au service de catalogue afin de lui permettre de réagir aux défaillances en procédant à des réallocations et à des réacheminements.

Service de positionnement

Le service de catalogue gère le positionnement des fragments dans les conteneurs disponibles. Le service de positionnement est responsable du maintien de l'équilibre des ressources entre les différentes ressources physiques. Il prend également en charge l'allocation des fragments à leur conteneur hôte. Il s'exécute en tant que service Un sur N choisi dans la grille de données afin qu'il ne s'exécute qu'une instance et une seule de ce service. Si cette instance échoue, un autre processus est choisi et il prend le relais. Pour garantir la redondance, l'état du service de catalogue est répliqué sur tous les serveurs qui hébergent le service de catalogue.

Administration

Le service de catalogue constitue le point d'entrée logique de l'administration du système. Le service de catalogue héberge un bean géré (MBean) et il fournit des URL JMX (Java Management Extensions) pour les serveurs gérés par le service.

Service de localisation

Le service de localisation sert de point tactile pour les clients qui recherchent les conteneurs hébergeant l'application qu'ils recherchent, ainsi que pour les

conteneurs qui enregistrent les applications hébergées auprès du service de positionnement. Il s'exécute sur tous les membres de la grille, permettant ainsi de supprimer cette fonction.

Le service de catalogue héberge une logique qui est généralement inactive lorsque l'état est stabilisé. Il a donc très peu d'incidence sur l'évolutivité. Il permet de gérer des centaines de conteneurs devenant disponibles simultanément. Configurez le service de catalogue dans une grille à des fins de disponibilité.

Planification

Une fois que le domaine de services de catalogue a démarré, ses membres se lient entre eux. La topologie du domaine de services de catalogue doit faire l'objet d'une planification prudente et méticuleuse, car il sera impossible de modifier la configuration du domaine au moment de l'exécution. Etendez la grille de manière aussi diversifiée que possible pour éviter d'éventuelles erreurs.

Démarrage d'un domaine de services de catalogue

Pour en savoir plus sur la création d'un domaine de services de catalogue, voir dans le *Guide d'administration* les explications sur le démarrage d'un domaine de services de catalogue.

Connexion à un domaine autonome de services de catalogue des conteneurs eXtreme Scale intégrés à WebSphere Application Server

Vous pouvez configurer des conteneurs eXtreme Scale qui sont imbriqués dans un environnement WebSphere Application Server pour qu'ils se connectent à un domaine autonome de services de catalogue.

-  (déprécié) Dans les précédentes versions, la connexion de services de catalogue à un domaine de services de catalogue s'effectuait par la création d'une propriété personnalisée. Cette propriété est toujours utilisable, mais son utilisation n'est pas encouragée car elle est considérée comme déprécié. Pour en savoir plus sur cette propriété personnalisée, voir dans le *Guide d'administration* les explications sur le démarrage du processus de service de catalogue dans un environnement WebSphere Application Server..

  Remarque : Collision des noms de serveur : cette propriété étant utilisée pour démarrer le serveur de catalogues eXtreme Scale et pour s'y connecter, un serveur de catalogues ne doit pas porter le même nom qu'un serveur WebSphere Application Server.

Pour plus d'informations, consultez la rubrique «Quorums de serveurs de catalogue».

Quorums de serveurs de catalogue

Le quorum est le nombre minimum de serveurs de catalogue nécessaires à l'exécution des opérations de positionnement pour la grille. Le nombre minimum est l'ensemble complet des serveurs de catalogue à moins que le quorum n'ait été redéfini.

Termes importants à connaître

Voici, avec leur définition, une liste des termes utilisés à propos des quorums dans WebSphere eXtreme Scale.

- **Microcoupure** :: une microcoupure est une perte provisoire de connectivité entre un ou plusieurs serveurs.
- **Interruption totale** :: une interruption totale est une perte permanente de connectivité entre un ou plusieurs serveurs.
- **Centre de données** : : un centre de données est un groupe géographique de serveurs, connectés en générale par un réseau local (LAN).
- **Zone**: une zone est une option de configuration servant à regrouper des serveurs ayant en commun une caractéristique physique particulière. Exemples de zones regroupant des serveurs : centre de données (voir plus haut), réseau de zone, bâtiment, étage d'un bâtiment, etc.
- **Signal de présence** : : mécanisme permettant de déterminer si une machine virtuelle Java donnée est ou non en cours d'exécution.

Topologie

Nous allons expliquer le fonctionnement de WebSphere eXtreme Scale sur un réseau comprenant des composant non fiables (par exemple, un réseau constitué de plusieurs centres de données épars).

Espace des adresses IP

WebSphere eXtreme Scale nécessite un réseau dans lequel tout élément adressable puisse se connecter sans entraves à n'importe quel autre élément adressable du réseau. En d'autres termes, WebSphere eXtreme Scale requiert un espace de noms non hiérarchisé d'adresses IP. Il requiert également que tous les pare-feu autorisent la totalité du trafic à circuler entre les adresses IP et les ports utilisés par les machines virtuelles Java hébergeant des éléments de WebSphere eXtreme Scale.

Réseaux locaux connectés

A chaque réseau local est attribué un identificateur de zone pour les besoins de WebSphere eXtreme Scale. WebSphere eXtreme Scale tente agressivement de détecter les signaux de présence des machines virtuelles Java d'une même zone. Il suffit que l'un de ces signaux manque pour que se déclenche un événement de basculement si le service de catalogue a le quorum.

Domaine de services de catalogue et serveurs conteneurs

Un domaine de services de catalogue est une collection de machines virtuelles Java semblables. Un domaine de services de catalogue est une grille composée de serveurs de catalogue et il est de taille fixe. Or, le nombre des serveurs conteneurs, lui, est dynamique. Des serveurs conteneurs peuvent être ajoutés ou supprimés à la demande. Dans une configuration de trois centres de données, WebSphere eXtreme Scale nécessite une machine virtuelle Java de service de catalogue par centre de données.

Le domaine de services de catalogue utilise un mécanisme de quorum complet. Du fait de ce mécanisme de quorum complet, les membres de la grille doivent tous être d'accord sur n'importe quelle action à entreprendre.

Les machines virtuelles Java des serveurs conteneurs sont marquées avec un identificateur de zone. La grille des machines virtuelles Java de conteneurs est automatiquement fractionnée en petits groupes centraux de machines virtuelles

Java. Un groupe central ne comprendra que des machines virtuelles Java de la même zone. Des machines virtuelles Java de zones différentes ne feront jamais partie du même groupe central.

Un groupe central essaiera agressivement de détecter la défaillance de l'une de ses machines virtuelles Java. Les machines virtuelles Java de conteneurs d'un groupe central ne doivent jamais s'étendre sur plusieurs réseaux locaux interconnectés comme dans un réseau étendu. Cela signifie qu'un groupe central ne peut avoir dans la même zone de conteneurs s'exécutant dans des centres de données différents.

Cycle de vie des serveurs

Démarrage des serveurs de catalogue

Les serveurs de catalogue sont démarrés à l'aide de la commande startOgServer. Les quorums sont désactivés par défaut. Pour les activer, deux possibilités : passer l'indicateur `-quorum enabled` dans la commande startOgServer ou ajouter la propriété `enableQuorum=true` dans le fichier des propriétés. Tous les serveurs de catalogue doivent avoir le même quorum.

```
# bin/startOgServer cat0 –serverProps objectGridServer.properties
```

```
objectGridServer.properties file
catalogClusterEndPoints=cat0:cat0.domain.com:6600:6601,
cat1:cat1.domain.com:6600:6601
catalogServiceEndPoints= cat0.domain.com:2809, cat1.domain.com:2809
enableQuorum=true
```

Démarrage des serveurs conteneurs

Les serveurs conteneurs sont démarrés à l'aide de la commande startOgServer. Dans le cas d'une grille de plusieurs centres de données, les serveurs doivent utiliser l'étiquette zone pour identifier le centre de données dans lequel ils résident. Définir la zone sur les serveurs d'une grille permet à WebSphere eXtreme Scale de ne surveiller la bonne santé que des serveurs du centre de données, en réduisant le trafic entre les centres de données.

```
# bin/startOgServer gridA0 –serverProps objectGridServer.properties –
objectgridfile xml/objectgrid.xml –deploymentpolicyfile xml/
deploymentpolicy.xml
```

```
objectGridServer.properties file
catalogServiceEndPoints= cat0.domain.com:2809, cat1.domain.com:2809
zoneName=ZoneA
```

Arrêt des serveurs d'une grille

Les serveurs d'une grille sont arrêtés à l'aide de la commande stopOgServer. Lors de l'arrêt d'un centre de données entier pour des raisons de maintenance, passez la liste de tous les serveurs appartenant à cette zone. Cela permettra une transition d'état nette entre la zone en cours d'arrêt et la ou les zones qui continuent de fonctionner.

```
# bin/stopOgServer gridA0,gridA1,gridA2 –catalogServiceEndPoints
cat0.domain.com:2809,cat1.domain.com:2809
```

Détection des défaillances

WebSphere eXtreme Scale détecte les morts de processus via des événements de fermeture anormale de sockets. Le service de catalogue est immédiatement informé lorsqu'un processus prend fin. Les interruptions totales sont détectées via l'absence des signaux de présence. WebSphere eXtreme Scale se protège lui-même contre les microcoupures entre les centres de données en utilisant une implémentation de quorums.

Implémentation de la détection des signaux de présence

Nous allons expliquer comment la détection de l'activité est implémentée dans WebSphere eXtreme Scale.

Détection des signaux de présence des membres de groupes centraux

Le service de catalogue place des machines virtuelles Java de conteneurs dans des groupes centraux dont la taille est limitée. Pour détecter la défaillance de l'un de ses membres, un groupe central pourra s'y prendre de deux manières. Si le socket d'une machine virtuelle Java est fermé, cette machine virtuelle Java sera considérée comme morte. Par ailleurs, chaque membre émet un signal de présence sur ces sockets à une fréquence qui est déterminée par la configuration. Si une machine virtuelle Java ne réagit pas à ces signaux dans le délai maximum prévu par la configuration, elle sera considérée comme morte.

Un membre du groupe central est toujours désigné comme leader. Le leader du groupe central (CGL) est chargé d'informer régulièrement le service de catalogue que le groupe central fonctionne et de signaler toutes les modifications intervenues au sein du service de catalogue. La défaillance d'une machine virtuelle Java ou l'ajout au groupe d'une nouvelle machine sont considérés comme des modifications de groupe central.

Si le leader du groupe central ne parvient pas à contacter l'un des membres du domaine de services de catalogue, il continuera de réessayer.

Détection des signaux de présence du domaine de services de catalogue

Le domaine de services de catalogue ressemble à un groupe central privé dont les membres sont statiques avec un mécanisme de quorum. La détection des défaillances s'effectue de la même manière que pour un groupe central normal. La différence tient à la modification de son comportement puisqu'il inclut une logique de quorum. Le service de catalogue utilise également une configuration moins agressive pour la détection des signaux de présence.

Détection des signaux de présence des groupes centraux

Le service de catalogue a besoin de savoir quand des serveurs conteneurs sont défaillants. Chaque groupe central est chargé de déterminer les défaillances des machines virtuelles Java conteneurs et de les signaler au service de catalogue par l'intermédiaire du leader du groupe. La défaillance complète de la totalité des membres d'un groupe central est une éventualité qui peut arriver. Si la totalité du groupe central est défaillant, c'est au service de catalogue de détecter cette perte.

Si le service de catalogue marque une machine virtuelle Java conteneur comme défaillante et que le conteneur est ultérieurement signalé comme en fonctionnement, la machine virtuelle Java conteneur recevra l'ordre d'arrêter les serveurs conteneurs WebSphere eXtreme Scale. Une machine virtuelle Java dans cet état ne sera pas visible dans les requêtes xsadmin. Des messages dans les journaux

de la machine virtuelle Java conteneur indiqueront que cette machine est tombée en panne. Vous devrez la redémarrer manuellement.

Si une perte de quorum s'est produite, la détection des signaux de présence sera suspendue en attendant le rétablissement du quorum.

Comportement du service de catalogue en matière de quorum

Normalement, les membres du service de catalogue disposent d'une connectivité pleine et entière. Le domaine de services de catalogue est un ensemble statique de machines virtuelles Java. WebSphere eXtreme Scale s'attend à ce que tous les membres du service de catalogue soient en permanence en ligne. Le service de catalogue ne répondra aux événements de conteneur que tant qu'il aura le quorum.

S'il perd le quorum, le service de catalogue attendra le rétablissement du quorum. Tant qu'il n'a pas le quorum, le service de catalogue ignorera les événements provenant des serveurs conteneurs. Ceux-ci retenteront pendant ce temps toutes les demandes rejetées par le serveur de catalogue puisque WebSphere eXtreme Scale eXtreme Scale s'attend à ce que le quorum soit rétabli.

Le message suivant indique que le quorum a été perdu. Recherchez la présence éventuelle de ce message dans les journaux de vos services de catalogue.

```
CWOBJ1254W: Le service de catalogue attend un quorum.
```

WebSphere eXtreme Scale s'attend à perdre le quorum pour les raisons suivantes :

- défaillance d'un membre machine virtuelle Java du service de catalogue
- microcoupure réseau
- perte de centre de données

Arrêter une instance de serveur de catalogue à l'aide de la commande `stopOgServer` ne provoque pas de perte de quorum car le système sait que l'instance a été arrêtée, ce qui n'est pas la même chose qu'une défaillance de machine virtuelle Java ou une microcoupure.

Perte de quorum due à une défaillance de machine virtuelle Java

La défaillance d'un serveur de catalogue provoquera la perte du quorum. Dans ce cas, le quorum doit être redéfini le plus vite possible. Le service de catalogue défaillant ne peut rejoindre la grille tant que le quorum n'a pas été redéfini.

Perte de quorum due à une microcoupure réseau

WebSphere eXtreme Scale est conçu pour intégrer la possibilité de microcoupures. Une microcoupure est une perte provisoire de connectivité entre des centres de données. Cette situation est transitoire par nature et en principe la situation est rétablie en quelques secondes ou en quelques minutes. WebSphere eXtreme Scale essaie de maintenir le bon fonctionnement pendant la microcoupure, celle-ci est donc considérée comme une simple défaillance. La défaillance est censée être réglée et le fonctionnement normal reprend sans intervention de WebSphere eXtreme Scale.

Une microcoupure qui s'éternise ne pourra être requalifiée en interruption totale que par une intervention d'utilisateur. La redéfinition du quorum sur l'un des côtés de la microcoupure est en effet nécessaire pour que l'événement soit classé comme interruption totale.

Cycle entre les services de catalogue

Si un serveur de catalogue est arrêté à l'aide de stopOgServer, le quorum diminue d'un serveur. Cela signifie que les serveurs restants ont toujours le quorum. Redémarrer le serveur de catalogue ramène le quorum au nombre précédent.

Conséquences de la perte de quorum

S'il arrive à une machine virtuelle Java conteneur de tomber en panne pendant la perte du quorum, la reprise n'aura pas lieu tant que l'on ne sera pas sorti de la microcoupure ou tant que l'utilisateur n'aura pas redéfini le quorum. WebSphere eXtreme Scale considère la perte de quorum et la défaillance d'un conteneur comme une double défaillance, ce qui est un événement rare. Cela signifie que des applications peuvent perdre l'accès en écriture aux données qui étaient stockées sur la machine virtuelle Java défaillante jusqu'à ce que le quorum soit restauré, et ce n'est qu'alors que la reprise normale aura lieu.

De même, toute tentative de démarrage d'un conteneur pendant une perte de quorum est vouée à l'échec : le conteneur ne démarrera pas.

La connectivité clients complète est autorisée pendant la perte de quorum. S'il ne se produit aucune défaillance de conteneur ou de problèmes de connectivité pendant la perte du quorum, les clients pourront toujours interagir complètement avec les serveurs conteneurs.

Si une microcoupure se produit, certains clients risquent de pas pouvoir accéder aux copies primaires ou répliques des données jusqu'à la fin de la microcoupure.

Il est possible de démarrer de nouveaux clients, car il doit y avoir une machine virtuelle Java service de catalogue dans chaque centre de données, donc au moins une machine virtuelle service de catalogue sera accessible par le client même pendant une microcoupure.

Rétablissement du quorum

Si le quorum est perdu pour une quelconque raison, lorsqu'il est rétabli, un protocole de récupération est exécuté. Lorsque la perte du quorum se produit, toute vérification d'activité des groupes centraux est suspendue et les rapports signalant des défaillances sont ignorés. Une fois que le quorum est rétabli, le service de catalogue procède à une vérification de l'activité de tous les groupes centraux pour déterminer immédiatement leur composition. C'est à ce stade que seront récupérés tous les fragments précédemment hébergés sur des machines virtuelles Java signalées comme défaillantes. En cas de perte de fragments primaires, les fragments répliques survivants passent au statut de fragments primaires. En cas de perte de fragments répliques, des fragments répliques supplémentaires seront créées à partir des survivantes.

Redéfinir le quorum

Cela n'est à utiliser qu'en cas de défaillance d'un centre de données. Le quorum perdu suite à la défaillance d'une machine virtuelle Java service de catalogue ou à

une microcoupure du réseau doit se rétablir automatiquement une fois la machine virtuelle Java redémarrée ou la microcoupure terminée.

Les administrateurs sont les seuls à être informés d'une défaillance du centre de données. WebSphere eXtreme Scale traite de manière semblable les microcoupures et les interruptions totales. Vous devez informer l'environnement eXtreme Scale de ces défaillances à l'aide de la commande xsadmin permettant de redéfinir le quorum. Le service de catalogue sera ainsi avisé de supposer que le quorum est atteint avec le nombre actuel de membres et la reprise complète aura lieu. En émettant une commande de redéfinition du quorum, vous garantissez que les machines virtuelles Java du centre de données défaillant sont réellement en panne et qu'elles ne reprendront pas leur activité.

La liste qui suit envisage quelques scénarios de redéfinition de quorum. Supposons que nous ayons trois serveurs de catalogue : A, B et C.

- Microcoupure : supposons que, suite à une microcoupure, C soit provisoirement isolé. Le service de catalogue perdra le quorum et attendra la fin de la microcoupure. A ce moment-là, C réintégrera le domaine de services de catalogue et le quorum sera rétabli. Votre application ne percevra aucun problème pendant ce temps.
- Défaillance provisoire : ici, C est défaillant et le service de catalogue perd le quorum. Vous devez donc redéfinir ce dernier. Une fois le quorum rétabli, C pourra être redémarré. C réintégrera le domaine de services de catalogue lorsqu'il redémarrera. L'application ne percevra aucun problème pendant ce temps.
- Défaillance de centre de données : vous vérifiez que le centre de données est réellement défaillant et qu'il est bien isolé sur le réseau. Puis vous lancez la commande xsadmin de redéfinition de quorum. Les deux centres de données survivants procèdent à une reprise complète en remplaçant les fragments qui étaient hébergés dans le centre défaillant. Le service de catalogue s'exécute à présent avec un quorum complet de A et de B. L'application risque de constater des retards ou des exceptions pendant l'intervalle séparant le début de l'interruption totale et la redéfinition du quorum . Une fois ce dernier redéfini, la grille et le fonctionnement normal reprennent.
- Reprise du fonctionnement de centre de données : les centres de données survivants fonctionnent déjà avec un quorum redéfini. Lorsque le centre de données contenant C est redémarré, toutes les machines virtuelles Java du centre doivent être redémarrées. C réintégrera alors le domaine de services de catalogue existant et le quorum reviendra à la situation normale sans intervention de l'utilisateur.
- Défaillance de centre de données et microcoupure : le centre de données contenant C tombe en panne. Le quorum est redéfini et rétabli sur les centres de données restants. Si une microcoupure se produit entre A et B, les règles normales de reprise en cas de microcoupure s'appliquent. Une fois la microcoupure terminée, le quorum est rétabli et la récupération nécessaire après la perte du quorum se produit.

Comportement des conteneurs

Nous allons décrire comment se comporte les machines virtuelles Java des serveurs conteneurs pendant une perte et une récupération de quorum.

Les conteneurs hébergent un ou plusieurs fragments. Les fragments sont soit des fragments primaires, soit des fragments répliques pour une partition spécifique. Le service de catalogue affecte des fragments à un conteneur et le conteneur honorera

cette affectation jusqu'à ce que nouvelles instructions arrivent du service de catalogue. Cela signifie que si, dans un conteneur, un fragment primaire ne parvient pas à communiquer avec un fragment réplique en raison d'une microcoupure, il continuera à réessayer jusqu'à ce qu'il reçoive de nouvelles instructions du service de catalogue.

Si une microcoupure se produit et qu'un fragment primaire perd la communication avec sa réplique, il continuera ses tentatives de connexion jusqu'à ce que le service de catalogue lui envoie de nouvelles instructions.

Comportement des fragments répliques synchrones

Pendant que la connexion est interrompue, le fragment primaire peut accepter de nouvelles transactions tant que le nombre de fragments répliques en ligne est au moins égal à la valeur définie pour la propriété minsync du groupe de mappes. Si de nouvelles transactions sont traitées sur le fragment primaire pendant que la liaison avec le fragment réplique synchrone est interrompue, le fragment réplique synchrone sera effacé et resynchronisé avec l'état actuel du fragment primaire lorsque la liaison sera rétablie.

La réplication synchrone est fortement découragée entre les centres de données ou sur les liaisons de type réseau étendu.

Comportement des fragments répliques asynchrones

Pendant que la connexion est interrompue, le fragment primaire peut accepter de nouvelles transactions. Le fragment primaire mettra les modifications en mémoire tampon jusqu'à une certaine limite. Si la connexion au fragment réplique est rétablie avant que cette limite ne soit atteinte, le fragment réplique sera actualisé avec les modifications mises en mémoire tampon. Si la limite est atteinte, le fragment primaire détruit la liste en tampon et, lorsqu'il se reconnecte, le fragment réplique est effacé et resynchronisé.

Comportement des clients

Les clients sont toujours en mesure de se connecter au serveur de catalogue pour s'amorcer sur la grille que le domaine de services de catalogue ait ou non le quorum. Le client essaiera de se connecter à n'importe quelle instance de serveur de catalogue pour obtenir une table de routage afin d'interagir avec la grille. La connectivité réseau peut empêcher le client d'interagir avec certaines partitions en raison de la configuration du réseau. Le client peut se connecter aux répliques locales des données distantes s'il a été configuré pour cela. Les clients ne seront pas en mesure d'actualiser des données si la partition primaire de ces données n'est pas disponible.

Commandes xsadmin de quorum

Nous allons passer en revue les commandes xsadmin utiles pour les quorums.

Interroger le statut du quorum

Il est possible d'interroger avec la commande xsadmin le statut du quorum d'une instance de serveur de catalogue.

```
xsadmin –ch cathost –p 1099 –quorumstatus
```

Cinq résultats sont possibles.

- Le quorum est désactivé : les serveurs de catalogue s'exécutent en mode quorum-disabled. L'on est en mode développement ou en mode centre de données unique. Ce n'est pas recommandé pour les configurations impliquant plusieurs centres de données.
- Le quorum est activé et le serveur de catalogue a le quorum : le quorum est activé et le système fonctionne normalement.
- Le quorum est activé et le serveur de catalogue attend le quorum : le quorum est activé et il a été perdu.
- Le quorum est activé et il est redéfini : le quorum est activé et il a été redéfini.
- Le quorum est proscrit : lorsqu'une microcoupure se produit, le service de catalogue est scindé en deux partitions, A et B. Le serveur de catalogue A a un quorum qui a été redéfini. La partition réseau se résout et le serveur dans la partition B est proscrit, nécessitant un redémarrage des machines virtuelles Java. Cela se produit également si la machine virtuelle Java du catalogue redémarre pendant la microcoupure et que cette dernière se termine.

Redéfinir le quorum

La commande xsadmin peut servir à redéfinir un quorum. Toutes les instances survivantes de serveurs de catalogue sont utilisables. Tous les survivants sont notifiés lorsque l'un d'entre eux reçoit l'injonction de redéfinir le quorum. La syntaxe est la suivante.

```
xsadmin –ch cathost –p 1099 –overridequorum
```

Commandes de diagnostics

- Statut du quorum : voir la section précédente.
- Liste des groupes centraux : affiche la liste de tous les groupes centraux. Les membres et les leaders des groupes centraux sont affichés.

  ```
  xsadmin –ch cathost –p 1099 –coregroups
  ```
- Arrêt de serveurs : cette commande retire manuellement un serveur de la grille. Ce n'est en principe pas nécessaire puisque les serveurs sont automatiquement retirés lorsqu'ils sont détectés comme défaillants, mais la commande est quand même fournie pour être utilisée sous les directives du support IBM.

  ```
  xsadmin –ch cathost –p 1099 –g Grid –teardown server1,server2,server3
  ```
- Affichage de la table de routage : cette commande affiche la table de routage en cours en simulant une nouvelle connexion client à la grille. Elle valide également la table de routage en confirmant que tous les serveurs conteneurs reconnaissent bien leur rôle dans la table (par exemple, quel type de fragment pour quelle partition).

  ```
  xsadmin –ch cathost –p 1099 –g myGrid -routetable
  ```
- Affichage des fragments non attribués : si certains fragments ne peuvent être placés dans la grille, cette commande permet d'afficher leur liste. Cela ne se produit que lorsque le service de positionnement comporte une contrainte qui empêche le positionnement. Exemple : si, en en mode de production, vous démarrez des machines virtuelles Java sur un seul système, seuls seront placés les fragments primaires. Les fragments répliques ne seront attribués que lorsque des machines virtuelles Java démarreront sur un second système. Le service de positionnement ne place des fragments répliques que sur des machines virtuelles Java dont les adresses IP sont différentes de celles des machines virtuelles Java hébergeant les fragments primaires. Ne pas avoir de machines virtuelles Java dans une zone peut également provoquer la non-attribution de fragments.

```
xsadmin –ch cathost –p 1099 –g myGrid –unassigned
```

- Paramétrer la trace : cette commande définit les paramètres de trace pour toutes les machines virtuelles Java correspondant au filtre spécifié pour la commande xsadmin. Les paramètres de trace ne sont modifiés que jusqu'à ce qu'une autre commande soit utilisée ou que les machines virtuelles Java tombent en panne ou s'arrêtent.

  ```
  xsadmin –ch cathost –p 1099 –g myGrid –fh host1 –settracespec
  ObjectGrid*=event=enabled
  ```

  La trace est activée sur toutes les machines virtuelles Java du système dont le nom d'hôte est spécifiée (host1 ici).
- Vérification de la taille des mappes : la commande mapsizes est utile pour vérifier que la répartition des clés est uniforme sur les fragments présents dans la clé. Si certains conteneurs détiennent un nombre de clés significativement plus important que d'autres, c'est l'indice que la fonction de hachage sur les objets key a une répartition de mauvaise qualité.

  ```
  xsadmin -ch cathost -p 1099 -g myGrid -m myMapSet -mapsizes myMap
  ```

Considérations relatives à la sécurisation des transports

Les centres de données étant normalement déployés dans des sites géographiquement dispersés, les utilisateurs voudront activer la sécurité des transports entre ces centres.

Lire les explications concernant la sécurité des couches de transport dans le *Guide d'administration*.

Répliques et fragments

Avec eXtreme Scale, il est possible de répliquer une base de données en mémoire ou un fragment d'une machine virtuelle Java (JVM) vers une autre. Un fragment représente une partition qui est placée dans un conteneur. Plusieurs fragments représentant différentes partitions peuvent coexister dans un même conteneur. Chaque partition a une instance qui est un fragment primaire et un nombre configurable de fragments répliques. Les fragments répliques sont synchrones ou asynchrones. Les types et le positionnement des fragments répliques est déterminé par eXtreme Scale à l'aide d'une règle de déploiement qui spécifie les nombres minimum et maximum de fragments synchrones et asynchrones.

Types de fragment

La réplication utilise trois types de fragments :

- primaire
- fragment réplique synchrone
- fragment réplique asynchrone

Le fragment primaire reçoit toutes les opérations d'insertion, d'actualisation et de suppression. Le fragment primaire ajoute et supprime des fragments répliques, réplique les données vers les fragments répliques et gère les validations et les annulations des transactions.

Les fragments répliques synchrones maintiennent le même état que le fragment primaire. Lorsqu'un fragment primaire réplique des données vers un fragment réplique synchrone, la transaction n'est validée qu'après avoir été validée dans le fragment réplique synchrone.

Les fragments répliques asynchrones ne sont pas forcément dans le même état que le fragment primaire. Lorsqu'un fragment primaire réplique des données vers un fragment réplique asynchrone, il n'attend pas la validation de ce dernier.

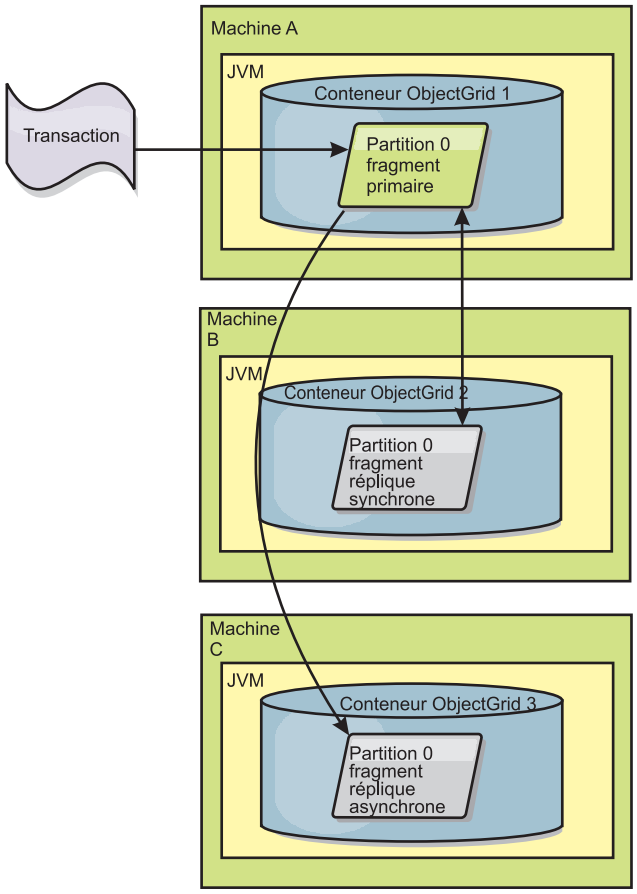


Figure 34. Chemin de la communication entre un fragment primaire et un fragment réplique

Fragments répliques synchrones minimum

Lorsqu'un fragment primaire se prépare à valider des données, il vérifie combien de fragments répliques synchrones ont élu de valider la transaction. Le fragment réplique élit de valider la transaction lorsqu'il traite normalement cette dernière. Si quelque chose ne va pas dans le fragment réplique synchrone, celui-ci décide de ne pas valider. Pour qu'un fragment primaire valide, le nombre de fragments répliques synchrones qui élisent de valider doit correspondre au paramètre minSyncReplica de la règle de déploiement. Si ce nombre est trop bas, le fragment primaire ne valide pas la transaction et une erreur se produit. Cette action garantit la disponibilité du nombre de fragments répliques synchrones avec les bonnes données. Les fragments répliques synchrones qui ont rencontré des erreurs se réenregistrent pour corriger leur état. Pour plus d'informations sur le réenregistrement, voir Récupération des fragments répliques.

Le fragment primaire lève une erreur ReplicationVotedToRollbackTransactionException si trop peu de fragments répliques synchrones ont élu de valider.

Réplication et loaders

En principe, le fragment primaire écrit de manière synchrone les modifications dans la base de données via le loader. Le loader et la base de données sont toujours synchrones. Lorsque le fragment primaire bascule vers un fragment réplique, la base de données et le loader risquent de ne plus l'être. Exemple :

- Le fragment primaire peut envoyer la transaction au fragment réplique, puis tombe en panne avant de valider vers la base de données.
- Le fragment primaire peut valider vers la base de données, puis tomber en panne avant d'envoyer au fragment réplique.

Dans les deux cas, le fragment réplique est décalé par rapport à la base de données : en avance ou en retard d'une transaction. Cette situation est inacceptable. eXtreme Scale utilise un protocole spécial et un contrat avec l'implémentation du loader afin de résoudre ce problème sans validation en deux phases. Le protocole fonctionne de la manière suivante :

Côté fragment primaire

- Envoi de la transaction avec les résultats de la transaction précédent.e
- Ecriture dans la base de données et tentative de validation de la transaction.
- Si la base de données valide, validation dans eXtreme Scale. Si elle ne valide pas, annulation de la transaction.
- Enregistrement du résultat.

Côté fragment réplique

- Réception et mise en mémoire tampon d'une transaction.
- Pour tous les résultats, envoi avec la transaction, validation de toutes les transactions mises en mémoire tampon et suppression des transactions annulées.

Côté fragment réplique lors d'un basculement

- Pour toutes les transactions mises en mémoire tampon, fourniture des transactions au loader et tentative par ce dernier de valider les transactions.
- Le loader doit avoir été écrit de manière à rendre chaque transaction idempotente.
- Si la transaction est déjà dans la base de données, le Loader n'effectue aucune opération.
- Si la transaction n'est pas dans la base de données, le Loader applique la transaction.
- Une fois que toutes les transactions ont été traitées, le nouveau fragment primaire peut commencer à servir les demandes.

Ce protocole garantit que la base de données est au même niveau que l'état du nouveau fragment primaire.

Allocation de fragments primaires et réplique

Le service de catalogue est responsable de l'organisation des fragments. Chaque grille d'objets contient un certain nombre de partitions et chaque partition contient un fragment primaire et un ensemble facultatif de fragments répliques. Le service de catalogue ne place pas les différents fragments d'une même partition dans le même conteneur. Il ne les place pas sur des conteneurs ayant la même adresse IP

(à moins que la configuration soit en mode développement). Le service de catalogue répartit les fragments de manière équilibrée entre les différents conteneurs disponibles.

Si un nouveau conteneur démarre, eXtreme Scale extrait certains fragments des conteneurs relativement surchargés et les place dans le nouveau conteneur vide. Ce comportement permet à eXtreme Scale de garantir la souplesse qui lui est essentielle. Celle-ci se manifeste à travers une puissante capacité d'évolutivité horizontale via des ajouts ou des suppressions.

Ajouts

On parle d'un ajout lorsque des machines virtuelles Java ou des conteneurs supplémentaires sont ajoutés à une grille eXtreme Scale, eXtreme Scale essaie de déplacer les fragments primaires ou réplique existants de l'ancien ensemble de JVM vers le nouveau. Ce déplacement permet à la grille de s'agrandir et de tirer parti du processeur, du réseau et de la mémoire des JVM récemment ajoutées. Il permet également d'équilibrer la grille et de garantir que chaque JVM héberge la même quantité de données. Au fur et à mesure que la taille de la grille augmente, chaque serveur héberge un sous-ensemble plus réduit de la totalité de la grille. eXtreme Scale suppose que les données sont réparties de façon égale entre les différentes partitions. Cet agrandissement correspond à un ajout.

Suppressions

On parle de suppression lorsqu'en cas de défaillance d'une JVM, eXtreme Scale essaie de procéder à une réparation. Si la JVM concernée a une machine réplique, eXtreme Scale remplace la machine réplique victime de la défaillance par une nouvelle machine réplique créée sur une JVM n'ayant subi aucun dommage. Si la JVM ayant échoué a une machine primaire, eXtreme Scale recherche la meilleure machine réplique parmi les machines n'ayant subi aucun dommage et promeut celle-ci au rang de nouvelle machine primaire. eXtreme Scale remplace alors la machine réplique promue par une nouvelle machine réplique créée sur les serveurs restants. Afin de garantir l'évolutivité, eXtreme Scale conserve le même nombre de machines réplique des partitions en cas de défaillance des serveurs.

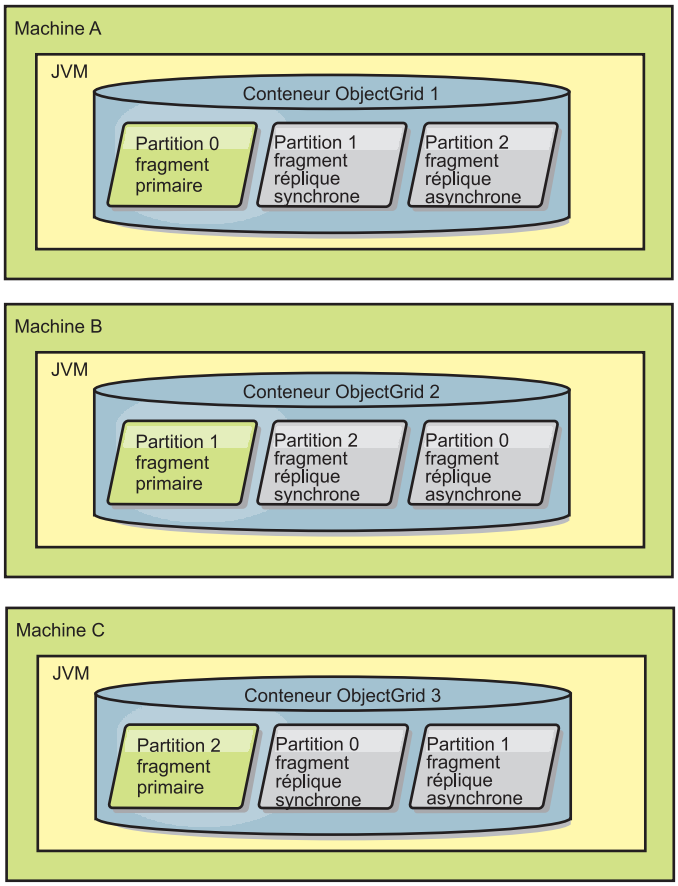


Figure 35. Organisation d'un groupe de mappes ObjectGrid avec règle de déploiement comprenant 3 partitions pour lesquelles minSyncReplicas est associé à la valeur 1, maxSyncReplicas est associé à la valeur 1 et maxAsyncReplicas est associé à la valeur 1

Lire les fragments répliques

Il est possible de configurer des groupes de mappes pour autoriser un client à lire les fragments répliqués au lieu d'être restreint aux fragments primaires.

Il peut souvent être intéressant de permettre aux fragments répliques de sortir de leur simple rôle de simples fragments primaires en puissance en cas de défaillances. Il est, par exemple, possible de configurer des groupes de mappes pour autoriser le routage des opérations de lecture vers les fragments répliques. Pour ce faire, il suffit de donner la valeur true à l'option replicaReadEnabled de MapSet. Par défaut, le paramètre a la valeur false.

Pour plus d'informations sur l'élément MapSet, voir dans le *Guide d'administration* la rubrique consacrée au fichier XML du descripteur de la règle de déploiement.

Activer la lecture des fragments répliques peut améliorer les performances en disséminant les demandes de lecture sur davantage de machines virtuelles Java™. Si l'option n'est pas activée, toutes les demandes de lecture, telles les méthodes ObjectMap.get ou Query.getResultIterator, seront routées vers les fragments primaires. Lorsque replicaReadEnabled a la valeur true, certaines demandes get risquent de retourner des données périmées ; l'application utilisant cette option doit donc pouvoir tolérer cette éventualité. Mais il ne se produira pas d'échec en

cache. Si les données ne sont pas dans le fragment réplique, la demande get est redirigée vers le fragment primaire et une nouvelle tentative est effectuée.

L'option replicaReadEnabled peut être utilisée dans les deux modes de réplication, synchrone et asynchrone.

Equilibrage de la charge entre les fragments répliques

Equilibrer la charge entre les fragments répliques s'utilise en général uniquement lorsque les clients mettent en cache des données qui changent en permanence ou lorsque les clients utilisent le verrouillage pessimiste.

eXtreme Scale, sauf configuration différente, envoie au serveur primaire toutes les demandes de lecture et d'écriture concernant un groupe de réplication donné. Ce serveur primaire doit servir toutes les demandes émanant des clients. Vous voudrez peut-être autoriser l'envoi des demandes de lecture aux répliques du fragment primaire. L'envoi des demandes de lecture aux fragments répliques permet à la charge de ces demandes d'être partagées par plusieurs machines virtuelles Java. Cela dit, il faut savoir que l'utilisation des fragments répliques pour les demandes de lecture peut donner des réponses incohérentes.

Equilibrer la charge entre les fragments répliques s'utilise en général uniquement lorsque les clients mettent en cache des données qui changent en permanence ou lorsque les clients utilisent le verrouillage pessimiste.

Si les données changent en permanence et qu'elles sont ensuite invalidées dans les caches locaux du client, le fragment primaire devrait constater en résultat un taux relativement élevé de demandes get provenant des clients. De même, en mode de verrouillage pessimiste, il n'existe aucun cache local, c'est pourquoi toutes les demandes sont envoyées au fragment primaire.

Si les données sont relativement statiques ou si le mode pessimiste n'est pas utilisé, l'envoi des demandes au fragment réplique n'a pas un énorme impact sur les performances. La fréquence des demandes get émanant des clients avec des caches pleins de données n'est pas élevée.

Lors du premier démarrage d'un client, son cache local est vide. Les demandes adressées au cache vide sont transmises au fragment primaire. Au fil du temps, le cache client obtient des données, ce qui fait tomber la charge des demandes. Si un grand nombre de clients démarrent simultanément, la charge peut être importante et la lecture par les fragments répliques peut être un choix approprié pour les performances.

Evénements de cycle de vie, de reprise et d'échec

Les fragments passent par différents états et événements pour prendre en charge la réplication. Le cycle de vie d'un fragment inclut la connexion, l'exécution, l'arrêt, le basculement et la gestion des erreurs. Les fragments peuvent passer de l'état de fragment de réplique à celui de fragment primaire afin de gérer les modifications d'état du serveur.

Evénements de cycle de vie

Lorsque les fragments primaires et répliques sont placés et démarrés, ils passent par une série d'événements qui leur permettent d'être en ligne et en mode écoute.

Fragment primaire

Le service de catalogue place un fragment primaire pour une partition. Le service de catalogue s'attache également à équilibrer les emplacements de fragment primaire et à lancer le basculement des fragments primaires.

Lorsqu'un fragment devient un fragment primaire, il reçoit du service de catalogue une liste de fragments répliques. Le nouveau fragment primaire crée un groupe de fragments répliques et enregistre tous les fragments répliques.

Lorsque le fragment primaire est prêt, un message signalant qu'il est prêt s'affiche dans le fichier `SystemOut.log` pour le conteneur d'exécution. Pour ouvrir le message ou le message CWOBJ1511I, répertorie le nom de mappe, le nom du groupe de mappes et le numéro de partition du fragment primaire démarré.

```
CWOBJ1511I: mapName:mapSetName:partitionNumber (primary) is open for business.
```

Pour plus d'informations sur le positionnement de fragments par le service de catalogue, reportez-vous à la rubrique «Allocation de fragments primaires et réplique», à la page 133.

Fragment réplique

Les fragments répliques sont principalement contrôlés par le fragment primaire à moins que le fragment réplique détecte un problème. Pendant un cycle de vie normal, le fragment primaire place, enregistre et annule l'enregistrement d'un fragment réplique.

Lorsque le fragment primaire initialise un fragment réplique, un message affiche le journal décrivant où s'exécute le fragment réplique pour indiquer que ce dernier est disponible. Pour ouvrir le message ou le message CWOBJ1511I, répertorie le nom de mappe, le nom du groupe de mappes et le numéro de partition du fragment réplique. Le message suivant s'affiche :

```
CWOBJ1511I: mapName:mapSetName:partitionNumber (synchronous replica) is open for business.
```

ou

```
CWOBJ1511I: mapName:mapSetName:partitionNumber (asynchronous replica) is open for business.
```

Fragment réplique asynchrone : Un fragment réplique scrute les données dans son fragment primaire. Le fragment réplique ajustera automatiquement l'intervalle auquel il scrute ces données s'il n'en reçoit pas du fragment primaire, ce qui est l'indice qu'il est à niveau avec ce dernier. Il ajustera également cet intervalle s'il reçoit une erreur pouvant indiquer que le fragment primaire est en panne ou qu'il y a un problème réseau.

Lorsqu'un fragment réplique passe en mode homologue, il imprime le message suivant dans son fichier `SystemOut.log`. Ce message peut apparaître plusieurs fois par message CWOBJ1511. Il s'imprimera à nouveau si le fragment réplique se connecte à un autre fragment primaire ou en cas d'ajout de mappes modèles.

```
CWOBJ1543I: Le fragment réplique asynchrone objectGridName:mapsetName:partitionNumber a démarré ou
a continué à se répliquer à partir du fragment primaire.
Réplication en cours pour les mappes : [nomMappe]
```

Fragment réplique synchrone : Lorsque le fragment réplique démarre pour la première fois, il n'est pas encore en mode homologue. Lorsqu'un fragment réplique est en mode homologue, il reçoit les données du fragment primaire au fur et à mesure de leur arrivée dans ce dernier. Avant de passer en mode homologue, le fragment réplique a besoin d'une copie de toutes les données existant dans le fragment primaire.

Le fragment réplique synchrone copie les données depuis le fragment primaire comme cela se passe vers un fragment réplique asynchrone, en scrutant le fragment primaire. Lorsqu'il copie les données existantes à partir du fragment primaire, il passe en mode homologue et commence à recevoir recevoir les données en même temps que celles-ci parviennent au fragment primaire.

Lorsqu'un fragment réplique atteint le mode homologue, il imprime un message dans son fichier SystemOut.log. La valeur de temps se réfère à la durée qu'il a fallu au fragment réplique pour obtenir toutes ses données initiales du fragment primaire. La valeur de temps s'affiche comme étant zéro ou étant très faible si le fragment primaire ne comporte aucune donnée existante à répliquer. Ce message peut apparaître plusieurs fois par message CWOBJ1511. Il s'imprimera à nouveau si le fragment réplique se connecte à un autre fragment primaire ou en cas d'ajout de mappes modèles.

```
CWOBJ1526I: Replica objectGridName:mapsetName:partitionNumber:mapName entering peer
mode after X seconds.
```

Lorsque le fragment réplique synchrone est en mode homologue, le fragment primaire doit répliquer les transactions vers la totalité des fragments répliques synchrones qui sont en mode homologue. Le fragment réplique synchrone demeure au même niveau que les données du fragment primaire. Si, dans la règle de déploiement, il est défini un nombre minimum de fragments répliques ou si minSync est défini dans cette règle, ce nombre de fragments répliques synchrones doit voter pour valider avant que la validation de la transaction puisse s'effectuer dans le fragment primaire.

Evénements de reprise

La finalité de la réplication est de permettre la reprise après des échecs et des événements d'erreur. Si un fragment primaire tombe en panne, un autre fragment réplique prend le relais. Si des erreurs sont détectées sur les fragments répliques, le fragment réplique tente de se rétablir. Le service de catalogue contrôle le positionnement et les transactions sur les nouveaux fragments primaires ou sur les nouveaux fragments répliques.

Les fragments répliques deviennent un fragment primaire

Un fragment réplique devient un fragment primaire pour deux raisons. Le fragment primaire s'est arrêté ou a échoué ou une décision d'équilibre a été prise pour déplacer le fragment précédent vers un nouvel emplacement.

Le service de catalogue sélectionne un nouveau fragment primaire des fragments répliques synchrones existants. Si un déplacement de fragment primaire est nécessaire et qu'il n'existe aucun fragment réplique, un fragment réplique temporaire sera positionné pour effectuer la transition. Le nouveau fragment primaire enregistre tous les fragments répliques existants et accepte les transactions en tant que nouveau fragment primaire. Si les fragments répliques existants comportent le niveau de données adéquat, les données en cours sont conservées lorsque le fragment réplique s'enregistre auprès du nouveau fragment primaire. Les fragments répliques asynchrones scruteront le nouveau fragment primaire.

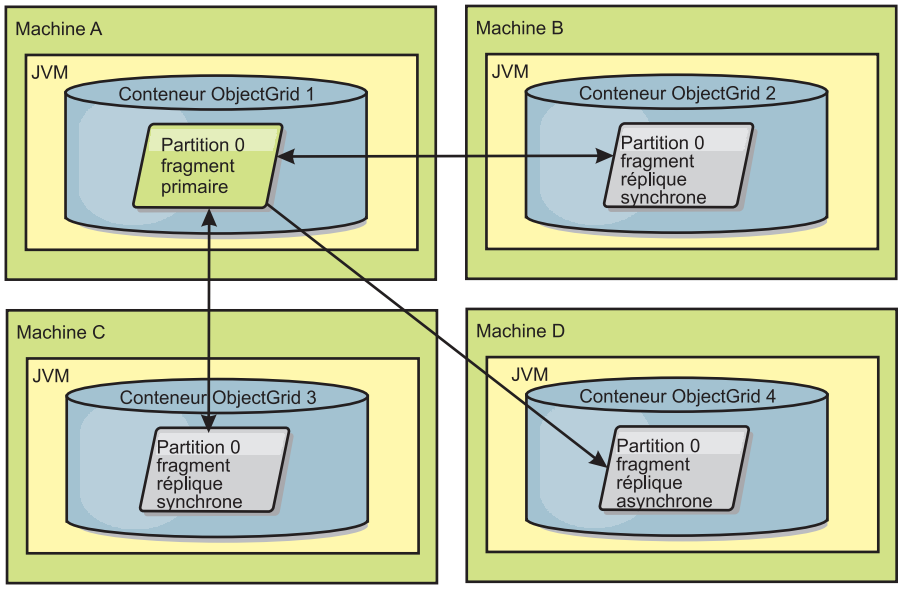


Figure 36. Exemple de positionnement d'une mappe ObjectGrid définie pour la partition partition0. La règle de déploiement comprend une valeur minSyncReplicas de 1, une valeur maxSyncReplicas de 2 et une valeur maxAsyncReplicas de 1.

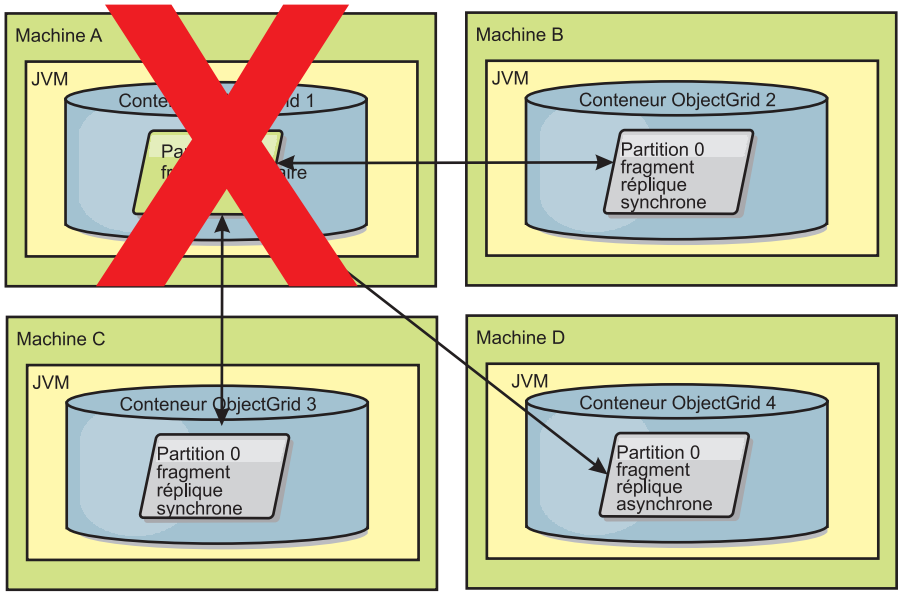


Figure 37. Le conteneur pour le fragment primaire échoue.

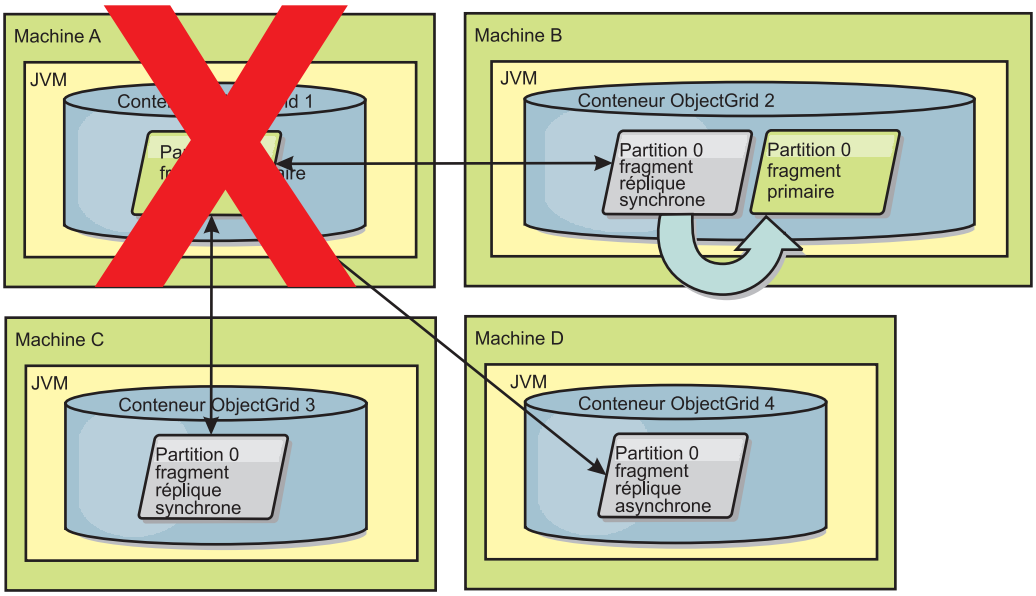


Figure 38. Le fragment de réplique synchrone sur le conteneur ObjectGrid 2 devient un fragment primaire.

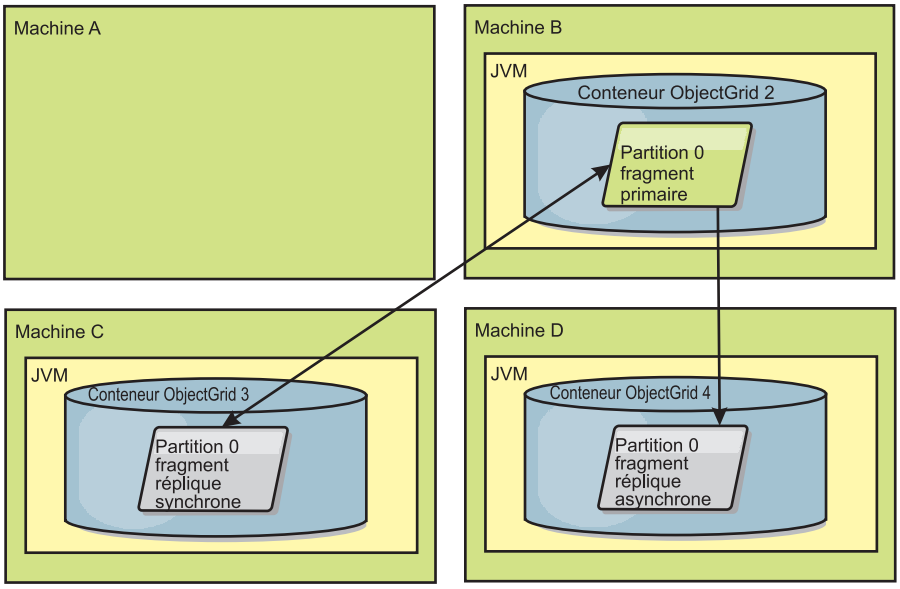


Figure 39. La machine B contient le fragment primaire. En fonction de la définition du mode de réparation automatique et de la disponibilité des conteneurs, un nouveau fragment réplique synchrone peut ou peut ne pas être placé sur une machine.

Reprise des fragments répliques

Un fragment réplique synchrone est contrôlé par le fragment primaire. Toutefois, si un fragment réplique détecte un problème, il peut déclencher un événement de réenregistrement pour corriger l'état des données. La réplique supprime les données en cours et obtient une nouvelle copie du fragment primaire.

Lorsqu'un fragment réplique lance un événement de réenregistrement, il imprime un message de journal.

```
CWOBJ1524I: Replica listener
objectGridName:mapSetName:partition must re-register with the primary.
Reason: Exception listed
```

Si une transaction cause une erreur sur un fragment réplique pendant le traitement, le fragment réplique est à l'état inconnu. La transaction a correctement eu lieu sur le fragment primaire mais un incident s'est produit sur le fragment réplique. Pour remédier à cette situation, le fragment réplique lance un événement de réenregistrement. Avec une nouvelle copie des données du fragment primaire, le fragment réplique peut se poursuivre. Si le même problème se reproduit, le fragment réplique n'effectue pas le réenregistrement en continu. Voir «Evénements d'arrêt anormal» pour plus de détails.

Evénements d'arrêt anormal

Un fragment réplique peut arrêter de répliquer les données s'il rencontre des situations d'erreur dans lesquelles il n'a aucun moyen de reprendre son activité.

Trop de tentatives d'enregistrement

Si un fragment réplique déclenche un réenregistrement plusieurs fois sans parvenir à valider les données, il s'arrête. L'arrêt empêche une réplique d'entrer dans une boucle de réenregistrement infinie. Par défaut, un fragment réplique tente de réenregistrer trois fois dans une ligne avant de s'interrompre.

Si une réplique réenregistre trop de fois, elle imprime le message suivant dans le journal.

```
CWOBJ1537E: objectGridName:mapSetName:partition exceeded the maximum number
of times to reregister (timesAllowed) without successful transactions.
```

Si le fragment réplique est incapable d'être restauré en se réenregistrant, un problème pervasif peut exister avec les transactions relatives à ce fragment. Il pourrait en résulter un manque de ressources sur le chemin d'accès aux classes si une erreur survient lors de l'inflation des clés ou des valeurs de la transaction.

Echec lors de l'activation du mode homologue

Si un fragment réplique tente de passer en mode homologue et rencontre une erreur lors du traitement des données en bloc existantes à partir du fragment primaire (données de point de contrôle), le fragment réplique s'arrête. L'arrêt empêche le fragment réplique démarrer avec des données initiales erronées. Etant donné qu'il reçoit les mêmes données du fragment primaire s'il se iréenregistre, le fragment réplique ne fait aucune nouvelle tentative.

Si une réplique ne parvient pas à entrer en mode homologue, elle imprime le message suivant dans le journal :

```
CWOBJ1527W Replica objectGridName:mapSetName:partition:mapName failed to enter peer mode after numSeconds seconds.
```

Un message supplémentaire s'affiche dans le journal et explique pourquoi le fragment réplique n'a pu passer en mode homologue.

Reprise après réenregistrement ou échec du mode homologue

Si une réplique ne peut pas se réenregistrer ou entrer en mode homologue, elle est à l'état inactif jusqu'à ce qu'un nouvel événement de positionnement ait lieu. Un nouvel événement de positionnement peut être le démarrage ou l'arrêt d'un nouveau serveur. Vous pouvez également démarrer un événement de

positionnement à l'aide de la méthode triggerPlacement sur le bean géré PlacementServiceMBean.

Routage par zone préférée

Grâce au routage par zone préférée, eXtreme Scale peut rediriger des transactions vers des zones en fonction de vos spécifications.

WebSphere eXtreme Scale vous permet d'exercer un contrôle significatif sur l'emplacement des fragments d'un composant ObjectGrid.

Le routage par zone préférée permet aux clients eXtreme Scale de spécifier une pondération pour une zone particulière ou un ensemble de zones. eXtreme Scale tentera d'acheminer les transactions client vers les zones préférées avant de les acheminer vers une autre zone.

Exigences concernant le routage par zone préférée

Il existe divers facteurs à prendre en compte avant de tenter un routage par zone préférée. Assurez-vous que l'application est capable de satisfaire les exigences de votre scénario.

Le positionnement de partition par conteneur est nécessaire à l'optimisation du routage par zone préférée. Cette stratégie de positionnement est adaptée aux applications stockant des données de session dans ObjectGrid. Par défaut, pour WebSphere eXtreme Scale, le partitionnement fixe est la stratégie de positionnement. Les clés sont hachées au moment de la validation de la partition, afin de déterminer quelle partition héberge la paire clé-valeur de la mappe par le biais du positionnement par partition fixe.

Le positionnement par conteneur assigne vos données à une partition aléatoire au moment de la validation de la transaction par le biais de SessionHandle. Vous devez être capable de reconstruire le SessionHandle afin de récupérer vos données à partir d'ObjectGrid.

Etant donné que vous pouvez utiliser des zones pour accroître votre contrôle sur l'emplacement où résideront les fragments primaires et leur réplique dans le domaine, le déploiement multi-zone est avantageux si vos données résident sur des emplacements physiques multiples. La séparation physique des fragments primaires et de leur réplique est une façon de s'assurer que la perte irrémédiable d'un centre de données n'aura pas de conséquence sur la disponibilité des données.

Lorsque les données sont réparties dans une topologie multi-zone, il est probable que les clients soient également répartis à travers la topologie. Le routage des clients vers leur zone ou centre de données local(e) présente un avantage en termes de performances, puisqu'il réduit le temps d'attente du réseau. Utilisez ce scénario autant que possible.

Configuration de votre topologie pour le routage par zone préférée

Réfléchissez au scénario suivant. Vous disposez de deux centres de données : Chicago et London. Afin de minimiser le temps de réponse des clients, vous souhaitez que les clients lisent et écrivent des données vers leur centre de données local.

Les fragments primaires ObjectGrid doivent être placés dans chaque centre de données de sorte que les transactions puissent être écrites localement à partir de chaque emplacement. De plus, les clients devront avoir connaissance des zones afin d'acheminer les données vers la zone locale.

Le positionnement par conteneur localise de nouveaux fragments primaires sur chaque conteneur démarré. Les fragments répliques sont positionnés en fonction des règles de zone et de positionnement spécifiées par la règle de déploiement. Par défaut, une réplique est placée dans une autre zone que la zone principale. Examinez la règle de déploiement suivante pour ce scénario.

```
<?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="universe">
  <mapSet name="mapSet1" placementStrategy="PER_CONTAINER"
   numberOfPartitions="3" maxAsyncReplicas="1">
   <map ref="planet" />
  </mapSet>
 </objectgridDeployment>
</deploymentPolicy>
```

Chaque conteneur appliquant la règle de déploiement reçoit trois nouveaux fragments. A chaque fragment primaire correspond une réplique asynchrone. Démarrez chaque conteneur avec le nom de zone approprié. Utilisez le paramètre -zone si vous lancez vos conteneurs avec le script startOgServer.

Pour un serveur de conteneur à Chicago :

- UNIX Linux

```
startOgServer.sh s1 -objectGridFile ../xml/universeGrid.xml
-deploymentPolicyFile ../xml/universeDp.xml
-catalogServiceEndpoints MyServer1.company.com:2809
-zone Chicago
```

- Windows

```
startOgServer.bat s1 -objectGridFile ../xml/universeGrid.xml
-deploymentPolicyFile ../xml/universeDp.xml
-catalogServiceEndpoints MyServer1.company.com:2809
-zone Chicago
```

Si vos conteneurs s'exécutent sur WebSphere Application Server, vous devez créer un groupe de nœuds et inclure le préfixe "ReplicationZone" dans son nom. Les serveurs s'exécutant sur les nœuds dans de tels groupes de nœuds seront placés dans la zone appropriée. Par exemple, les serveurs s'exécutant sur un nœud de Chicago peuvent appartenir à un groupe de nœuds "ReplicationZoneChicago".

Les fragments primaires pour la zone de Chicago auront leur réplique dans la zone de London. Les fragments primaires de la zone de London auront leur réplique dans la zone de Chicago.

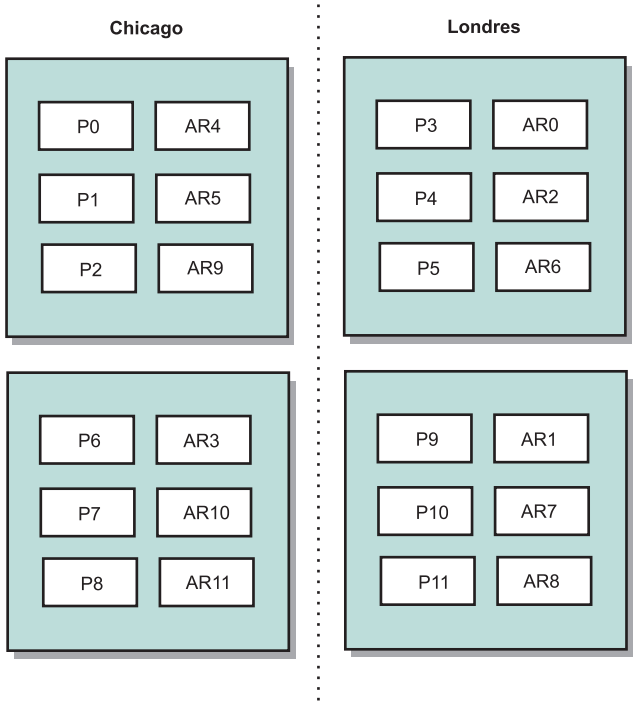


Figure 40. Segments principaux et répliques dans les zones

Définissez les zones préférées pour les clients. Ces informations peuvent être fournies de plusieurs façons. La façon la plus simple est de fournir un fichier de propriétés client à votre machine virtuelle Java. Créez un fichier objectGridClient.properties et assurez-vous qu'il est au même emplacement que les classes. Pour plus d'informations, voir dans le *Guide d'administration* la rubrique consacrée au fichier de propriétés.

Incluez la propriété preferZones dans le fichier. Définissez la valeur de propriété sur la zone appropriée. La ligne de code suivante doit être incluse dans le fichier objectGridClient.properties pour les clients à Chicago.

```
preferZones=Chicago
```

Le fichier de propriétés pour les clients de London doit inclure la ligne

```
preferZones=London
```

Cette propriété donne l'instruction à chaque client d'acheminer les transactions vers la zone locale dans la mesure du possible. Les données insérées dans un fragment primaire dans la zone locale sont répliquées de façon asynchrone vers la zone externe.

Utilisation du SessionHandle pour un routage vers la zone locale

La stratégie de positionnement par conteneur n'utilise pas un algorithme basé sur hachage pour déterminer l'emplacement de vos paires clé-valeur dans ObjectGrid. Votre ObjectGrid doit utiliser SessionHandles pour s'assurer que les transactions sont acheminées vers l'emplacement adéquat lorsque vous utilisez cette stratégie de positionnement. Lorsqu'une transaction est validée, un SessionHandle est lié à la session si aucun n'a été défini. Le SessionHandle peut être également lié à la

Session en appelant Session.getSessionHandle avant la validation de la transaction. L'extrait de code suivant illustre la liaison d'un SessionHandle préalablement à la validation de la transaction.

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

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

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

// tran est acheminé vers la répartition spécifiée par SessionHandle
ogSession.commit();
```

Supposez que le code ci-dessus a été exécuté sur un client dans le centre de données de Chicago. Etant donné que l'attribut preferZones a été défini sur Chicago pour ce client, cette transaction doit être acheminée vers l'une des partitions principales de la zone de Chicago (partition 0, 1, 2, 6, 7 ou 8).

Ce SessionHandle est le chemin de retour vers la partition stockant ces données validées. Le SessionHandle doit être réutilisé ou reconstruit et défini sur la Session afin de revenir à la partition contenant les données validées.

```
ogSession.setSessionHandle(sessionHandle);
ogSession.begin();

// la valeur renvoyée sera "mercury"
String value = map.get("planet1");
ogSession.commit();
```

Etant donné que cette transaction réutilise le SessionHandle créé au cours de la transaction insert, la transaction get sera acheminée vers la partition contenant les données insérées. Cette transaction sera dans l'impossibilité de récupérer les données précédemment insérées si le SessionHandle n'a pas été défini.

Conséquences des échecs de conteneur et de zone sur le routage basé sur zones

Un client ayant la propriété preferZones définie verra toutes ses transactions acheminées vers la ou les zones spécifiées dans ces circonstances normales. Cependant, la perte d'un conteneur entraîne la promotion d'une réplique en un fragment primaire dans une zone externe. Un client qui acheminait auparavant les transactions vers les partitions de la zone locale peut être forcé à entrer dans la zone externe afin de récupérer les données préalablement insérées.

Imaginez le scénario suivant. Un conteneur de la zone de Chicago est perdu. Il contenait précédemment des fragments primaires pour les partitions 0, 1 et 2. Les nouveaux fragments primaires pour ces partitions seront dans la zone de London étant donné que cette zone hébergeait les fragments répliques pour ces partitions.

Tout client de la zone de Chicago utilisant un SessionHandle pointant vers l'une des partitions de basculement sera à présent acheminé vers London. Les clients de Chicago utilisant des nouveaux SessionHandles seront acheminés vers les fragments primaires basés à Chicago.

De même, si toute la zone de Chicago est perdue, tous les fragments répliques de la zone de London deviennent des fragments primaires. Dans ce cas, tous les clients de Chicago voient leurs transactions acheminées vers London.

Topologies de réplication de grilles multi-maîtres (PA)

La réplication asynchrone multi-maître permet à deux grilles, voire plus, de devenir des miroirs exacts les unes des autres. Cette mise en miroir est effectuée à l'aide d'une réplication asynchrone entre des liens interconnectant les grilles. Chaque grille est hébergée au sein d'un domaine complètement indépendant, possédant son propre service de catalogue et ses propres serveurs conteneurs et portant un nom exclusif. La réplication asynchrone multi-maître permet d'utiliser des liens pour interconnecter une collection de ces domaines et de synchroniser ensuite ces derniers grâce, précisément, à la réplication via ces liens. eXtreme Scale permet de construire quasiment n'importe quelle topologie, car la définition des liens entre les domaines est laissée à l'appréciation et à l'initiative des administrateurs.

Les domaines : des grilles avec des caractéristiques spécifiques

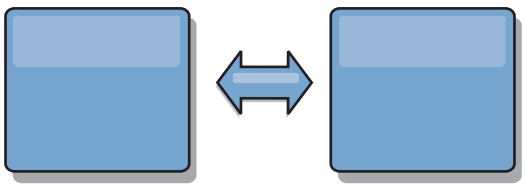
Les grilles utilisées dans les topologies de réplication multi-maître sont également désignées sous le nom de *domaines*. Chaque domaine doit avoir les caractéristiques suivantes :

- disposer d'un service de catalogue privé avec un nom de domaine exclusif
- avoir le même nom de grille que les autres grilles du domaine
- avoir le même nombre de partitions que les autres grilles du domaine
- être une grille FIXED_PARTITION (les grilles PER_CONTAINER ne peuvent être répliquées)
- avoir le même nombre de partitions (sans forcément pour autant avoir le même nombre et le même type de fragments répliques)
- avoir les mêmes types de données à répliquer que les autres grilles du domaine
- avoir le même nom de groupes de mappes (mapsets), le même nom de mappes et les mêmes modèles de mappes dynamiques que les autres grilles du domaine

Tous les groupes de mappes ayant les caractéristiques énoncées ci-dessus seront répliqués après que les domaines de services de catalogue auront été démarrés. Voir «Topologies de réplication de grilles multi-maîtres (PA)», à la page 24 pour connaître les groupes de mappes qui ne seront pas répliquées.

Liens connectant des domaines

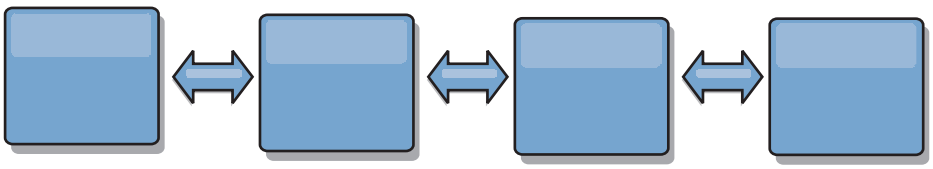
Une infrastructure de grilles de réplication est un graphe connecté de domaines reliés par des liens bidirectionnels. Un lien permet à deux domaines d'échanger des modifications de données. Ainsi, la topologie la plus simple sera une paire de domaines reliés par un seul lien. Les domaines sont nommés en partant de A, puis B, et ainsi de suite, à partir de la gauche. Le lien peut traverser un réseau WAN étendu, couvrant de longues distances. En cas d'interruption du lien, les modifications peuvent toujours être apportées aux données dans l'un ou l'autre des deux domaines. La réconciliation des modifications s'effectue plus tard, lorsque le lien recommence à connecter les domaines. Les liens tentent automatiquement de se reconnecter si la connexion réseau est interrompue.



Une fois que les liens ont été définis, eXtreme Scale va tout d'abord tenter de rendre identique chaque domaine, puis il va essayer ensuite de les maintenir dans un état identique lorsque des modifications se produiront dans l'un de ces domaines. L'objectif d'eXtreme Scale est que chaque domaine soit un miroir exact de tout autre domaine connecté par les liens. Les liens de réplication entre les domaines aident à garantir que toute modification effectuée dans un domaine est copiée vers les autres domaines.

Topologies linéaires

Bien qu'elle figure au rang des topologies les plus simples, la topologie linéaire illustre un certain nombre de qualités des liens. Tout d'abord, il n'est pas nécessaire qu'un domaine soit directement connecté à chacun des autres domaines pour recevoir des modifications. Le domaine B ira prendre des modifications dans le domaine A. Le domaine C reçoit les modifications du domaine A via le domaine B, lequel connecte les domaines A et C. De la même manière, 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.



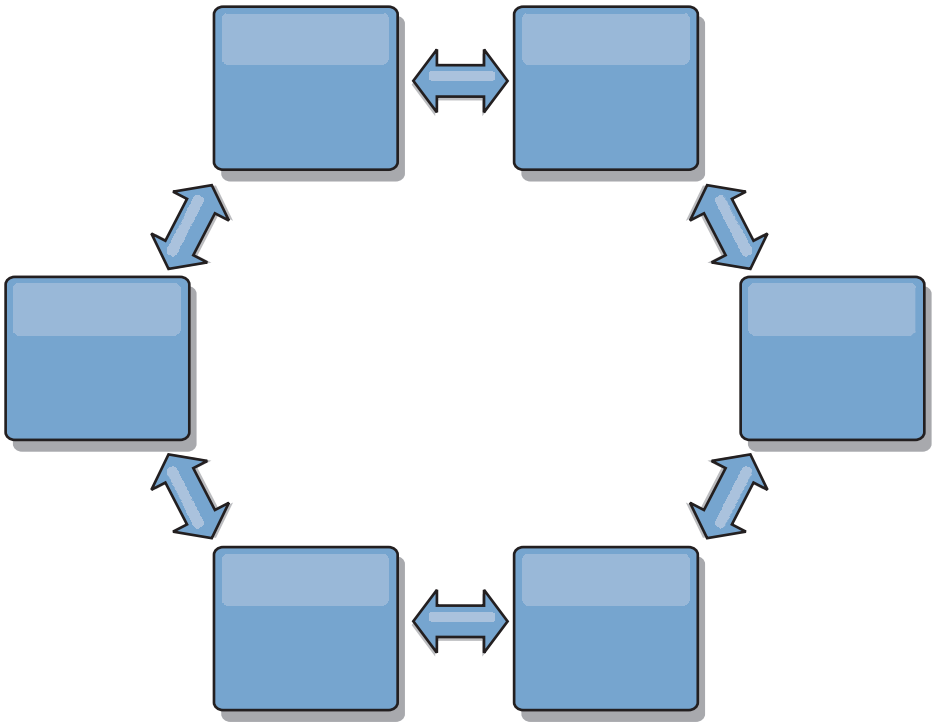
Et si le domaine C tombait en panne, les événements suivants se produiraient :

1. Le domaine D serait orphelin jusqu'au redémarrage du domaine C.
2. Le domaine C se synchroniserait avec le domaine B, lequel est une copie du domaine A.
3. Le domaine D utiliserait le domaine C pour se synchroniser avec les modifications intervenues sur les domaines A et B pendant que le domaine D était orphelin (et que le domaine C était arrêté).

A la fin, les domaines A, B, C et D redeviendraient tous identiques.

Topologies en anneau

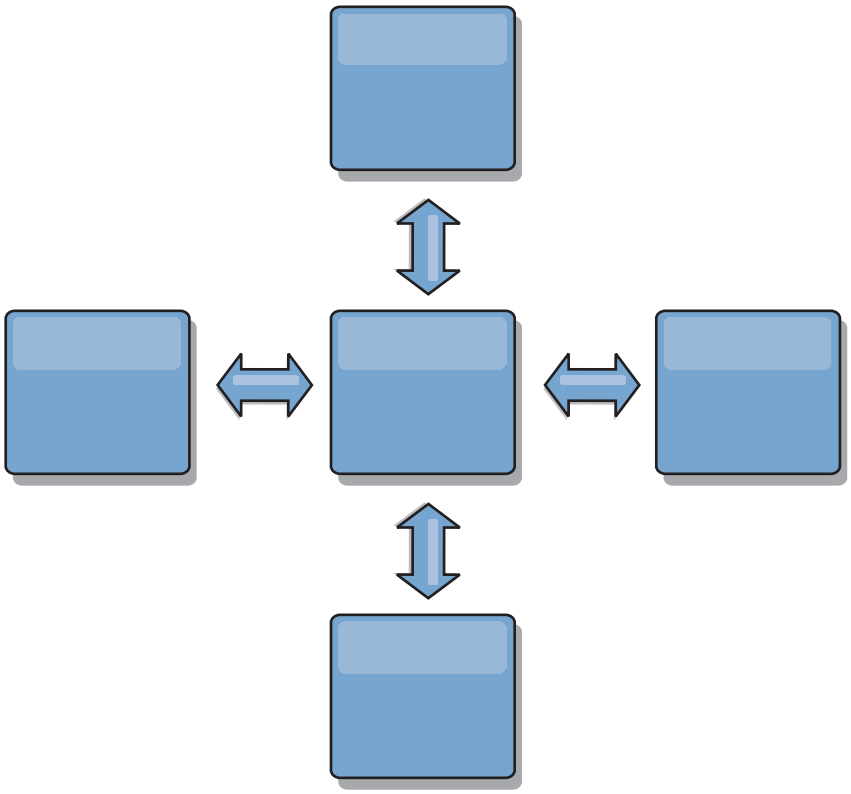
Les topologies en anneau sont un exemple de topologie encore plus résilientes. En effet, la défaillance d'un domaine ou d'un lien n'empêche pas les domaines survivants de continuer à obtenir des modifications en circulant autour de l'anneau et en s'éloignant de la défaillance. Chaque domaine a deux liens vers les autres domaines. Chaque domaine a au maximum deux liens, quelle que soit la taille de la topologie en anneau. Le délai de propagation des modifications peut être important, car les modifications d'un domaine particulier peuvent avoir besoin de traverser plusieurs domaines avant que la totalité des domaines ne puisse voir la totalité des modifications. Une topologie linéaire a le même problème.



Représentez-vous une topologie en anneau plus sophistiquée, avec un domaine racine au centre de l'anneau. Le domaine racine joue le rôle de chambre centrale de compensation tandis que les autres domaines font office de chambres distantes de compensation pour les modifications se produisant dans le domaine racine. Le domaine racine peut arbitrer les modifications entre les domaines. Si une topologie en anneau contient plusieurs anneaux autour d'un domaine racine, ce dernier ne pourra arbitrer les modifications que dans les domaines de l'anneau le plus proche du centre. Mais les résultats de l'arbitrage seront ventilés vers les domaines des autres anneaux.

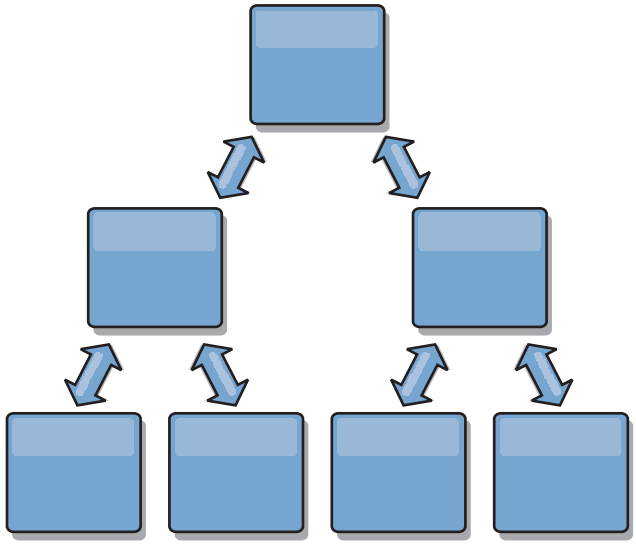
Topologies en étoile

Si, dans une topologie en étoile, les délais d'attente sont moindres, les modifications voyageant au maximum sur un domaine intermédiaire (le concentrateur), cette topologie ne va pas sans autres problèmes. Elle a un domaine central qui fait office de « moyeu » ou de concentrateur. Ce domaine concentrateur est connecté via un lien à chacun des domaines qui jouent le rôle de « rayons » de la roue. On le voit, la charge de la répartition des modifications entre les domaines repose toute entière sur le concentrateur. Ce dernier fait office de chambre de compensation en cas de collisions, ce qui peut s'avérer important dans certains scénarios. Dans un environnement soumis à une fréquence élevée de modifications, pour pouvoir rester actif et opérationnel, le concentrateur peut avoir besoin de s'exécuter sur plus de matériels que les autres noeuds. eXtreme Scale est conçu pour évoluer de manière linéaire, ce qui signifie que l'on peut, si nécessaire, étoffer le concentrateur sans difficultés. Mais, si le concentrateur vient à tomber en panne, il faudra attendre qu'il ait redémarré pour que les changements puissent être répartis. Toutes les modifications intervenues sur les domaines autres que le concentrateur seront réparties après la reconnexion de ce dernier.



Topologies en arbre

Dernier exemple de topologie : une arborescence acyclique dirigée. Par acyclique, l'on entend qu'aucun cycle (aucune boucle) ne se passe sur l'arborescence. Dirigée signifie qu'il n'existe de liens qu'entre des parents et des enfants. Cette configuration peut être utile pour les topologies comprenant tant de domaines qu'il ne serait pas pratique d'avoir un concentrateur connecté de manière centralisée à chaque ordinateur possible. Autre cas de figure où cette topologie trouve toute son utilité : le besoin d'ajouter des domaines enfants sans avoir à modifier le domaine racine.



Cette topologie peut toujours avoir une chambre de compensation centrale avec le domaine racine, mais le deuxième niveau peut faire office de chambres de compensation pour les modifications se produisant dans le domaine situé au niveau inférieur. Le domaine racine ne peut arbitrer que les modifications entre les domaines de ce deuxième niveau. Des arbres n-aires sont également possibles. Un arbre n-aire a n enfants à chaque niveau. Chaque domaine a un déploiement de n.

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 pour qu'ils aient tous la même quantité de processeurs, de mémoire et de ressources réseau. Vous vous rendrez sans doute compte que les domaines gérant les collisions de modifications (arbitrage) utilisent plus de ressources que les autres domaines. Les collisions sont détectées de manière automatique. Deux mécanismes permettent de les résoudre :

- **Arbitre par défaut**. Le protocole par défaut est d'utiliser les modifications provenant du domaine dont le nom vient en premier par ordre alphabétique. Supposons par exemple que les domaines A et B génèrent un conflit pour un enregistrement, dans ce cas, la modification du domaine B sera ignorée. Le domaine A conserve sa version et l'enregistrement dans le domaine B est modifié de manière à correspondre à celui du domaine A. Cela s'applique également pour les applications où les utilisateurs ou les sessions sont liées de manière normale ou ont une affinité avec l'une des grilles.
- **Arbitre personnalisé**. Les applications peuvent très bien fournir un arbitre personnalisé. Lorsqu'un domaine détecte une collision, il fait appel à l'arbitre. Pour savoir comment développer un arbitre personnalisé de bonne qualité, voir Développement d'arbitres personnalisés pour la réplication multi-maître.

Si des collisions doivent être possibles dans les topologies que vous envisagez, pensez à utiliser une topologie en étoile ou en arbre. Il est en effet plus facile dans ces deux topologies d'éviter des collisions sans fin, ce qui peut arriver lorsque :

1. plusieurs domaines subissent une collision
2. chaque domaine résout la collision en local, ce qui produit des révisions
3. les révisions entrent en collision, d'où des révisions de révisions
4. et ainsi de suite, au fur et à mesure que les révisions sont propagées dans les divers domaines tout en essayant d'atteindre la synchronicité

Pour éviter des collisions sans fin, choisissez un domaine spécifique – un *domaine d'arbitrage* – comme gestionnaire des collisions d'un sous-ensemble de domaines. Exemple : une topologie en étoile pourra utiliser le concentrateur comme gestionnaire des collisions. Le gestionnaire de collisions ignore les collisions détectées par les sous-domaines. Le domaine du concentrateur va créer des révisions en empêchant les révisions de collisions qui échappent à tout contrôle. Le domaine qui s'est vu attribuer la gestion des collisions doit se lier à tous les domaines dont il est chargé de résoudre les collisions. Dans une topologie en arbre, tous les domaines parents internes résolvent les collisions pour leurs enfants immédiats. Par contraste, si vous utilisez une topologie en anneau, vous ne pouvez désigner un domaine de l'anneau comme résolvant les collisions.

Le tableau qui suit récapitule les approches en matière d'arbitrage qui sont les plus compatibles avec les diverses topologies.

Tableau 14. Approches en matière d'arbitrage. Ce tableau énonce si l'arbitrage entre applications est compatible avec les diverses topologies.

Topologie	Arbitrage entre applications ?	Commentaires
Linéaire à deux domaines	Oui	Choisissez l'un des domaines comme arbitre.
Linéaire à trois domaines	Oui	L'arbitre doit être le domaine du milieu. Pensez à ce domaine comme au concentrateur d'une topologie en étoile simple.
Linéaire à plus de trois domaines	Non	L'arbitrage entre applications n'est pas pris en charge.
Concentrateur avec n "rayons"	Oui	Le domaine d'arbitrage doit être le concentrateur avec des liens vers tous les "rayons".
Anneau de n domaines	Non	L'arbitrage entre applications n'est pas pris en charge.
Arbre dirigé acyclique (arbre n-aire)	Oui	Tous les noeuds racine ne doivent arbitrer que leurs descendants directs.

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**

  Les temps d'attente des modifications sont déterminés par le nombre de domaines intermédiaires que doivent traverser les modifications avant d'arriver à destination.

  Les topologies qui offrent les meilleurs temps d'attente sont celles qui éliminent les domaines intermédiaires en liant chacun des domaines à chacun des autres domaines. Mais un domaine doit effectuer la réplication à proportion du nombre de ses liens. Pour les topologies de grande taille, le nombre de liens à définir peut constituer une lourde charge pour les administrateurs.

  La vitesse à laquelle est une modification copiée vers les autres domaines dépend de facteurs supplémentaires :

  – le processeur et la bande passante réseau sur le domaine source
  – le nombre de domaines intermédiaires et de liens entre le domaine source et le domaine cible
  – les ressources de processeur et de réseau utilisables par les domaines source, cible et intermédiaires

- **Tolérance aux pannes**

  La tolérance aux pannes est déterminée par le nombre de chemins existant entre deux domaines pour la réplication des modifications.

  S'il n'existe qu'un seul lien entre les domaines, en cas de défaillance du lien, les modifications ne seront pas propagées. Si ce seul lien entre deux domaines passe par des domaines intermédiaires, les modifications ne seront pas non plus propagées si l'un des domaines intermédiaires tombe en panne.

  Prenons la topologie linéaire à trois domaines, A, B et C :

  ```
  A <-> B <-> C
  ```

Si l'une des situations suivantes se présente, le domaine C ne verra pas les modifications de A :

– le domaine A est actif et le domaine B est arrêté
– le lien entre A et B ne fonctionne pas
– le lien entre B et C ne fonctionne pas

Par contraste, une topologie en anneau permet à chaque domaine d'extraire les modifications dans un sens ou dans l'autre.

```
A <-> B <-> C <-> retour vers A
```

Par exemple, si le domaine B est arrêté, le domaine C continue d'extraire les modifications directement depuis le domaine A.

Une conception en étoile dépend du concentrateur par qui passent toutes les modifications ; s'il s'arrête, tout s'arrête. Mais il ne faut pas oublier qu'un domaine reste une grille totalement tolérante aux pannes et qui peut souffrir des défaillances du réseau WAN ou des centres de données physiques.

- **Performances**

  Le nombre des liens définis sur un domaine affecte les performances. Plus de liens utilisent davantage de ressources, et cela peut se traduire par une chute des performances de la réplication. La capacité d'un domaine A à extraire les modifications via d'autres domaines décharge efficacement ce domaine A de répliquer partout ses transactions. *La charge de la répartition des modifications sur un domaine est limitée au nombre des liens qu'utilise ce domaine. Cela n'a rien à voir avec le nombre de domaines dans la topologie.* Cette propriété permet l'évolutivité et autorise à partager la charge de la répartition des modifications entre les domaines de la topologie, plutôt que d'imposer cette charge à un seul domaine.

  Un domaine peut extraire les modifications indirectement via d'autres domaines. Prenons une topologie linéaire à cinq domaines :

  ```
  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 et les modifications de A, B et C via D

  La charge de la répartition sur les domaines A et E est la plus faible, car ces domaines ont chacun un seul lien avec un seul domaine. La charge de la répartition sur les domaines B, C et D est le double de celle sur les domaines A et E car B, C et D ont chacun un lien avec deux domaines. Cette répartition des charges demeurerait constante même si la topologie linéaire contenait 1 000 domaines, car la charge dépend du nombre de liens de chaque domaine et non du nombre globale de domaines dans la topologie.

Points à prendre en considération concernant les performances

Les limitations suivantes sont à prendre en compte pour les topologies de réplication multi-maître.

- **Optimisation de la répartition des modifications** (abordée plus haut).
- **Temps d'attente de la réplication** (abordée plus haut).

- **Performances des liens de réplication** eXtreme Scale crée un seul socket TCP/IP entre n'importe quelle paire de machines virtuelles Java. Tout le trafic entre ces machines virtuelles Java passe par ce socket, y compris la réplication multi-maître. Les domaines étant hébergés sur au moins n machines virtuelles Java conteneurs, fournissant au moins n liens TCP vers les domaines homologues, les domaines ayant le plus grand nombre de conteneurs sont ceux qui offrent les plus hauts niveaux de performances pour la réplication. Plus de conteneurs est synonyme de davantage de ressources processeur et réseau.
- **Optimisation de la fenêtre dynamique TCP et RFC 1323** Activer la prise en charge de la RFC 1323 aux deux extrémités d'un lien permet à davantage de données d'effectuer des aller-retour, ce qui donne des débits plus élevés. Cette technique étend les capacités de la fenêtre d'un facteur d'environ 16 000.

  N'oubliez pas que les sockets TCP utilisent un mécanisme de fenêtre dynamique pour contrôle le flux des données en vrac, qui limite normalement le socket à 64 Ko pour un intervalle d'aller-retour. Si cet intervalle 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 les débit sur les liens à hauts temps d'attente.

  Plusieurs facteurs peuvent affecter les performances de la réplication :
  - la vitesse à laquelle eXtreme Scale peut extraire les modifications
  - la vitesse à laquelle eXtreme Scale peut extraire les demandes de réplication
  - la capacité de la fenêtre dynamique
  - l'optimisation de la mémoire tampon réseau des deux côtés d'un lien pour permettre à eXtreme Scale d'extraire les modifications sur le socket à la vitesse maximale possible
- **Sérialisation des objets** Toutes les données doivent être sérialisables. Si un domaine n'utilise pas COPY_TO_BYTES, il doit utiliser la sérialisation Java ou ObjectTransformers pour optimiser les performances de la sérialisation.
- **Compression** eXtreme Scale compresse par défaut toutes les données envoyées entre les domaines. La version actuelle ne permet pas de désactiver la compression.
- **Optimisation de la mémoire** *L'utilisation de la mémoire pour une topologie de réplication multi-maître est largement indépendante du nombre de domaines présents dans la topologie.*

  Activer la réplication multi-maître ajoute du temps système fixe par entrée de mappe pour gérer la vérification des versions. Chaque conteneur suit également une quantité fixe de données pour chacun des domaines de la topologie. Une topologie à deux domaines utilise approximativement la même mémoire qu'une topologie à cinquante domaines. eXtreme Scale n'utilise pas dans son implémentation de replay logs ou d'autres files d'attente du même genre. Cela veut dire que, si un lien de réplication est indisponible pendant une période assez longue, il n'y a aucune structure de données en train de croître en taille dans l'attente d'une reprise de la réplication au redémarrage du lien.

Centres de données multiples avec FIXED_PARTITION

Vous pouvez à présent utiliser une grille FIXED_PARTITION entre au moins deux centres de données. Chaque centre de données a besoin de son propre domaine, en termes de réplication multi-maître. Chaque centre de données peut lire et écrire des données sur le domaine local. Ces modifications seront propagées vers l'autre

centre de données à l'aide des liens que vous aurez définis.

Clients intégralement répliqués

Cette variante de la topologie implique une paire de serveurs eXtreme Scale s'exécutant comme concentrateur. Chaque client crée une grille à conteneur unique autonome avec un catalogue dans sa machine virtuelle Java. Un client utilise sa grille 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 seront réparties rapidement via le concentrateur entre les machines virtuelles Java clients. 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.

Limitations

Les limitations suivantes sont à prendre en compte pour décider s'il y a lieu d'utiliser des topologies de réplication multi-maître et, si oui, quand.

- **Précautions à prendre dans la configuration de chargeurs de classes avec plusieurs domaines**

  Les domaines doivent avoir accès à toutes les classes qui sont utilisées comme clés et comme valeurs. Toutes les dépendances doivent être reflétées dans tous les chemins de classes des machines virtuelles Java conteneurs de grille de la totalité des 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 fait appel à l'arbitre.
- **L'utilisation de chargeurs n'est pas recommandée**

  Les chargeurs peuvent servir à l'interfaçage des modifications entre une grille et une base de données. Il y a fort peu de probabilités que toutes les grilles (domaines) d'une topologie cohabitent géographiquement avec la même base de données. De toute façon, vu les temps d'attente du WAN et d'autres facteurs, ce cas de figure est peu souhaitable.

  Autre problème qui nécessite que l'on aborde la conception avec précaution : le préchargement des grilles. D'ordinaire, lorsqu'elle redémarre, une grille est préchargée à nouveau. Le préchargement n'est pas nécessaire ni même souhaitable lorsqu'on utilise la réplication multi-maître. Dès qu'un domaine est en ligne, il se recharge automatiquement avec le contenu des domaines auxquels il est lié. Il en découle qu'il n'est pas nécessaire de lancer un préchargement manuel d'une grille qui est un domaine dans une topologie de réplication multi-maître.

Les chargeurs obéissent en général à des règles d'insertion et d'actualisation. Dans le cadre de la réplication multi-maître, les insertions doivent être traitées comme des fusions. Lorsque les données sont extraites à distance après le redémarrage d'un domaine, les données existantes seront "insérées" dans le domaine local. Comme il se peut que ces données se trouvent déjà dans la base de données locale, une insertion classique dans la base de données échouera avec une exception de clé en double. Plutôt que des insertions classiques, il faut utiliser une sémantique des fusions.

Il est possible de configurer eXtreme Scale pour qu'il effectue des préchargements en fonction des fragments en utilisant les méthodes preload des plug-in Loader. N'utilisez pas cette technique dans une topologie de réplication multi-maître. Utilisez plutôt un préchargement en fonction du client lorsque la topologie est démarrée (initialement). Autorisez la topologie multi-maître à actualiser les domaines redémarrés à l'aide d'une copie actuelle de ce qui est stocké dans les autres domaines de la topologie. Après que les domaines ont été redémarrés, la responsabilité de leur synchronisation incombera à la topologie multi-maître.

- **Pas de prise en charge d'EntityManager**

  Un groupe de mappes contenant une mappe d'entités n'est pas répliqué entre les domaines.

- **Pas de prise en charge des mappes de tableaux d'octets**

  Un groupe de mappes contenant une mappe qui est configurée avec COPY_TO_BYTES n'est pas répliqué entre les domaines.

- **Pas de prise en charge de l'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 domaines.

JMS pour la répartition des modifications de transaction

Utilisez JMS (Java Message Service) pour les modifications de transaction répartie entre différents groupes de serveurs ou dans des environnements mixtes.

JMS est un protocole idéal pour les modifications réparties entre différents groupes de serveurs ou dans des environnements mixtes. Par exemple, certaines applications qui utilisent eXtreme Scale peuvent être déployées sur IBM WebSphere Application Server Community Edition, Apache Geronimo ou Apache Tomcat alors que d'autres applications peuvent être exécutées sur WebSphere Application Server version 6.x. JMS se prête parfaitement aux modifications réparties entre des homologues eXtreme Scale dans ces environnements différents. Les messages du gestionnaire de haute disponibilité sont transférés très rapidement mais peuvent uniquement répartir les modifications vers les machines virtuelles Java rassemblées dans un groupe central unique. JMS est plus lent mais il autorise le partage d'un ObjectGrid par des ensembles de clients d'applications plus importants et plus variés. JMS est adapté au partage de données dans un ObjectGrid entre un client Swing lourd et une application déployée sur WebSphere Extended Deployment.

Deux fonctionnalités pré-intégrées, le mécanisme d'invalidation client et la réplication entre homologues, sont des exemples de répartition de modifications transactionnelles JMS. Reportez-vous aux informations relatives à la configuration de la réplication entre homologues avec JMS du *Guide d'administration* pour plus de détails.

Implémentation de JMS

JMS est implémenté pour la répartition de modifications transactionnelles à l'aide d'un objet Java qui se comporte comme un ObjectGridEventListener. Cet objet peut propager l'état à l'aide de l'une des quatre méthodes suivantes :

1. Invalidate : toute entrée expulsée, mise à jour ou supprimée est retirée de toutes les machines virtuelles Java homologues à la réception du message.
2. Invalidate conditional : l'entrée est expulsée uniquement si la version locale est identique ou ultérieure à celle disponible sur le diffuseur de publications.
3. Push : toute entrée expulsée, mise à jour, supprimée ou insérée est ajoutée ou écrasée sur toutes les machines virtuelles Java homologues à la réception du message JMS.
4. Push conditional : l'entrée est uniquement mise à jour ou ajoutée côté récepteur si l'entrée locale est moins récente que la version en cours de publication.

Mode écoute pour les modifications de publication

Le plug-in implémente l'interface ObjectGridEventListener pour intercepter l'événement transactionEnd. Lorsque eXtreme Scale appelle cette méthode, le plug-in tente de convertir la liste LogSequence pour toutes les mappes concernées par la transaction en un message JMS et ensuite de la publier. Le plug-in peut être configuré de façon à publier les modifications pour toutes mappes ou un sous-ensemble de mappes. Les objets LogSequence sont traités pour les mappes pour lesquelles la publication est activée. La classe LogSequenceTransformer de l'ObjectGrid sérialise vers un flux une liste LogSequence filtrée pour chaque mappe. Après sérialisation de toutes les listes LogSequence vers le flux, un message ObjectMessage JMS est créé et publié dans une rubrique connue.

Mode écoute pour les messages JMS et application à l'ObjectGrid locale

Le même plug-in lance une unité d'exécution qui tourne en boucle, recevant tous les messages publiés dans la rubrique connue. A l'arrivée d'un message, il transmet le contenu de ce dernier à la classe LogSequenceTransformer au niveau de laquelle il est converti en un ensemble d'objets LogSequence. Une transaction no-write-through est ensuite démarrée. Chaque objet LogSequence est fourni à la méthode Session.processLogSequence qui met à jour les mappes locales pour refléter les modifications. La méthode processLogSequence comprend le mode de répartition. La transaction est validée et le cache local reflète désormais les modifications. Pour plus d'informations sur l'utilisation de JMS pour la répartition de modifications transactionnelles, reportez-vous à la rubrique les informations relatives à la répartition des modifications entre les machines virtuelles Java homologues dans le *Guide d'administration*.

Groupes de mappes pour la réplication

La réplication est activée par l'association de mappes de sauvegarde à un groupe de mappes.

Un groupe de mappes (MapSet) est une collection de mappes qui sont catégorisées par une partition-key. Cette partition-key est dérivée de la clé des mappes individuelles par une opération consistant à prendre son hachage modulo le nombre de partitions. Ainsi, si un groupe de mappes au sein du MapSet a une partition-key X, ces mappes seront stockées dans une partition X correspondante dans la grille, toutes les mappes seront stockées dans la partition Y, et ainsi de

suite. Egalement, les données contenues dans les mappes sont répliquées en fonction des règles définies dans le MapSet, règles qui ne sont utilisées que pour les topologies eXtreme Scale réparties (l'on n'en a pas besoin pour des instances locales).

Voir «Partitionnement», à la page 93 pour plus de détails.

Aux MapSets sont attribués le nombre de partitions qu'ils auront ainsi qu'une règle de réplication. La configuration de la réplication d'un MapSet consiste simplement à identifier le nombre de fragments synchrones et asynchrones que doit avoir le MapSet en plus du fragment primaire. Par exemple, s'il doit y avoir un fragment réplique synchrone et un fragment réplique asynchrone, toutes les mappes de sauvegarde attribuées au MapSet auront chacune un fragment réplique automatiquement réparti au sein de l'ensemble des conteneurs disponibles pour eXtreme Scale. La configuration de la réplication peut également permettre aux clients de lire les données depuis les serveurs répliqués de manière synchrone. Cela peut étaler la charge des demandes de lecture sur d'autres serveurs de la grille eXtreme Scale. La réplication a un impact sur le modèle de modèle de programmation lorsqu'on précharge les mappes de sauvegarde.

Nous allons rentrer dans le détail des diverses options de configuration :

Chapitre 6. Traitement des transactions

Sessions et traitement des transactions

WebSphere eXtreme Scale utilise les transactions comme mécanisme d'interaction avec les données.

Pour interagir avec les données, l'unité d'exécution de votre application requiert sa propre Session. Lorsque l'application souhaite utiliser ObjectGrid sur une unité d'exécution, appelez l'une des méthodes ObjectGrid.getSession pour obtenir une unité d'exécution. Avec la session, l'application peut travailler avec les données stockées dans les mappes ObjectGrid.

Lorsqu'une application utilise un objet Session, la session doit être dans le contexte d'une transaction. Une transaction commence et se valide ou commence et s'annule en utilisant les méthodes begin, commit et rollback sur l'objet Session. Les applications fonctionnent également en mode d'auto-validation : la session commence et valide automatiquement une transaction chaque fois qu'une opération est effectuée sur la mappe. Le mode d'auto-validation ne permet pas de regrouper des opérations multiples en une seule transaction. Il s'agit donc de l'option la plus lente si vous créez un lot d'opérations multiples dans une seule transaction. Cependant, pour les transactions contenant une seule opération, l'auto-validation est l'option la plus rapide.

Transactions

Les transactions disposent de nombreux avantages pour le stockage et la manipulation de données. Vous pouvez utiliser les transactions pour protéger la grille contre les changements simultanés, appliquer plusieurs changements comme unité simultanée, répliquer des données et implémenter un cycle de vie aux verrous appliqués aux changements.

Quand une transaction démarre, WebSphere eXtreme Scale alloue une mappe spéciale des différences pour contenir les changements en cours ou des copies des paires de valeur-clé que la transaction utilise. En règle générale, quand un accès à une paire valeur-clé se produit, la valeur est copiée avant que l'application ne reçoive la valeur. La mappe des différences contrôle tous les changements pour les opérations telles qu'insérer, mettre à jour, obtenir, supprimer, etc. Les clés ne sont pas copiées, car elles sont considérées comme non modifiables. Si un objet ObjectTransformer est spécifié, il est utilisé pour copier la valeur. Si la transaction utilise un verrouillage optimiste, les images précédentes des valeurs sont également suivies afin d'être comparées quand la transaction est validée.

Si une transaction est annulée, les informations de la mappe des différences sont supprimées et les verrous sur les entrées sont retirés. Quand une transaction est validée, les changements sont appliqués à la mappe et les verrous retirés. Si un verrouillage optimiste est utilisé, eXtreme Scale compare les versions des images précédentes des valeurs avec les valeurs qui se trouvent dans la mappe. Ces valeurs doivent correspondre pour que la transaction soit validée. Cette comparaison autorise un plan de verrouillage à version multiple, mais au prix de deux copies créées quand la transaction accède à l'entrée. Toute les valeurs sont à nouveau copiées et la nouvelle copie est stockée dans la mappe. WebSphere

eXtreme Scale crée cette copie pour se protéger contre le fait que l'application change sa référence à la valeur après validation.

Il est possible de ne pas utiliser plusieurs copies des informations. L'application peut enregistrer une copie en utilisant un verrouillage pessimiste au lieu d'un verrouillage optimiste au prix d'une limitation des accès simultanés. La copie de la valeur au moment de la validation peut également être évitée si l'application accepte de ne pas changer la valeur après une validation.

Avantages des transactions

Utilisez les transactions pour les raisons suivantes :

Par le biais des transactions, vous pouvez :

- Annuler les modifications si une exception se produit ou si la logique application requiert l'annulation des changements d'état.
- Pour appliquer plusieurs changements en tant qu'unité atomique au moment de la validation
- Verrouiller et déverrouiller les données afin d'appliquer des changements multiples en tant qu'unité atomique au moment de la validation.
- Protéger une unité d'exécution de modifications simultanées.
- Implémenter un cycle de vie pour les verrous appliqués aux modifications.
- Produire une unité atomique de réplication.

Taille des transactions

Les transactions les plus grandes sont plus efficaces, en particulier pour la réplication. Cependant, les grandes transactions peuvent avoir un impact sur les accès simultanés car les verrous sur entrées sont maintenus plus longtemps. Si vous utilisez de grandes instructions, vous pouvez accroître les performances de réplication. Cette augmentation des performances est important lors du pré-chargement d'une mappe. Essayez différentes tailles de lots pour déterminer ce qui fonctionne le mieux pour votre scénario.

Les grandes transactions aident également avec les programmes de chargement. Si un chargeur utilisé peut effectuer du traitement par lots SQL, alors des gains de performances significatifs peuvent être réalisés sur la transaction, ainsi que des réductions de charges significatives du côté de la base de données. Ces gains de performances dépendent de l'implémentation du chargeur.

Mode de validation automatique

Si aucune transaction n'a démarré activement, quand une application interagit avec un objet ObjectMap, une opération automatique de démarrage et de validation est effectuée pour l'application. Cette opération fonctionne, mais elle empêche l'annulation et le verrouillage de fonctionner efficacement. La vitesse de réplication synchrone est affectée à cause de la très petite taille des transactions. Si vous utilisez une application de gestion des entités, n'utilisez pas le mode de validation automatique car les objets recherchés avec la méthode EntityManager.find deviennent immédiatement non gérés au retour de la méthode et deviennent inutilisables.

Coordinateurs de transactions externes

En règle générale, les transactions commencent avec la méthode session.begin et se termine par la méthode session.commit. Cependant, quand eXtreme Scale est imbriqué, les transactions peuvent être démarrées et terminées par un coordinateur de transactions externes. Si vous en utilisez un, vous n'avez pas besoin d'appeler la méthode session.begin et de terminer avec la méthode session.commit. Pour plus d'informations sur eXtreme Scale et l'interaction des transactions externes, voir *Guide de programmation*. Si vous utilisez WebSphere Application Server, vous pouvez utiliser le plug-in WebSphereTransactionCallback. Voir *Guide de programmation* pour plus d'informations sur les plug-in disponibles avec WebSphere eXtreme Scale.

Attribut CopyMode

Vous pouvez ajuster le nombre de copies en définissant l'attribut CopyMode des objets BackingMap et ObjectMap.

Vous pouvez ajuster le nombre de copies en définissant l'attribut CopyMode des objets BackingMap et ObjectMap. Cet attribut a les valeurs suivantes :

- COPY_ON_READ_AND_COMMIT
- COPY_ON_READ
- NO_COPY
- COPY_ON_WRITE
- COPY_TO_BYTES

COPY_ON_READ_AND_COMMIT est la valeur par défaut. La valeur COPY_ON_READ copie les données initiales récupérées, mais ne copie pas au moment de la validation. Ce mode est sûr si l'application ne modifie pas une valeur après la validation d'une transaction. La valeur NO_COPY ne copie pas de données, ce qui n'est sûr que pour les données en lecture seule. Si les données ne changent jamais, vous n'avez alors pas besoin de les copier pour des raisons d'isolement.

Lorsque vous utilisez la valeur d'attribut NO_COPY, faites attention aux mappes pouvant être mises à jour. WebSphere eXtreme Scale utilise la copie en premier appui pour autoriser l'annulation de la transaction. L'application a changé uniquement la copie, et par conséquent, eXtreme Scale supprime la copie. Si la valeur d'attribut NO_COPY est utilisée et que l'application modifie la valeur validée, il est impossible de procéder à une annulation. La modification de la valeur validée génère des problèmes d'index, de réplication, etc. car les index et les fragments répliques se mettent à jour à la validation de la transaction. Si vous modifiez des données validées puis annulez la transaction, qui n'est pas vraiment annulée, alors les index ne sont pas mis à jour et les réplications ne se produisent pas. D'autres unités d'exécution peuvent voir les changements non validés immédiatement, même s'ils sont verrouillés. Utilisez la valeur d'attribut NO_COPY pour les mappes en lecture seule pour les applications qui effectuent la copie appropriée avant la modification de la valeur. Si vous utilisez la valeur d'attribut NO_COPY et que vous contactez le support technique IBM pour un problème d'intégrité de données, il vous est demandé de reproduire le problème avec le mode de copie défini sur COPY_ON_READ_AND_COMMIT.

La valeur COPY_TO_BYTES stocke les valeurs dans la mappe dans un formulaire sérialisé. Au moment de la lecture, eXtreme Scale gonfle la valeur depuis un

formulaire sérialisé et la stocke dans un autre au moment de la validation. Avec cette méthode, une copie est créée au moment de la lecture et au moment de la validation.

Le mode de copie par défaut pour une mappe est configurable sur l'objet BackingMap. Vous pouvez également changer le mode de copie sur les mappes avant de commencer une transaction en utilisant la méthode ObjectMap.setCopyMode.

Un exemple de fragment de mappe de sauvegarde provenant d'un fichier `objectgrid.xml` qui montre comment définir le mode de copie pour une mappe de sauvegarde donnée suit. Cet exemple part du principe que vous utilisez `cc` comme espace de noms `objectgrid/config`.

```
<cc:backingMap name="RuntimeLifespan" copyMode="NO_COPY"/>
```

Voir les informations concernant les meilleures pratiques de CopyMode dans le *Guide de programmation* pour plus d'informations.

Verrouillage d'entrées de mappe

Une mappe de sauvegarde ObjectGrid prend en charge plusieurs stratégies de verrouillage pour les mappes à des fins de cohérence des entrées de cache.

Chaque mappe de sauvegarde peut être configurée pour utiliser l'une de ces stratégies de verrouillage :

1. mode de verrouillage optimiste
2. mode de verrouillage pessimiste
3. aucune

La stratégie de verrouillage OPTIMISTIC est le mode par défaut. Utilisez le verrouillage optimiste lorsque les données sont modifiées rarement. Les verrous sont uniquement maintenus pendant un laps de temps limité tandis que les données sont lues depuis le cache et copiées dans la transaction. Lorsque le cache de transaction est synchronisé avec le cache principal, tous les objets mis en cache qui ont été mis à jour sont vérifiés avec la version d'origine. Si la vérification échoue, la transaction est annulée et une exception OptimisticCollisionException est provoquée.

La stratégie de verrouillage PESSIMISTIC obtient des verrous pour les entrées de cache et doit être utilisée lorsque les données sont modifiées fréquemment. A chaque lecture d'une entrée de cache, un verrou est obtenu et maintenu de façon conditionnelle jusqu'à la fin de la transaction. La durée de certains verrous peut être paramétrée à l'aide des niveaux d'isolement de transaction pour la session.

Si le verrouillage n'est pas obligatoire car les données ne sont jamais mises à jour ou le sont au cours de période calmes, vous pouvez le désactiver à l'aide de la stratégie de verrouillage NONE. Cette stratégie est très rapide car un gestionnaire de verrou n'est pas requis. La stratégie de verrouillage NONE est idéale pour les tables de recherche et les mappes en lecture seule.

Pour plus d'informations sur les stratégies de verrouillage, consultezles informations concernant les stratégies de verrouillage dans le *Présentation du produit*.

Spécification d'une stratégie de verrouillage

L'exemple ci-dessous illustre comment la stratégie de verrouillage peut être définie sur les mappes de sauvegarde map1, map2 et map3, où chaque mappe utilise une stratégie de verrouillage différente. Le premier fragment de code montre comment utiliser le langage XML pour la configuration des stratégies de verrouillage et le deuxième fragment de code illustre une approche à l'aide d'un programme.

Approche XML

Configuration des mappes de sauvegarde : exemple XML

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

    <objectGrids>
        <objectGrid name="test">
            <backingMap name="map1"
                lockStrategy="PESSIMISTIC" numberOfLockBuckets="31"/>
            <backingMap name="map2"
                lockStrategy="OPTIMISTIC" numberOfLockBuckets="409"/>
            <backingMap name="map3"
                lockStrategy="NONE"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

Approche à l'aide d'un programme

Configuration des mappes de sauvegarde : exemple à l'aide d'un programme

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.LockStrategy;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
...
ObjectGrid og =
 ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("test");
BackingMap bm = og.defineMap("map1");
bm.setLockStrategy( LockStrategy.PESSIMISTIC );
bm.setNumberOfLockBuckets(31);
bm = og.defineMap("map2");
bm.setNumberOfLockBuckets(409);
bm.setLockStrategy( LockStrategy.OPTIMISTIC );
bm = og.defineMap("map3");
bm.setLockStrategy( LockStrategy.NONE );
```

Pour éviter une exception java.lang.IllegalStateException, la méthode setLockStrategy doit être appelée avant d'appeler les méthodes initialize ou getSession sur une instance ObjectGrid locale.

Pour plus d'informations, consultez la rubrique relative aux stratégies de verrouillage dans le *Présentation du produit*.

Configuration du gestionnaire de verrou

Lors de l'utilisation d'une stratégie de verrouillage PESSIMISTIC ou OPTIMISTIC, un gestionnaire de verrou est créé pour la mappe de sauvegarde. Le gestionnaire de verrou utilise une mappe de hachage pour rechercher les entrées verrouillées par une ou plusieurs transactions. S'il existe plusieurs entrées de mappe dans la mappe hash, plus les compartiments de verrouillage sont nombreux, plus les performances sont meilleures. Le risque de collision de synchronisation Java diminue lorsque le nombre de compartiments augmente. Plus les compartiments

de verrouillage sont nombreux, plus les accès simultanés le sont également. Les exemples précédents montrent comment une application peut définir le nombre de compartiments de verrouillage à utiliser pour une instance BackingMap donnée.

Pour éviter une exception java.lang.IllegalStateException, la méthode setNumberOfLockBuckets doit être appelée avant d'appeler les méthodes initialize ou getSession sur une instance ObjectGrid. Le paramètre de la méthode setNumberOfLockBuckets est un entier primitif Java spécifiant le nombre de compartiments de verrouillage à utiliser. L'utilisation d'un nombre primitif peut permettre une distribution uniforme des entrées de mappe entre les compartiments de verrouillage. Pour optimiser les performances, commencez par définir le nombre de compartiments de verrouillage sur environ 10 % du nombre attendu d'entrées BackingMap.

LockDeadlockException

Voici un exemple de code illustrant l'interception de l'exception et le message résultant s'affiche ensuite.

```
try {
...
} catch (ObjectGridException oe) {
System.out.println(oe);
}
```

Le résultat est le suivant :

```
com.ibm.websphere.objectgrid.plugins.LockDeadlockException: _Message
```

Ce message représente la chaîne transmise en tant que paramètre lorsque l'exception est créée et émise.

Cause de l'exception

Le type le plus courant d'exception d'interblocage survient lorsque vous utilisez la stratégie de verrouillage pessimiste et que deux clients distincts possède chacun un verrou partagé sur un objet particulier. Les deux clients tentent ensuite de promouvoir un verrou exclusif sur cet objet. Le schéma ci-dessous illustre cette situation, notamment les blocs de transaction à l'origine de l'exception.

Le fragment de code Java ci-dessous montre comment transmettre un fichier XML de configuration pour créer une ObjectGrid.

Il s'agit d'un résumé de ce qu'il se produit dans votre programme lorsque cette exception survient. Dans une application impliquant plusieurs unités d'exécution mettant à jour la même ObjectMap, il est possible de rencontrer cette situation. Voici un exemple de deux clients exécutant les blocs de code de transaction, tel qu'illustré d dans la figure précédente.

Solutions possibles

Vous pouvez rencontrer la situation illustrée à la Figure 1 lorsque plusieurs unités d'exécution démarrent les transactions sur une mappe particulière. Dans ce cas, l'exception est émise pour éviter le blocage du programme. Vous pouvez définir un rappel et ajouter du code dans le bloc catch pour obtenir plus de détails sur la cause. Etant donné que vous ne rencontrez cette exception que dans une stratégie de verrouillage pessimiste, l'utilisation d'une stratégie de verrouillage optimiste peut tout simplement remédier à la situation. Cependant, si vous avez besoin

d'une stratégie de verrouillage pessimiste, vous pouvez utiliser la méthode getForUpdate et non la méthode get. Vous éviterez ainsi de recevoir des exceptions lors des situations décrites plus haut.

Stratégies de verrouillage

Les stratégies de verrouillage sont de type pessimiste, optimiste ou aucune. Pour choisir une stratégie de verrouillage, vous devez prendre en compte les aspects tels que le pourcentage des types d'opérations, l'utilisation ou non d'un chargeur, etc.

Les verrous sont liés aux transactions. Vous pouvez spécifier les paramètres de verrouillage suivants :

- **Aucun verrouillage** : l'exécution sans verrouillage est la plus rapide. Si vous utilisez des données en lecture seule, vous n'avez peut-être pas besoin de verrouillage.
- **Verrouillage pessimiste** : place des verrous sur les entrées, puis les maintient jusqu'au moment de la validation. Cette stratégie offre une bonne cohérence au prix de la capacité de traitement.
- **Verrouillage optimiste** : prend une image précédente de chaque enregistrement sur lequel la transaction appuie et compare l'image avec les valeurs d'entrées en cours quand la transaction est validée. Si les valeurs d'entrées changent, la transaction est annulée. Aucun verrou n'est maintenu jusqu'au moment de la validation. Cette stratégie de verrouillage offre un meilleur accès simultané que les stratégies pessimistes, au risque que la transaction soit annulée et au prix de la mémoire nécessaire pour une copie supplémentaire de l'entrée.

Définissez la stratégie de verrouillage sur la mappe de sauvegarde. Il n'est pas possible de changer de stratégie pour chaque transaction. Un fragment de code XML suit, montrant comment définir le mode de verrouillage sur une mappe à l'aide du fichier XML, en partant du principe que `cc` est le nom d'espace pour `objectgrid/config` :

```
<cc:backingMap name="RuntimeLifespan" lockStrategy="PESSIMISTIC" />
```

Verrouillage pessimiste

La stratégie de verrouillage pessimiste est à utiliser pour les opérations de mappe en lecture et en écriture lorsqu'aucune autre stratégie de verrouillage n'est possible. Lorsqu'une mappe ObjectGrid est configurée en mode de stratégie de verrouillage pessimiste, un verrou de transaction pessimiste est obtenu pour une entrée de mappe à la première transmission de l'entrée à la mappe de sauvegarde. Le verrou pessimiste est maintenu jusqu'à la fin de la transaction. La stratégie de verrouillage pessimiste est généralement utilisée dans les cas suivants :

- Lorsque la mappe de sauvegarde est configurée avec ou sans chargeur et que les informations sur les versions ne sont pas disponibles.
- Lorsque la mappe de sauvegarde est utilisée directement par une application qui nécessite l'assistance de eXtreme Scale pour le contrôle des accès simultanés.
- Lorsque les informations sur les versions sont disponibles mais que les transactions de mise à jour entrent régulièrement en conflit avec les entrées de sauvegarde, ce qui entraîne des échecs de mise à jour optimiste.

Etant donné que la stratégie de verrouillage pessimiste a un impact majeur sur les performances et l'évolutivité, elle doit uniquement être utilisée pour les mappes en lecture et écriture lorsque les autres stratégies de verrouillage ne sont pas adaptées. Par exemple, ces situations peuvent être : échecs réguliers des mises à jour optimistes ou reprise après échec optimiste difficile à gérer pour une application.

Verrouillage optimiste

La stratégie de verrouillage optimiste présuppose qu'il ne peut se faire que deux transactions tentent d'actualiser la même entrée de mappe au même moment. A partir de ce principe, il n'est pas nécessaire de maintenir le mode de verrouillage pendant tout le cycle de vie de la transaction car il est peu probable que plusieurs transactions puissent actualiser la même entrée de mappe exactement au même moment. La stratégie de verrouillage optimiste est généralement utilisée dans les situations suivantes :

- Lorsqu'une mappe de sauvegarde est configurée avec ou sans chargeur et que les informations sur les versions sont disponibles.
- Lorsqu'une mappe de sauvegarde contient essentiellement des transactions qui exécutent des opérations de lecture. Les opérations insert, update ou remove sur les entrées de mappe ne sont pas fréquentes pour une mappe de sauvegarde.
- Lorsqu'une mappe de sauvegarde est insérée, mise à jour ou supprimée plus fréquemment qu'elle n'est lue mais que les transactions entrent rarement en conflit sur la même entrée de mappe.

A l'instar de la stratégie de verrouillage pessimiste, les méthodes dans l'interface ObjectMap déterminent comment eXtreme Scale tente automatiquement d'obtenir un mode de verrouillage pour l'entrée de mappe à laquelle l'accès est octroyé. Toutefois, les stratégies pessimiste et optimiste diffèrent sur les points suivants :

- Comme la stratégie de verrouillage pessimiste, un mode de verrou S est obtenu par les méthodes get et getAll lorsque la méthode est appelée. En revanche, avec le verrouillage optimiste, le mode de verrou S n'est maintenu que lorsque la transaction s'achève. Le mode de verrou S est libéré avant que la méthode soit renvoyée à l'application. L'objectif d'un mode de verrou consiste en ce que eXtreme Scale vérifie que seules les données validées provenant d'autres transactions soient visibles pour la transaction en cours. Une fois que eXtreme Scale a confirmé la validation des données, le mode de verrou S est libéré. Au moment de la validation, une vérification optimiste des versions est effectuée pour faire en sorte qu'aucune autre transaction n'a modifié l'entrée de mappe après la libération du mode de verrou S par la transaction en cours. Si une entrée n'est pas extraite d'une mappe avant d'être mise à jour, invalidée ou supprimée, l'exécution eXtreme Scale extrait implicitement l'entrée de la mappe. Cette opération get implicite est effectuée pour obtenir la valeur actuelle au moment de la demande de modification de l'entrée.
- Contrairement à la stratégie de verrouillage pessimiste, les méthodes getForUpdate et getAllForUpdate sont traitées exactement comme les méthodes get et getAll de la stratégie de verrouillage optimiste. Un mode de verrou S est obtenu au début de la méthode et est libéré avant d'être renvoyé à l'application.

Toutes les autres méthodes ObjectMap sont traitées exactement comme elles le sont dans la stratégie de verrouillage pessimiste. Lorsque la méthode commit est appelée, un mode de verrou X est obtenu pour toutes les entrées de mappe insérées, mises à jour, supprimées, corrigées ou invalidées et le mode de verrou X est maintenu jusqu'à ce que la transaction effectue le processus de validation.

La stratégie de verrouillage optimiste considère qu'aucune transaction simultanée ne tente de mettre à jour la même entrée de mappe. A partir de ce principe, le mode de verrouillage ne doit pas être maintenu pour le cycle de vie de la transaction car il est peu probable qu'une transaction puisse mettre à jour simultanément la même entrée de mappe. Toutefois, étant donné qu'aucun mode

de verrouillage n'est maintenu, une autre transaction simultanée peut potentiellement mettre à jour l'entrée de mappe après la libération du mode de verrou S par la transaction en cours.

Pour gérer cette possibilité, eXtreme Scale obtient un verrou X au moment de la validation et effectue une vérification optimiste des versions pour faire en sorte qu'aucune autre transaction n'a modifié l'entrée de mappe après la lecture de l'entrée de mappe de la mappe de sauvegarde par la transaction en cours. Si une autre transaction modifie l'entrée de mappe, la vérification de versions échoue et une exception OptimisticCollisionException se produit. Cette exception force l'annulation de la transaction en cours et l'application doit réessayer la transaction entière. La stratégie de verrouillage optimiste s'avère très utile lors qu'une mappe est principalement lue et que les mises à jour de la même entrée de mappe sont peu probables.

Aucun verrouillage

Lorsqu'une mappe de sauvegarde est configurée pour n'utiliser aucune stratégie de verrouillage, la valeur retournée est aucun verrou de transaction pour une entrée de mappe.

La non-utilisation d'une stratégie de verrouillage est utile lorsqu'une application est un gestionnaire de persistance tel qu'un conteneur EJB (Enterprise JavaBeans™) ou qu'une application utilise Hibernate pour obtenir les données persistantes. Dans ce scénario, la mappe de sauvegarde est configurée sans chargeur et le gestionnaire de persistance utilise la mappe de sauvegarde comme cache de données. Dans ce scénario, le gestionnaire de persistance fournit le contrôle des accès simultanés entre les transactions qui accèdent aux mêmes entrées de mappe.

WebSphere eXtreme Scale n'a pas besoin d'obtenir de verrous de transaction à des fins de contrôle des accès simultanés. Cette situation considère que le gestionnaire de persistance ne libère pas ses verrous de transaction avant de mettre à jour la mappe ObjectGrid avec les modifications validées. Si le gestionnaire de persistance libère ses verrous, une stratégie de verrouillage pessimiste ou optimiste doit être utilisée. Par exemple, supposons que le gestionnaire de persistance d'un conteneur d'EJB met à jour une mappe ObjectGrid avec les données validées dans la transaction EJB gérée par conteneur. Si la mise à jour de la mappe ObjectGrid a lieu avant la libération des verrous du gestionnaire de persistance, vous pouvez utiliser la stratégie sans verrou. Si la mise à jour de la mappe ObjectGrid a lieu après la libération des verrous du gestionnaire de persistance, vous devez utiliser la stratégie optimiste ou pessimiste.

La stratégie sans verrou peut être utilisée également lorsque l'application utilise directement une mappe de sauvegarde et qu'un chargeur est configuré pour la mappe. Dans ce scénario, le chargeur utilise le fonction de contrôle des accès simultanés fournies par un système de gestion de base de données relationnelle (SGBDR) à l'aide de la connectivité JDBC (Java Database Connectivity) ou le plug-in Hibernate pour accéder aux données dans une base de données relationnelle. L'implémentation du chargeur peut utiliser une approche optimiste ou pessimiste. Un chargeur qui utilise une approche optimiste de verrouillage ou de gestion des versions contribue à optimiser les performances et l'accès simultané. Pour plus d'informations relatives à l'implémentation d'une approche de verrouillage optimiste, reportez-vous à la section OptimisticCallback de la rubrique les informations relatives aux remarques sur le chargeur dans le*Guide d'administration*. Si vous utilisez un chargeur qui utilise le verrouillage pessimiste d'un programme d'arrière plan, il est conseillé d'utiliser le paramètre forUpdate

transmis à la méthode get de l'interface Loader. Définissez ce paramètre sur true si la méthode getForUpdate de l'interface ObjectMap a été utilisée par l'application pour obtenir les données. Le chargeur peut se servir de ce paramètre pour déterminer s'il convient de demander un verrou pouvant être mis à niveau sur la ligne en cours de lecture. Par exemple, DB2 obtient un verrou pouvant être mis à niveau lorsqu'une instruction SQL select contient une clause `FOR UPDATE`. Cette approche offre la même protection contre les interblocages que celle décrite dans la rubrique«Verrouillage pessimiste», à la page 165.

Pour plus d'informations, reportez-vous à la rubrique relative à la gestion des verrous dans le *Guide de programmation* ou au verrouillage des entrées de mappe dans le *Guide d'administration*.

JMS pour la répartition des modifications de transaction

Utilisez JMS (Java Message Service) pour les modifications de transaction répartie entre différents groupes de serveurs ou dans des environnements mixtes.

JMS est un protocole idéal pour les modifications réparties entre différents groupes de serveurs ou dans des environnements mixtes. Par exemple, certaines applications qui utilisent eXtreme Scale peuvent être déployées sur IBM WebSphere Application Server Community Edition, Apache Geronimo ou Apache Tomcat alors que d'autres applications peuvent être exécutées sur WebSphere Application Server version 6.x. JMS se prête parfaitement aux modifications réparties entre des homologues eXtreme Scale dans ces environnements différents. Les messages du gestionnaire de haute disponibilité sont transférés très rapidement mais peuvent uniquement répartir les modifications vers les machines virtuelles Java rassemblées dans un groupe central unique. JMS est plus lent mais il autorise le partage d'un ObjectGrid par des ensembles de clients d'applications plus importants et plus variés. JMS est adapté au partage de données dans un ObjectGrid entre un client Swing lourd et une application déployée sur WebSphere Extended Deployment.

Deux fonctionnalités pré-intégrées, le mécanisme d'invalidation client et la réplication entre homologues, sont des exemples de répartition de modifications transactionnelles JMS. Reportez-vous aux informations relatives à la configuration de la réplication entre homologues avec JMS du *Guide d'administration* pour plus de détails.

Implémentation de JMS

JMS est implémenté pour la répartition de modifications transactionnelles à l'aide d'un objet Java qui se comporte comme un ObjectGridEventListener. Cet objet peut propager l'état à l'aide de l'une des quatre méthodes suivantes :

1. Invalidate : toute entrée expulsée, mise à jour ou supprimée est retirée de toutes les machines virtuelles Java homologues à la réception du message.
2. Invalidate conditional : l'entrée est expulsée uniquement si la version locale est identique ou ultérieure à celle disponible sur le diffuseur de publications.
3. Push : toute entrée expulsée, mise à jour, supprimée ou insérée est ajoutée ou écrasée sur toutes les machines virtuelles Java homologues à la réception du message JMS.
4. Push conditional : l'entrée est uniquement mise à jour ou ajoutée côté récepteur si l'entrée locale est moins récente que la version en cours de publication.

Mode écoute pour les modifications de publication

Le plug-in implémente l'interface ObjectGridEventListener pour intercepter l'événement transactionEnd. Lorsque eXtreme Scale appelle cette méthode, le plug-in tente de convertir la liste LogSequence pour toutes les mappes concernées par la transaction en un message JMS et ensuite de la publier. Le plug-in peut être configuré de façon à publier les modifications pour toutes mappes ou un sous-ensemble de mappes. Les objets LogSequence sont traités pour les mappes pour lesquelles la publication est activée. La classe LogSequenceTransformer de l'ObjectGrid sérialise vers un flux une liste LogSequence filtrée pour chaque mappe. Après sérialisation de toutes les listes LogSequence vers le flux, un message ObjectMessage JMS est créé et publié dans une rubrique connue.

Mode écoute pour les messages JMS et application à l'ObjectGrid locale

Le même plug-in lance une unité d'exécution qui tourne en boucle, recevant tous les messages publiés dans la rubrique connue. A l'arrivée d'un message, il transmet le contenu de ce dernier à la classe LogSequenceTransformer au niveau de laquelle il est converti en un ensemble d'objets LogSequence. Une transaction no-write-through est ensuite démarrée. Chaque objet LogSequence est fourni à la méthode Session.processLogSequence qui met à jour les mappes locales pour refléter les modifications. La méthode processLogSequence comprend le mode de répartition. La transaction est validée et le cache local reflète désormais les modifications. Pour plus d'informations sur l'utilisation de JMS pour la répartition de modifications transactionnelles, reportez-vous à la rubrique les informations relatives à la répartition des modifications entre les machines virtuelles Java homologues dans le *Guide d'administration*.

Transactions dans une partition unique et transactions dans plusieurs partitions de la grille

La différence majeure entre WebSphere eXtreme Scale et les solutions classiques de stockage de données (bases de données relationnelles ou bases de données en mémoire) consiste en l'utilisation du partitionnement, qui permet au cache d'évoluer de manière linéaire. Les principaux types de transactions sont les transactions impliquant une seule partition et les transactions impliquant toutes les partitions (d'une grille).

Les interactions avec le cache se rangent dans deux catégories : les transactions impliquant une seule partition et les transactions impliquant l'ensemble de la grille. Ces deux types de transactions sont décrits ci-dessous.

Transactions dans une partition unique

Les transactions dans une partition unique constituent le mode préférentiel d'interaction avec les mémoires cache hébergées par WebSphere eXtreme Scale. Lorsqu'une transaction est limitée à une partition, elle se limite par défaut à une seule machine virtuelle Java et donc à un seul serveur. Un serveur peut exécuter un nombre *M* de transactions par seconde. Si votre système contient *N* ordinateurs, vous pouvez exécuter `M*N` transactions par seconde. Si votre activité augmente et que vous ayez besoin de doubler le nombre de transactions par seconde, vous pouvez doubler *N* en installant davantage d'ordinateurs. Vous pouvez alors répondre à vos besoins en capacité sans modifier l'application, mettre à niveau le matériel ni utiliser l'application hors ligne.

Outre qu'elles permettent au cache d'évoluer de manière significative, les transactions s'exécutant dans une seule partition permettent aussi d'optimiser la disponibilité de ce dernier. Chaque transaction est liée à un seul ordinateur. Une défaillance peut se produire sur l'un des autres ordinateurs (N-1) sans que la réussite ou le temps de réponse de la transaction en soit affecté. Si 100 ordinateurs sont en cours d'exécution et que l'un d'eux échoue, 1 % des transactions en cours au moment de l'échec sont annulées. Après cet échec, WebSphere eXtreme Scale relocalise les partitions qui sont hébergées par le serveur défaillant vers les 99 autres ordinateurs. Pendant cette courte période, ces ordinateurs continuent à exécuter des transactions. Seules les transactions impliquant les partitions qui sont en cours de relocalisation sont bloquées. Une fois le basculement terminé, le cache peut continuer à s'exécuter en étant pleinement opérationnel, c'est-à-dire à 99 % de sa capacité de traitement d'origine. Une fois le serveur défaillant remplacé et revenu dans la grille, le cache retrouve 100 % de sa capacité de traitement.

Transactions dans l'ensemble de la grille

En termes de performances, de disponibilité et d'évolutivité, les transactions s'exécutant dans l'ensemble de la grille sont l'opposé des transactions impliquant une seule partition. Elles accèdent à toutes les partitions et donc à chaque ordinateur de la configuration. Chaque ordinateur de la grille est sollicité pour rechercher des données, puis renvoyer le résultat. La transaction n'est pas terminée tant que tous les ordinateurs n'ont pas répondu. La capacité de traitement de la grille est donc limitée par l'ordinateur le plus lent. L'ajout d'ordinateurs ne permet pas d'améliorer la vitesse de l'ordinateur le plus lent ni d'augmenter la capacité de traitement du cache.

Les transactions impliquant l'ensemble de la grille ont des effets semblables sur la disponibilité. Reprenons l'exemple précédent : si 100 serveurs sont en cours d'exécution et que l'un d'eux échoue, 100 % des transactions en cours sont annulées. Après cet échec, WebSphere eXtreme Scale commence à relocaliser les partitions qui sont hébergées par ce serveur vers les 99 autres ordinateurs. En attendant la fin du basculement, la grille ne parvient à traiter aucune de ces transactions. Une fois le basculement terminé, le cache continue à s'exécuter, mais à un niveau de capacité réduit. Si chaque ordinateur de la grille gère 10 partitions, ces 10 ordinateurs reçoivent au moins une partition supplémentaire dans le cadre du basculement. L'ajout d'une partition supplémentaire augmente la charge de travail de cet ordinateur d'au moins 10 %. La capacité de la grille de traitement étant limitée à la capacité de traitement de l'ordinateur le plus lent, cette capacité est limitée en moyenne de 10 %.

Les transactions s'exécutant dans une partition unique sont préférables aux partitions impliquant l'ensemble de la grille pour garantir l'évolutivité d'un cache objet réparti hautement disponible comme WebSphere eXtreme Scale. Pour optimiser les performance de ces systèmes, vous devez utiliser des techniques autres que les méthodologies relationnelles traditionnelles, mais vous pouvez transformer les transactions impliquant l'ensemble de la grille en transactions s'exécutant dans une seule partition.

Méthodes recommandées en matière de génération de modèles de données évolutifs

Les méthodes recommandées pour générer des applications évolutives avec des produits tels que WebSphere eXtreme Scale sont au nombre de deux : les principes fondamentaux et les conseils relatifs à l'implémentation. Les principes fondamentaux sont les grandes idées que vous devez capturer lors de la

conception des données. Une application n'observant pas ces principes aura peu de chance de pouvoir évoluer de manière satisfaisante, même pour les transactions principales. Les conseils d'implémentation s'appliquent aux transactions posant problème dans une application par ailleurs bien conçue et observant les principes généraux relatifs aux modèles de données évolutifs.

Principes fondamentaux

Certains concepts ou principes de base à ne pas oublier constituent les éléments clés pour optimiser l'évolutivité.

Duplication plutôt que normalisation

Il est essentiel de bien comprendre que les produits tels que WebSphere eXtreme Scale sont conçus pour répartir des données dans un grand nombre d'ordinateurs. Si votre objectif est d'exécuter la plupart ou l'ensemble des transactions sur un même ordinateur, le modèle de données doit s'assurer que toutes les données nécessaires à la transaction sont situées dans cette partition. Dans la plupart des cas, la seule manière d'y parvenir consiste à dupliquer les données.

Prenons l'exemple d'un forum électronique. Deux transactions très importantes pour ce forum présentent tous les messages publiés par un utilisateur donné et tous les messages publiés sur un sujet donné. Imaginez tout d'abord comment ces transactions se comporteraient dans le cadre d'un modèle de données normalisé contenant un enregistrement utilisateur, un enregistrement relatif au sujet et un enregistrement relatif au message et contenant le texte réel. Si les messages publiés sont partitionnés avec les enregistrements utilisateur, l'affichage du sujet devient une transaction impliquant l'ensemble de la grille, et inversement. Il est impossible de partitionner les sujets et les utilisateurs car leur relation est de type many-to-many.

La meilleure méthode pour permettre à ce forum électronique d'évoluer est de dupliquer les messages publiés, c'est-à-dire de copier chaque message avec l'enregistrement relatif au sujet et avec l'enregistrement utilisateur. L'affichage des messages publiés à partir d'un utilisateur constitue alors une transaction dans une partition unique, de même que l'affichage des messages publiés dans un sujet, tandis que la mise à jour ou la suppression d'un message publié est une transaction impliquant deux partitions. Ces trois transactions évoluent de manière linéaire au fur et à mesure que le nombre d'ordinateurs de la grille augmente.

L'évolutivité plutôt que les ressources

Le principal obstacle à surmonter en matière de modèles de données dénormalisés est l'impact de ces modèles sur les ressources. La conservation de deux ou trois copies, voire plus, de certaines données risque d'entraîner la consommation d'une trop grande quantité de ressources pour être pratique. Lorsque vous êtes confronté à un tel scénario, gardez ceci en mémoire : les coûts liés à l'achat de matériel diminuent d'année en année. Autre considération, et non des moindres : WebSphere eXtreme Scale permet de supprimer la plupart des coûts associés au déploiement de ressources supplémentaires.

Mesurez les ressources en termes de coût plutôt qu'en termes purement informatiques comme les mégaoctets et les processeurs. Les magasins de données utilisant des données relationnelles normalisées doivent généralement être situés sur le même ordinateur. Cela signifie que vous

devez acheter un seul ordinateur d'entreprise puissant, plutôt que plusieurs machines modestes. Il n'est pas rare qu'un ordinateur d'entreprise pouvant exécuter un million de transactions par seconde soit bien plus onéreux que dix ordinateurs pouvant traiter 100 000 transactions par seconde.

L'ajout de ressources a également un coût. Une entreprise en pleine croissance finit par manquer de capacité. Lorsque vous vous trouvez dans cette situation, vous devez soit arrêter votre système lors du passage à un ordinateur plus rapide et plus puissant, soit créer un second environnement de production. Quelle que soit l'option choisie, vous devrez prendre en charge des coûts supplémentaires liés à la réduction du volume de traitement ou au maintien de la capacité de traitement, pratiquement doublée pendant la durée de la transaction.

Avec WebSphere eXtreme Scale, il n'est pas nécessaire de fermer l'application lors de l'accroissement de la capacité. Si les prévisions relatives à votre entreprise indiquent que vous devez augmenter de 10 % votre capacité pour l'année à venir, augmentez d'autant le nombre d'ordinateurs de la grille. Ce pourcentage peut augmenter sans entraîner d'indisponibilité de l'application et sans que vous ayez à accroître la capacité.

Des transformations de données inutiles

Lorsque vous utilisez WebSphere eXtreme Scale, vous devez stocker les données dans un format directement utilisable par la logique métier. La répartition des données dans un format plus primitif consomme davantage de ressources. La transformation doit se faire lors de l'écriture et lors de la lecture des données. Dans le cas d'une base de données relationnelle, cette transformation est nécessaire car les données sont enregistrées fréquemment sur le disque, mais avec WebSphere eXtreme Scale, elle n'est pas obligatoire. La plupart des données sont stockées en mémoire et peuvent donc être stockées au format requis par l'application.

L'observation de cette règle simple vous aide à dénormaliser les données conformément au premier principe. Le type de transformation des données métier le plus commun est l'opération JOIN nécessaire pour transformer des données normalisées en un ensemble de résultats correspondant aux besoins de l'application. Le stockage des données au format correct permet implicitement d'éviter d'avoir à exécuter ces opérations de type JOIN et génère un modèle de données dénormalisé.

Abandon des requêtes illimitées

Quelle que soit la structure de vos données, les requêtes illimitées évoluent mal. Nous ne vous recommandons par exemple pas d'exécuter une transaction visant à obtenir une liste de tous les articles triés par valeur. Elle peut renvoyer un résultat correct au début, lorsque le nombre d'articles est égal à 1 000, mais lorsque celui-ci atteint 10 millions, la transaction renvoie ces 10 millions d'articles. L'exécution d'une telle transaction risque d'entraîner un délai d'inactivité de celle-ci ou une erreur liée à une insuffisance de mémoire du client.

La meilleure solution consiste à modifier la logique métier de façon que seuls les 10 ou 20 vingt premiers articles soient renvoyés. Cette modification permet de faire en sorte que la transaction soit gérable quel que soit le nombre d'articles présents en cache.

Définition d'un schéma

Le principal avantage lié à la normalisation des données est que la base de données peut prendre en charge la cohérence des données en arrière-plan. Lorsque les données sont dénormalisées à des fins d'évolutivité, la gestion automatique de la cohérence des données disparaît. Pour la rétablir, vous devez implémenter un modèle de données pouvant fonctionner dans la couche d'applications ou en tant que plug-in de la grille répartie.

Prenons l'exemple du forum électronique. Si une transaction supprime un message publié d'un sujet, sa copie dans l'enregistrement utilisateur doit également être supprimée. Sans modèle de données, il est possible que le développeur prévoie la suppression du message publié dans le code de l'application, mais qu'il oublie de le supprimer de l'enregistrement utilisateur. Toutefois, s'il avait utilisé un modèle de données au lieu d'interagir directement avec le cache, la méthode removePost aurait permis d'extraire l'ID utilisateur du message, de rechercher l'enregistrement utilisateur et de supprimer la copie du message en arrière-plan.

Vous pouvez également implémenter un programme d'écoute s'exécutant sur la partition et capable de détecter la modification apportée au sujet et de modifier automatiquement l'enregistrement utilisateur. Un tel programme peut se révéler avantageux car la modification de l'enregistrement utilisateur est effectuée localement si celui-ci se trouve sur la partition ou, s'il se trouve sur une autre partition, la transaction est exécutée entre deux serveurs et non entre le client et le serveur. La connexion réseau entre des serveurs est probablement plus rapide que la connexion existant entre le client et le serveur.

Disparition des conflits

Evitez les scénarios contenant par exemple un compteur global. La grille ne parvient pas à évoluer si un enregistrement unique est utilisé un nombre disproportionné de fois par rapport aux autres enregistrements. Les performances de la grille seront limitées par les performances de l'ordinateur détenant cet enregistrement.

Dans une telle situation, essayez de fractionner l'enregistrement afin qu'il soit géré au niveau de la partition. Prenons le cas d'une transaction renvoyant le nombre total d'entrées présentes dans le cache réparti. Plutôt que d'exécuter une opération d'insertion et de suppression accédant à un enregistrement unique qui s'incrémente, installez un programme d'écoute sur chaque partition pour suivre ces opérations. Grâce à ce suivi, les opérations d'insertion et de suppression deviennent des transactions s'exécutant dans une partition unique.

La lecture du compteur devient une opération impliquant l'ensemble de la grille, mais elle était déjà en grande partie aussi inefficace qu'une telle opération car ses performances étaient liées à celles de l'ordinateur hébergeant l'enregistrement.

Conseils relatifs à l'implémentation

Pour optimiser l'évolutivité, prenez en compte les conseils suivants.

Utilisation des index de recherche inversée

Prenons le cas d'un modèle de données dénormalisé dans lequel les enregistrements client sont partitionnés en fonction du numéro d'ID du client. Cette méthode de partitionnement est un choix logique car pratiquement toutes les opérations métier exécutées avec l'enregistrement client utilisent cet ID. Toutefois, la transaction de connexion, qui est

essentielle, ne l'utilise pas. Les données utilisées pour la connexion sont plus fréquemment le nom d'utilisateur ou l'adresse électronique.

L'approche la plus simple consiste alors à utiliser une transaction impliquant l'ensemble de la grille pour rechercher l'enregistrement client. Comme expliqué précédemment, cette approche n'est pas évolutive.

Autre option : procéder à un partitionnement en fonction du nom d'utilisateur ou de l'adresse électronique. Cette option n'est pas pratique car toutes les opérations liées à l'ID client deviendraient alors des transactions impliquant l'ensemble de la grille. De plus, les clients de votre site pourraient souhaiter modifier leur nom d'utilisateur ou leur adresse électronique. Dans les produits tels que WebSphere eXtreme Scale, la valeur utilisée pour partitionner les données doit rester constante.

La bonne solution consiste alors à utiliser un index de recherche inversée. Avec WebSphere eXtreme Scale, il est possible de créer un cache dans la même grille répartie que le cache contenant tous les enregistrements utilisateur. Ce cache est à haute disponibilité, partitionnée et évolutif. Il permet de mapper un nom d'utilisateur ou une adresse électronique vers un ID client. Plutôt que d'avoir une opération impliquant l'ensemble de la grille, il transforme la procédure de connexion en transaction s'exécutant sur deux partitions. Ce scénario n'est pas aussi optimal qu'une transaction impliquant une seule partition, mais la capacité de traitement augmente cependant de manière linéaire par rapport au nombre d'ordinateurs.

Calcul des valeurs lors de l'écriture

Les calculs les plus fréquents, tels que les moyennes ou les totaux, peuvent consommer une grande quantité de ressources car ils supposent la lecture d'un grand nombre d'entrées. Les lectures étant plus fréquentes que les écritures dans la plupart des applications, il est plus efficace de calculer ces valeurs lors de l'écriture, puis de stocker le résultat dans le cache. Les opérations gagnent ainsi en rapidité et en évolutivité.

Zones facultatives

Prenons l'exemple d'un enregistrement utilisateur contenant un numéro de téléphone professionnel, un numéro de téléphone personnel et un numéro de téléphone portable. Tous ces numéros ou une combinaison d'entre eux (ou aucun) peuvent être définis pour un utilisateur. Si les données ont été normalisées, une table utilisateur et une table contenant les numéros de téléphone existent. Vous pouvez alors identifier les numéros de téléphone d'un utilisateur donné à l'aide d'une opération JOIN entre les deux tables.

Pour dénormaliser cet enregistrement, aucune duplication des données n'est nécessaire, car la plupart des utilisateurs ne partagent pas leurs numéros de téléphone. Au contraire, les emplacements vides doivent être autorisés dans l'enregistrement utilisateur. Au lieu de constituer une table contenant les numéros de téléphone, ajoutez trois attributs à l'enregistrement utilisateur, chacun correspondant à un type de numéro de téléphone. L'ajout de ces attributs rend superflue l'opération JOIN et permet d'effectuer la recherche des numéros de téléphone d'un utilisateur en tant qu'opération impliquant une seule partition.

Positionnement des relations many-to-many

Prenons l'exemple d'une application qui assure le suivi de certains produits et des magasins dans lesquels ceux-ci sont commercialisés. Un même produit est vendu dans plusieurs magasins et un même magasin vend plusieurs produits. Supposons que cette application assure le suivi de 50

revendeurs importants. Chaque produit est vendu dans 50 magasins au maximum, chaque magasin commercialisant des milliers de produits.

Constituez une liste des magasins dans l'entité produit (organisation A) plutôt qu'une liste des produits dans chaque entité magasin (organisation B). Une simple observation des transactions que cette application devrait exécuter permet de comprendre pourquoi l'organisation A est la plus évolutive.

Considérons d'abord les mises à jour. Dans l'organisation A, la suppression d'un produit du stock d'un magasin verrouille l'entité produit. Si la grille contient 10 000 produits, seul 1/10 000è de la grille doit être verrouillé lors de la mise à jour. Dans l'organisation B, la grille contient 50 magasins, si bien qu'1/50è de la grille doit être verrouillé pendant la mise à jour. Même si les deux opérations peuvent être considérées comme des opérations impliquant une seule partition, l'organisation A permet une meilleure évolutivité.

Considérons maintenant les opérations de lecture avec l'organisation A : la recherche des magasins dans lesquels un produit est commercialisé est une transaction impliquant une seule partition, pouvant évoluer et rapide car elle transmet une faible quantité de données. Avec l'organisation B, cette transaction devient une transaction impliquant plusieurs grilles car chaque entité de magasin doit faire l'objet d'un accès permettant d'identifier si le produit est vendu dans ce magasin, ce qui donne un avantage certain à l'organisation A en termes de performances.

Evolutivité avec les données normalisées

Les transactions impliquant l'ensemble de la grille sont utilisées à juste titre pour faire évoluer le traitement des données. Si une grille contient cinq ordinateurs et qu'une transaction visant à trier environ 100 000 enregistrements dans chaque ordinateur est lancée sur l'ensemble de la grille, cette transaction trie 500 000 enregistrements. Si l'ordinateur le plus lent peut traiter 10 de ces transactions par seconde, la grille peut trier 5 millions d'enregistrements par seconde. Si les données de la grille doublent, chaque ordinateur doit trier 200 000 enregistrements et chaque transaction trie 1 million d'enregistrements. Cette progression des données ramène la capacité de traitement de l'ordinateur le plus lent à cinq transactions par seconde et celle de la grille à cinq transactions par seconde. La grille trie toutefois 5 millions d'enregistrements par seconde.

Dans un tel scénario, le fait de doubler le nombre d'ordinateurs permet à chaque ordinateur de revenir à sa charge précédente et à l'ordinateur le plus lent de traiter 10 de ces transactions par seconde. La capacité de traitement de la grille reste la même (10 demandes par seconde), mais chaque transaction traite désormais 1 million d'enregistrements, si bien que la grille a doublé sa capacité en termes de traitement d'enregistrements, atteignant les 10 millions d'enregistrements par seconde.

Avec les applications telles que les moteurs de recherche, qui ont besoin d'évoluer en termes de traitement des données pour s'adapter à la taille croissante de l'Internet et en termes de capacité afin de s'adapter à la croissance du nombre d'utilisateurs, vous devez créer plusieurs grilles avec permutation circulaire des demandes entre les grilles. Si vous avez besoin de faire évoluer la capacité de traitement, ajoutez des ordinateurs et

ajoutez une grille pour gérer les demandes. Si le traitement des données doit évoluer, ajoutez des ordinateurs et conservez le même nombre de grilles.

Chapitre 7. Sécurité

WebSphere eXtreme Scale permet de sécuriser l'accès aux données et l'intégration de fournisseurs de sécurité externes.

Remarque : Dans un magasin de données non mis en cache, une base de données, par exemple, il est probable que certaines fonctions pré-intégrées de sécurité ne vous serviront à rien pour la configuration ou l'activation. Cependant, une fois vos données mises en cache avec eXtreme Scale, vous devez prendre en compte le fait que vos fonctions de sécurité du dorsal ne sont plus actives. Vous pouvez configurer la sécurité de eXtreme Scale aux niveaux nécessaires, de sorte que votre nouvelle architecture mise en cache soit également sécurisée.
Vous trouverez ci-dessous un bref récapitulatif des fonctions de sécurité de eXtreme Scale. Pour des informations plus détaillées sur la configuration de la sécurité, voir *Guide d'administration* et *Guide de programmation*.

Notions de base sur la sécurité répartie

La sécurité répartie eXtreme Scale se base sur trois concepts :

Authentification approuvée
: Possibilité de déterminer l'identité du demandeur. WebSphere eXtreme Scale prend en charge l'authentification client-serveur et serveur-serveur.

Autorisation
: Possibilité d'octroyer des droits d'accès au demandeur. WebSphere eXtreme Scale prend en charge différentes autorisations pour des opérations diverses.

Transfert sécurisé
: Transmission sécurisé des données sur le réseau. WebSphere eXtreme Scale prend en charge les protocoles Transport Layer Security/Secure Sockets Layer (TLS/SSL).

Authentification

WebSphere eXtreme Scale prend en charge les structures de serveurs clients répartis. Une infrastructure de sécurité du serveur client est en place pour sécuriser l'accès aux serveurs eXtreme Scale. Par exemple, lorsque l'authentification est requise par le serveur eXtreme Scale, un client eXtreme Scale doit fournir ses informations d'identification pour s'authentifier sur le serveur. Ces informations peuvent être un nom d'utilisateur et un mot de passe, un certificat client, un ticket Kerberos ou des données présentées dans un format choisi par le client et le serveur.

Autorisation

Les autorisations WebSphere eXtreme Scale sont basées sur des objets et des permissions. Vous pouvez utiliser le service JAAS (Java Authentication and Authorization Services) pour autoriser l'accès, ou vous pouvez choisir une approche personnalisée, telle que Tivoli Access Manager (TAM), pour gérer les autorisations. Les autorisations suivantes peuvent être octroyées à un client ou un groupe :

Autorisation de mappes
Effectuez des opérations d'insertion, de lecture, de mise à jour, d'expulsion ou de suppression sur les mappes.

Autorisation ObjectGrid
Lancez des requêtes sur un objet ou une entité et des requêtes de flux sur les objets ObjectGrid.

Autorisation de l'agent DataGrid
Permet aux agents DataGrid d'être déployés en une base de données ObjectGrid.

Autorisation de mappes côté serveur
Répliquez une mappe de serveur côté client ou créez un index dynamique pour la mappe de serveur.

Autorisation d'administration
Effectuez des tâches d'administration.

Sécurité du transfert

Pour sécuriser la communication du serveur client, WebSphere eXtreme Scale prend en charge les protocoles TLS/SSL. Ces protocoles fournissent une sécurité de couche de transport, avec des fonctions d'authentification, d'intégrité et de confidentialité pour une connexion sécurisée entre le client eXtreme Scale et le serveur.

Sécurité de grille

Dans un environnement sécurisé, un serveur doit être capable de vérifier l'authenticité d'un autre serveur. WebSphere eXtreme Scale utilise un mécanisme de clé secrète partagée dans ce but. Ce mécanisme est similaire à un mot de passe partagé. Tous les serveurs eXtreme Scale s'accordent sur une clé secrète partagée. Lorsqu'un serveur rejoint la grille, le serveur est invité à fournir la clé secrète. Si la clé secrète du serveur tentant de se joindre correspond à la clé sur serveur principal, le serveur peut se joindre à la grille. Dans le cas contraire, la requête de jointure est rejetée.

L'envoi d'une clé secrète en texte clair n'est pas sécurisé. L'infrastructure de sécurité eXtreme Scale fournit un plug-in SecureTokenManager pour permettre au serveur de sécuriser cette clé secrète avant l'envoi. Vous pouvez choisir la façon dont vous souhaitez implémenter l'opération sécurisée. Avec WebSphere eXtreme Scale, une opération sécurisée est implémentée pour chiffrer et signer la clé secrète.

Fonctions de sécurité Java Management Extensions (JMX) dans une topologie de déploiement dynamique

Les fonctions de sécurité JMX MBeans sont prises en charge dans toutes les versions de eXtreme Scale. Les clients des beans gérés de serveur de catalogue et de serveur de conteneur peuvent être authentifiés, et l'accès aux opérations MBean peut être forcé.

Sécurité eXtreme Scale locale

La sécurité eXtreme Scale locale est différente du modèle eXtreme Scale réparti car l'application s'instancie directement et utilise une instance ObjectGrid. Votre application et les instances eXtreme Scale se trouvent dans la même machine virtuelle Java (JVM). Etant donné qu'aucun concept client-serveur n'existe dans ce

modèle, l'authentification n'est pas prise en charge. Vos applications doivent gérer leur propre authentification, puis transmettre l'objet authentifié à eXtreme Scale. Cependant, le mécanisme d'autorisation utilisé pour le modèle de programmation eXtreme Scale est le même que celui utilisé pour le modèle client-serveur.

Configuration et programmation

Pour plus d'informations sur la configuration et la programmation de la sécurité, voir *Guide d'administration* et *Guide de programmation*.

Chapitre 8. 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.

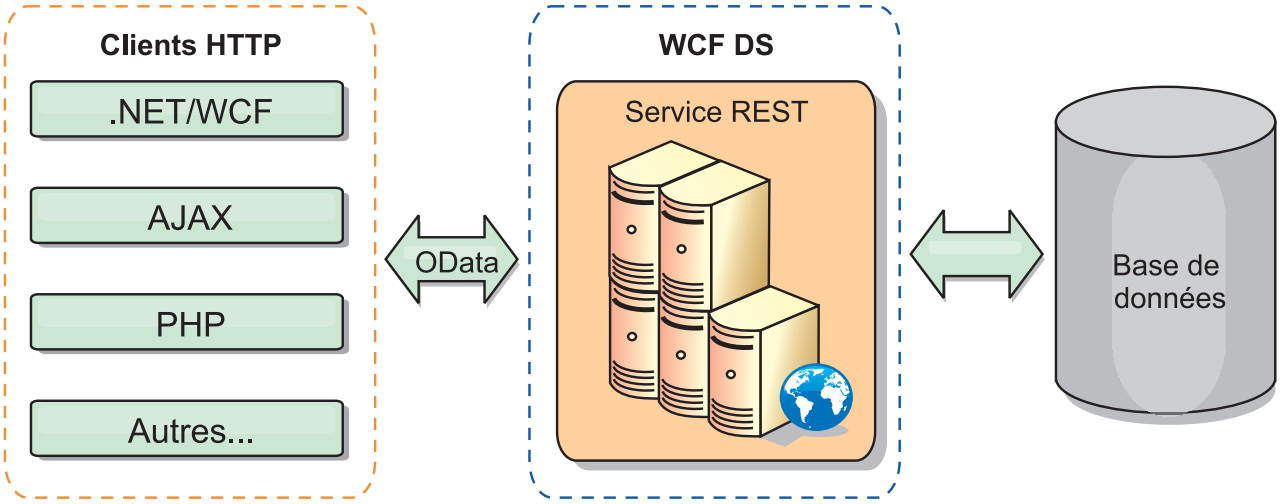


Figure 41. Microsoft WCF Data Services

WebSphere eXtreme Scale inclut un ensemble d'API riche en fonctionnalités pour les clients Java. Comme le montre l'illustration suivante, le service de données REST est une passerelle entre les clients HTTP et la grille WebSphere eXtreme Scale, communiquant avec la grille 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. Il communique avec la grille WebSphere eXtreme Scale à l'aide des 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.

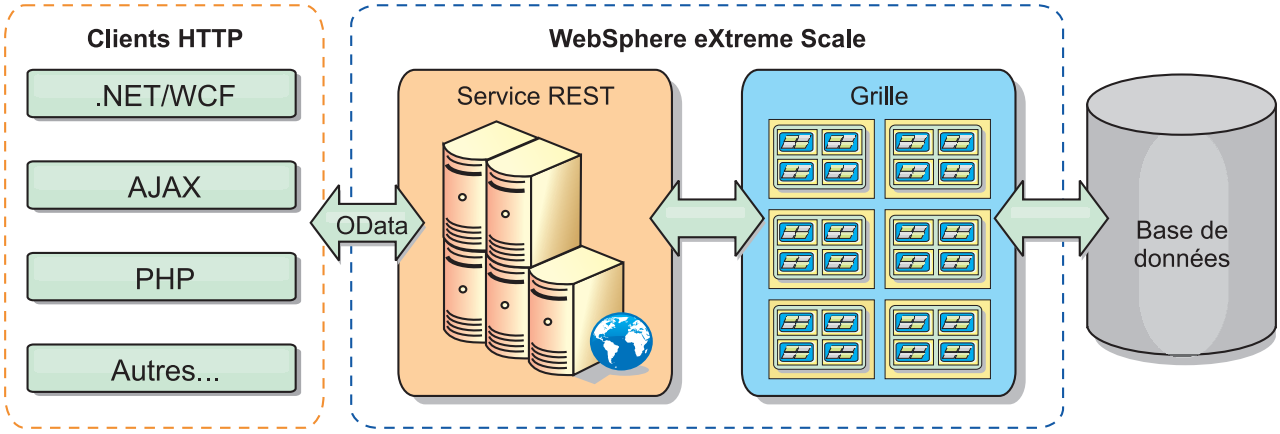


Figure 42. Service de données REST de WebSphere eXtreme Scale

Pour en savoir davantage sur WCF Data Services, reportez-vous à «Exemple et tutoriel sur les services de données REST», à la page 221 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

  Voir Modèles d'entités pour plus d'informations
- 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
- prise en charge des grilles 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.

Chapitre 9. Présentation générale de l'intégration à Spring

Spring est une infrastructure très utilisée pour le développement d'applications Java. WebSphere eXtreme Scale assure une prise en charge permettant à Spring de gérer les transactions eXtreme Scale et de configurer les clients et serveurs qui contiennent votre grille de données internes à la mémoire.

Transactions natives gérées par Spring

Spring fournit des transactions gérées par les conteneurs qui sont similaires à un serveur d'application Java Platform, Enterprise Edition. Cependant, le mécanisme Spring peut se connecter à différentes implémentations. WebSphere eXtreme Scale fournit une intégration du gestionnaire de transactions permettant à Spring de gérer les cycles de vie des transactions ObjectGrid. Voir dans le *Guide de programmation* les explications sur les transactions natives.

Prise en charge des beans d'extension et des espaces de noms gérés par Spring

En outre, eXtreme Scale s'intègre à Spring pour autoriser les beans de style Spring définis pour les points d'extension ou les plug-in. Cette fonction permet d'obtenir des configurations plus complexes et davantage de flexibilité pour la configuration des points d'extension.

En plus des beans d'extension gérés par Spring, eXtreme Scale fournit un espace de noms Spring intitulé "objectgrid". Les beans et les implémentations pré-intégrées sont prédéfinis dans cet espace de noms, ce qui facilite la configuration d'eXtreme Scale pour l'utilisateur. Voir Prise en charge des beans d'extension Spring et des espaces de nom pour des explications plus détaillées, avec un exemple de comment démarrer un serveur conteneur eXtreme Scale à l'aide de configurations Spring.

Prise en charge de la portée du fragment

Avec la configuration classique de style Spring, un bean ObjectGrid peut être de type singleton ou prototype. ObjectGrid prend aussi en charge une nouvelle portée dite "de segment". Si un bean est défini en tant que portée de fragment, seul un bean est créé par fragment. Toutes les demandes de beans avec un ou plusieurs ID correspondant à cette définition dans le même fragment entraînera le renvoi de cette instance de bean par le conteneur Spring.

L'exemple suivant montre un com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl défini avec une portée de fragment. Par conséquent, seule une instance de la classe JPAPropFactoryImpl est créée par fragment.

```
<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl" scope="shard" />
```

Spring Web Flow

Spring Web Flow stocke son état de session dans la session HTTP par défaut. Si une application est configurée pour utiliser eXtreme Scale pour la gestion de session, alors Spring l'utilise automatiquement pour stocker cet état et devient tolérante aux pannes de la même manière que la session.

Encapsulation

Les extensions eXtreme Scale Spring se trouvent dans le fichier `ogspring.jar`. Ce fichier d'archive Java (JAR) doit se trouver sur le chemin de classe pour que la prise en charge de Spring fonctionne. Si une application JEE qui s'exécute dans un WebSphere Extended Deployment augmente le WebSphere Application Server Network Deployment, alors l'application doit placer le fichier `spring.jar` et ses fichiers associés dans les modules de fichiers d'archives d'entreprise. Vous devez également placer le fichier `ogspring.jar` au même emplacement.

Chapitre 10. Tutoriels, exemples et échantillons

Plusieurs tutoriels, exemples et échantillons WebSphere eXtreme Scale sont disponibles.

Tutoriels

Les tutoriels actuellement disponibles sont les suivants.

- Tutoriel sur ObjectMap
- «Tutoriel du gestionnaire d'entités : présentation»
-
-

Exemples

Les rubriques ci-dessous abordent les fonctions clés de WebSphere eXtreme Scale.

- Voir l'exemple d'API de la grille de données dans le *Guide de programmation*
- Voir dans le *Guide d'administration*

Exemples

Les échantillons illustrant la façon d'utiliser les API ObjectGrid dans divers environnements sont fournis avec le produit WebSphere eXtreme Scale.

Articles avec tutoriels et exemples

Tableau 15. Articles disponibles par fonction

Article	**Caractéristiques**
Création d'applications compatibles avec la grille	API ObjectMap, API EntityManager, requête, agents, Java SE et EE, statistiques, partitionnement, administration/opérations, Eclipse
Calcul évolutif de style grille et traitement des données	API EntityManager, agents
Création d'une autre base de données hautes performances évolutive et robuste	API ObjectMap, réplication, partitionnement, administration/opérations, Eclipse
Amélioration de xsadmin pour WebSphere eXtreme Scale	Administration
Redbook : guide d'utilisation	Toutes les rubriques

Tutoriel du gestionnaire d'entités : présentation

Le tutoriel sur le gestionnaire d'entités présente l'utilisation de WebSphere eXtreme Scale pour stocker des informations de commande sur un site Web. Vous pouvez créer une application Java Platform, Standard Edition 5 qui utilise une version d'eXtreme Scale locale en mémoire. Les entités utilisent les annotations et les valeurs génériques de Java SE 5.

Avant de commencer

Assurez-vous de respecter les exigences suivantes avant de commencer le tutoriel :

- Vous devez disposer de Java SE 5.
- Vous devez disposer du fichier `objectgrid.jar` dans le chemin d'accès aux classes.

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

La première étape du tutoriel du gestionnaire d'entités présente la création d'un ObjectGrid local à une entité en créant une classe entité, en enregistrant le type d'entité avec eXtreme Scale et en stockant une instance d'entité dans le cache.

Pourquoi et quand exécuter cette tâche

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 contient l'annotation @Id pour indiquer qu'il s'agit d'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 ce dernier 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 et est conservée à l'aide de la méthode EntityManager.persist, qui identifie l'entité comme étant en attente d'inclusion dans la mappe ObjectGrid.
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, car les valeurs génériques de Java SE 5 ne sont pas utilisées pour vérifier que le fichier `objectgrid.jar` fonctionne sur les machine virtuelle Java Java SE 1.4 et version ultérieure.

Tutoriel du gestionnaire d'entités : formation 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 188. 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 43. 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;
    @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 ObjectQuery

Avec la procédure ci-après, vous pouvez développer un ObjectGrid local en mémoire qui peut stocker des informations de commande pour un site Web et montrer comment utiliser ObjectQuery pour interroger les données de la grille.

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 1.4 (ou ultérieure) simple qui utilise un ObjectGrid local en mémoire.

Procédure

1. «Tutoriel ObjectQuery - Etape 1»
   - Comment créer un ObjectGrid local
   - Comment définir un schéma pour un seul objet à l'aide de l'accès par champ
   - Comment stocker l'objet
   - Comment interroger l'objet avec ObjectQuery
2. «Tutoriel ObjectQuery - Etape 2», à la page 198
   - Comment créer un index utilisable par la requête
3. «Tutoriel ObjectQuery - Etape 3», à la page 199
   - Comment créer un schéma avec deux entités associées
   - Comment stocker des objets avec une référence de clé externe entre eux
   - Comment interroger les objets à l'aide d'une seule requête avec un opérateur JOIN
4. «Tutoriel ObjectQuery - Etape 4», à la page 201
   - Comment créer un schéma avec plusieurs entités associées
   - Comment utiliser l'accès par méthode ou par propriété au lieu de l'accès par champ

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 197 avant de passer à cette étape du tutoriel.

Procédure

Schéma et index

Application.java

```
// Créez 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 198 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 199 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 sur la sécurité Java SE : vue d'ensemble

Le tutoriel suivant vous permet de créer un environnement eXtreme Scale dans un environnement Java Platform, Standard Edition.

Avant de commencer

Assurez-vous que vous connaissez les principes de base d'une configuration eXtreme Scale répartie.

Pourquoi et quand exécuter cette tâche

Dans ce tutoriel, le serveur de catalogue, le serveur de conteneur et le client s'exécutent tous dans un environnement Java SE. Chaque étape du tutoriel est liée à l'étape qui la précède. Effectuez toutes les étapes afin de sécuriser un eXtreme Scale réparti et de développer une application Java SE simple pour accéder au eXtreme Scale sécurisé.

Commencer le tutoriel

Procédure

1. «Tutoriel sur la sécurité Java SE - Etape 1», à la page 204
   - Démarrer un serveur de catalogue non sécurisé
   - Démarrer un serveur de conteneur non sécurisé

- Démarrer un client pour accéder aux données
- Utiliser xsadmin pour afficher la taille de la mappe
- Arrêter le serveur

2. «Tutoriel sur la sécurité Java SE - Etape 2», à la page 207
   - Utiliser CredentialGenerator
   - Utiliser Authenticator
   - Démarrer un serveur de catalogue sécurisé
   - Démarrer un serveur de conteneur sécurisé
   - Démarrer un client pour accéder à l'ObjectGrid sécurisé
   - Utiliser xsadmin pour afficher la taille de la mappe
   - Arrêter le serveur sécurisé
3. «Didacticiel sur la sécurité Java SE - Etape 3», à la page 214
   - Utiliser la politique d'autorisation JAAS
4. «Tutoriel sur la sécurité Java SE - Etape 4», à la page 218
   - Créer un fichier de clés et un fichier de clés certifiées
   - Configurer les propriétés SSL pour le serveur
   - Configurer les propriétés SSL pour le client
   - Utiliser xsadmin pour afficher la taille de la mappe
   - Arrêter le serveur sécurisé

Tutoriel sur la sécurité Java SE - Etape 1

Cette rubrique décrit *un exemple simple non sécurisé*. Des fonctions de sécurité supplémentaires sont ajoutées au fur et à mesure des étapes du tutoriel afin d'augmenter le niveau de sécurité intégrée disponible.

Avant de commencer

Remarque : Tous les fichiers requis pour cette étape du tutoriel sont fournis dans la section suivante.

Procédure

Exécution de l'exemple
Démarrez le service de catalogue en utilisant les scripts suivants. Pour plus d'informations sur le démarrage du service de catalogue, voir les informations concernant le démarrage du service de catalogue dans le *Guide d'administration*.

1. Placez-vous dans le répertoire bin : `cd objectgridRoot/bin`
2. Démarrez un serveur de catalogue nommé catalogServer :
   - UNIX Linux `startOgServer.sh catalogServer`
   - Windows `startOgServer.bat catalogServer`
3. Placez-vous dans le répertoire bin `cd objectgridRoot/bin`
4. Démarrez un serveur de conteneur nommé c0 avec le script suivant :
   - UNIX Linux `startOgServer.sh c0 -objectGridFile ../xml/SimpleApp.xml -deploymentPolicyFile ../xml/SimpleDP.xml -catalogServiceEndpoints localhost:2809`
   - Windows `startOgServer.bat c0 -objectGridFile ../xml/SimpleApp.xml -deploymentPolicyFile ../xml/SimpleDP.xml -catalogServiceEndpoints localhost:2809`

Exemple

Pour plus d'informations sur le démarrage des serveurs de conteneur, voir aux informations relatives au démarrage des processus de conteneur dans le *Guide d'administration*.

Une fois le serveur de catalogue et le serveur de conteneur démarrés, lancez le client comme suit.

1. Placez-vous dans le répertoire bin une nouvelle fois.
2. `java -classpath ../lib/objectgrid.jar;../applib/secsample.jar com.ibm.websphere.objectgrid.security.sample.guide.SimpleApp`

Le fichier `secsample.jar` contient la classe SimpleApp.

La sortie générée par ce programme est la suivante :

```
Le nom de client pour ID 0001 est fName lName
```

Vous pouvez également utiliser xsadmin pour afficher les tailles de mappes de la grille "accounting".

- Placez-vous dans le répertoire `objectgridRoot/bin`.
- Utilisez la commande `xsadmin` avec l'option -mapSizes comme suit.
  - UNIX Linux `xsadmin.sh -g accounting -m mapSet1 -mapSizes`
  - Windows `xsadmin.bat -g accounting -m mapSet1 -mapSizes`

  La sortie suivante s'affiche.

  ```
  Cet utilitaire administratif est fourni à titre d'exemple uniquement et
  ne doit pas être considéré en tant que composant pris en charge par
  WebSphere eXtreme Scale.
  Connexion au service de catalogue localhost:1099
  *********** Résultats pour la grille - accounting, MapSet - mapSet1
  ***********
  *** Liste des mappes pour c0 ***
  Nom de la mappe : customer N° de partition : 0 Taille de la mappe : 1
  Type de fragment : principal
  Nombre de serveurs : 1
  Nombre de domaines : 1
  ```

Arrêt des serveurs

Serveur de conteneur

Utilisez la commande suivante pour arrêter le serveur de conteneur c0.

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

Windows `stopOgServer.bat c0 -catalogServiceEndPoints localhost:2809`

Le message suivant s'affiche.

```
CWOBJ2512I: ObjectGrid server c0 stopped.
```

Serveur de catalogue

Vous pouvez arrêter un serveur de catalogue avec la commande suivante.

UNIX Linux
```
stopOgServer.sh catalogServer -catalogServiceEndPoints
localhost:2809
```

Windows
```
stopOgServer.bat catalogServer -catalogServiceEndPoints
localhost:2809
```

Si vous arrêtez le serveur de catalogue, le message suivant s'affiche.

```
CWOBJ2512I: ObjectGrid server catalogServer stopped.
```

Fichiers requis

Le fichier ci-dessous correspond à la classe Java pour SimpleApp.

SimpleApp.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 (C) COPYRIGHT International Business Machines Corp. 2007-2009
package com.ibm.websphere.objectgrid.security.sample.guide;

import com.ibm.websphere.objectgrid.ClientClusterContext;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;

public class SimpleApp {

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

        SimpleApp app = new SimpleApp();
        app.run(args);
    }

    /**
     * read and write the map
     * @throws Exception
     */
    protected void run(String[] args) throws Exception {
        ObjectGrid og = getObjectGrid(args);

        Session session = og.getSession();

        ObjectMap customerMap = session.getMap("customer");

        String customer = (String) customerMap.get("0001");

        if (customer == null) {
            customerMap.insert("0001", "fName lName");
        } else {
            customerMap.update("0001", "fName lName");
        }
        customer = (String) customerMap.get("0001");

        System.out.println("The customer name for ID 0001 is " + customer);
    }

    /**
     * Get the ObjectGrid
     * @return an ObjectGrid instance
     * @throws Exception
     */
```

```
    protected ObjectGrid getObjectGrid(String[] args) throws Exception {
        ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();

        // Crée un ObjectGrid
        ClientClusterContext ccContext = ogManager.connect("localhost:2809", null, null);
        ObjectGrid og = ogManager.getObjectGrid(ccContext, "accounting");

        return og;

    }

}
```

La méthode getObjectGrid de cette classe obtient un ObjectGrid et la méthode run lit un enregistrement à partir de la mappe client et met à jour la valeur.

Pour exécuter cet échantillon de code dans un environnement réparti, un fichier XML de descripteur d'ObjectGrid, `SimpleApp.xml`, et un fichier XML de déploiement, `SimpleDP.xml`, sont créés. Les fichiers sont inclus dans l'exemple suivant :

SimpleApp.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="accounting">
            <backingMap name="customer" readOnly="false" copyKey="true"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

Le fichier XML suivant permet de configurer l'environnement de déploiement.

SimpleDP.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="accounting">
  <mapSet name="mapSet1" numberOfPartitions="1" minSyncReplicas="0" maxSyncReplicas="2" maxAsyncReplicas="1">
   <map ref="customer"/>
  </mapSet>
 </objectgridDeployment>
</deploymentPolicy>
```

Il s'agit une configuration ObjectGrid simple avec une instance ObjectGrid nommée "accounting" et une mappe nommée "customer" (faisant partie de l'ensemble de mappes "mapSet1"). Le fichier `SimpleDP.xml` contient un ensemble de mappes configuré avec 1 partition et 0 réplique minimum requise.

Etape suivante du tutoriel

Tutoriel sur la sécurité Java SE - Etape 2

Une fois les étapes précédentes effectuées, la rubrique suivante illustre l'implémentation d'une authentification client dans un environnement eXtreme Scale réparti.

Avant de commencer

Assurez-vous d'avoir effectué les étapes du «Tutoriel sur la sécurité Java SE - Etape 1», à la page 204.

Pourquoi et quand exécuter cette tâche

Une fois l'authentification client activée, un client est authentifié avant de se connecter au serveur eXtreme Scale. Cette section illustre comment activer l'authentification client dans un environnement de serveur eXtreme Scale et inclut des extraits de code et des scripts.

Comme tout autre mécanisme d'authentification, l'authentification se compose au minimum des étapes suivantes :

1. L'administrateur modifie la configuration afin de rendre l'authentification obligatoire.
2. Le client fournit des données d'identification au serveur.
3. Le serveur compare les données d'identification fournies au registre.

Procédure

1. **Données d'identification client**

   Les données d'identification client sont représentées par une interface com.ibm.websphere.objectgrid.security.plugins.Credential. Les données d'identification client peuvent être une paire nom-mot de passe, un ticket Kerberos, un certificat client ou des données au format convenu par le client et le serveur. Pour plus d'informations, référez-vous à la documentation sur l'API de données d'identification.

   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.

   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 est utilisé pour générer des données d'identification client. Cela est utile en cas de données d'identification temporaires. Dans ce cas, la méthode getCredential() est appelée pour renouveler les données d'identification. Pour plus d'informations, référez-vous à la documentation sur l'API CredentialGenerator.

   Vous pouvez implémenter ces deux interfaces pour l'exécution du client eXtreme Scale afin d'obtenir les données d'identification client.

   Cet exemple utilise les deux extraits d'implémentation de plug-in suivants fournis par eXtreme Scale.

   ```
   com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential
   com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator
   ```

   Pour plus d'informations sur ces plug-in, reportez-vous à la rubrique sur la programmation de l'authentification de client dans le *Guide de programmation*.

2. **Authentificateur de serveur** 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 eXtreme Scale authentifie l'objet Credential avant de traiter la requête. 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 le 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 autoriser la requête, ce qui sera illustré plus loin.

Un serveur eXtreme Scale utilise le plug-in Authenticator pour authentifier l'objet Credential. Référez-vous à la documentation sur l'API de l'Authenticator pour plus de détails.

Cet exemple utilise une implémentation pré-intégrée à eXtreme Scale, KeyStoreLoginAuthenticator, utilisée à des fins de test et d'exemple (un fichier de clés est un registre d'utilisateurs et ne doit pas être utilisé pour la production). Programmation de l'authentification de client dans le *Guide de programmation*.

Cet KeyStoreLoginAuthenticator utilise un KeyStoreLoginModule pour authentifier l'utilisateur avec le fichier de clés par le biais du module de connexion JAAS "KeyStoreLogin". Le fichier de clés peut être configuré en tant qu'option de la classe KeyStoreLoginModule. L'exemple suivant illustre l'alias keyStoreLogin configuré dans le fichier de configuration JAAS og_jaas.config :

```
KeyStoreLogin{
com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginModule required
     keyStoreFile="../security/sampleKS.jks" debug = true;
};
```

Les commandes suivantes créent un fichier de clés sampleKS.jks dans le répertoire %OBJECTGRID_HOME%/security avec le mot de passe sampleKS1. De plus, ces trois certificats utilisateur représentant les utilisateurs administrator, manager et cashier sont créés avec leur propre mot de passe.

a. Placez-vous dans le répertoire racine eXtreme Scale.

```
cd objectgridRoot
```

b. Créez un répertoire nommé "security".

```
mkdir security
```

c. Placez-vous dans le répertoire security que vous venez de créer.

```
cd security
```

d. Utilisez la commande keytool (sous le répertoire javaHOME/bin) pour créer un utilisateur "administator" avec le mot de passe "administrator1" dans le fichier de clés sampleKS.jks.

```
keytool -genkey -v -keystore ./sampleKS.jks -storepass sampleKS1
-alias administrator -keypass administrator1
-dname CN=administrator,O=acme,OU=OGSample -validity 10000
```

e. Utilisez la commande keytool (sous le répertoire javaHOME/bin) pour créer un utilisateur "administrator" avec le mot de passe "administrator1" dans le fichier de clés sampleKS.jks.

```
keytool -genkey -v -keystore ./sampleKS.jks -storepass sampleKS1
-alias manager -keypass manager1
-dname CN=manager,O=acme,OU=OGSample -validity 10000
```

f. Utilisez la commande keytool (sous le répertoire javaHOME/bin) pour créer un utilisateur "cashier" avec le mot de passe "cashier1" dans le fichier de clés sampleKS.jks.

```
keytool -genkey -v -keystore ./sampleKS.jks -storepass sampleKS1
-alias cashier -keypass cashier1 -dname CN=cashier,O=acme,OU=OGSample
-validity 10000
```

La configuration des paramètres de sécurité client est définie dans le fichier de propriétés client. Utilisez la commande suivante pour créer une copie dans le répertoire %OBJECTGRID_HOME%/security :

a. Placez-vous dans le répertoire security.

```
cd objectgridRoot/security
```

b. Copiez le fichier sampleClient.properties vers le fichier client.properties.

```
cp ../properties/sampleClient.properties client.properties
```

Les propriétés suivantes sont mises en évidence dans le fichier client.properties situé sous le répertoire security.

a. **securityEnabled :** définissez securityEnabled sur true (valeur par défaut) ; cela active la sécurité client, ce qui inclut l'authentification.

b. **credentialAuthentication :** définissez credentialAuthentication sur Supported (valeur par défaut) ; cela signifie que le client prend en charge l'authentification des données d'identification.

c. **transportType :** définissez transportType sur TCP/IP, ce qui signifie qu'aucune couche Secure Sockets Layer ne sera utilisée.

d. **singleSignOnEnabled :** définissez ce paramètre sur false (valeur par défaut). La connexion unique (Single sign-on) n'est pas disponible.

3. **Configuration des paramètres de sécurité du serveur**

La configuration des paramètres de sécurité du serveur est définie dans le fichier XML du descripteur de sécurité et dans le fichier des propriétés de sécurité du serveur.Le fichier XML du descripteur de sécurité décrit les propriétés de sécurité communes à tous les serveurs (y compris les serveurs de catalogue et les serveurs de conteneur). Un exemple de propriété est la configuration de l'authentificateur qui représente le registre utilisateur et le mécanisme d'authentification.

Voici le fichier `security.xml` à utiliser pour cet exemple :

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

 <security securityEnabled="true" loginSessionExpirationTime="300" >

        <authenticator className ="com.ibm.websphere.objectgrid.security.plugins.builtins.
        KeyStoreLoginAuthenticator">
        </authenticator>
    </security>

</securityConfig>
```

a. **securityEnabled :** définissez ce paramètre sur true, ce qui active la sécurité du serveur, dont la fonction d'authentification.

b. **loginSessionExpirationTime :** définissez la valeur sur 300 (valeur par défaut).

c. **authenticator :** ajoutez la classe d'authentificateur KeyStoreLoginAuthenticator au fichier XML du cluster, comme suit :

```
<authenticator className ="com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginAuthenticator">
        </authenticator>
```

d. **credentialAuthentication :** définissez l'attribut credentialAuthentication sur Required afin d'activer l'authentification sur le serveur

Pour des explications plus détaillées sur le fichier `security.xml`, voir les informations relatives au fichier XML du descripteur de sécurité dans le *Guide d'administration*.

Copiez le fichier des propriétés du serveur dans le répertoire security. Pour le moment, aucune modification n'est requise dans le fichier.

a. Placez-vous dans le répertoire security.

```
cd objectgridRoot/security
```

b. Copiez le fichier d'exemple `sampleServer.properties` de l'objectGrid, du répertoire property vers le fichier `server.properties`.

```
cp ../properties/containerServer.properties server.properties
```

Effectuez les modifications suivantes dans le fichier `server.properties` :

a. **securityEnabled :** définissez l'attribut **securityEnabled** sur true.

b. **transportType :** définissez l'attribut **transportType** sur TCP/IP, ce qui signifie qu'aucune couche Secure Sockets Layer ne sera utilisée.

c. **secureTokenManagerType :** définissez l'attribut **secureTokenManagerType**sur sur none pour ne pas configurer le gestionnaire des jetons sécurisés.

4. **Client sécurisé** Connectez l'application client au serveur en toute sécurité, tel qu'illustré dans l'exemple suivant :

```
// 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 (C) COPYRIGHT International Business Machines Corp. 2007-2009
package com.ibm.websphere.objectgrid.security.sample.guide;

import com.ibm.websphere.objectgrid.ClientClusterContext;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.security.config.ClientSecurityConfiguration;
import com.ibm.websphere.objectgrid.security.config.ClientSecurityConfigurationFactory;
import com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator;
import com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator;

public class SecureSimpleApp extends SimpleApp {

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

        SecureSimpleApp app = new SecureSimpleApp();
        app.run(args);
    }

    /**
     * Get the ObjectGrid
     * @return an ObjectGrid instance
     * @throws Exception
     */
    protected ObjectGrid getObjectGrid(String[] args) throws Exception {
        ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
        ogManager.setTraceFileName("logs/client.log");
        ogManager.setTraceSpecification("ObjectGrid*=all=enabled:ORBRas=all=enabled");

        // crée un objet ClientSecurityConfiguration à l'aide du fichier spécifié
        ClientSecurityConfiguration clientSC = ClientSecurityConfigurationFactory
                .getClientSecurityConfiguration(args[0]);

        // crée un CredentialGenerator en utilisant le nom de l'utilisateur et le mot de
        // passe fournis.
        CredentialGenerator credGen = new UserPasswordCredentialGenerator(args[1], args[2]);
        clientSC.setCredentialGenerator(credGen);

        // crée un ObjectGrid en se connectant au serveur de catalogue
        ClientClusterContext ccContext = ogManager.connect("localhost:2809", clientSC, null);
        ObjectGrid og = ogManager.getObjectGrid(ccContext, "accounting");

        return og;

    }

}
```

Trois éléments diffèrent par rapport à l'application non sécurisée :

a. A créé un objet ClientSecurityConfiguration en fournissant le fichier `client.properties` configuré.

b. A créé un UserPasswordCredentialGenerator en utilisant l'ID utilisateur et le mot de passe fournis.

c. S'est connecté au serveur de catalogue pour obtenir un ObjectGrid auprès du ClientClusterContext en fournissant un objet ClientSecurityConfiguration.

5. **Exécution de l'application**

Pour exécuter l'application, démarrez le serveur de catalogue. Utilisez les options -clusterFile et -serverProps de la ligne de commande pour indiquer les propriétés de sécurité :

a. Placez-vous dans le répertoire bin :

```
cd objectgridRoot/bin
```

b. Lancez le serveur de catalogue :

- UNIX Linux

```
startOgServer.sh catalogServer -clusterSecurityFile ../security/security.xml
-serverProps ../security/server.properties -jvmArgs
-Djava.security.auth.login.config="../security/og_jaas.config"
```

- Windows

```
startOgServer.bat catalogServer -clusterSecurityFile ../security/security.xml
-serverProps ../security/server.properties -jvmArgs
-Djava.security.auth.login.config="../security/og_jaas.config"
```

Lancez ensuite un serveur de conteneur sécurisé en utilisant le script suivant :

a. Placez-vous de nouveau dans le répertoire bin :

```
cd objectgridRoot/bin
```

b. Lancez un serveur de conteneur sécurisé :

- Linux UNIX

```
startOgServer.sh c0 -objectgridFile ../xml/SimpleApp.xml
-deploymentPolicyFile ../xml/SimpleDP.xml
-catalogServiceEndpoints localhost:2809
-serverProps ../security/server.properties
-jvmArgs -Djava.security.auth.login.config="../security/og_jaas.config"
```

- Windows

```
startOgServer.bat c0 -objectgridFile ../xml/SimpleApp.xml
-deploymentPolicyFile ../xml/SimpleDP.xml
-catalogServiceEndpoints localhost:2809
-serverProps ../security/server.properties
-jvmArgs -Djava.security.auth.login.config="../security/og_jaas.config"
```

Le fichier de propriétés du serveur est indiqué par le biais de l'option -serverProps.

Une fois le serveur démarré, lancez le client en utilisant la commande suivante :

a. `cd objectgridRoot/bin`

b.

```
java -classpath ../lib/objectgrid.jar;../applib/secsample.jar
  com.ibm.websphere.objectgrid.security.sample.guide.SecureSimpleApp
  ../security/client.properties manager manager1
```

Le fichier `secsample.jar` contient la classe SimpleApp.

L'application SecureSimpleApp utilise les trois paramètres suivants :

a. Le fichier `../security/client.properties` est le fichier de propriétés de la sécurité client.

b. `manager` est l'ID utilisateur.

c. `manager1` est le mot de passe.

Une fois la classe publiée, la sortie est la suivante :

Le nom du client pour `ID 0001 est fName lName`.

Vous pouvez également utiliser xsadmin pour afficher les tailles de mappe de la grille "accounting".

- Placez-vous dans le répertoire `objectgridRoot/bin`.
- Utilisez la commande `xsadmin` avec l'option -mapSizes comme suit.

– UNIX Linux `xsadmin.sh -g accounting -m mapSet1 -username manager -password manager1 -mapSizes`

– Windows `xsadmin.bat -g accounting -m mapSet1 -username manager -password manager1 -mapSizes`

La sortie suivante s'affiche.

```
Cet utilitaire administratif est fourni à titre d'exemple uniquement
et ne doit pas être considéré en tant que composant pris en charge par
WebSphere eXtreme Scale.

Connexion au service de catalogue localhost:1099

********** Résultats pour la grille - accounting, MapSet - mapSet1
**********

*** Liste des mappes pour c0 ***

Nom de la mappe : customer N° de partition : 0 Taille de la mappe : 1
Type de fragment : principal

Nombre de serveur : 1

Nombre de domaines : 1
```

Vous pouvez à présent utiliser la commande stopOgServer pour arrêter le processus du serveur de conteneur ou du service de catalogue. Il est néanmoins nécessaire de fournir un fichier de configuration des paramètres de sécurité. L'exemple de fichier de propriétés du client définit les deux propriétés suivantes pour générer un ID d'utilisateur et un mot de passe (manager/manager1).

```
credentialGeneratorClass=com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator
credentialGeneratorProps=manager manager1
```

Arrêtez le conteneur c0 avec la commande suivante.

- UNIX Linux `stopOgServer.sh c0 -catalogServiceEndPoints localhost:2809 -clientSecurityFile ..\security\client.properties`
- Windows `stopOgServer.bat c0 -catalogServiceEndPoints localhost:2809 -clientSecurityFile ..\security\client.properties`

Si vous n'utilisez pas l'option -clientSecurityFile, un message d'exception s'affiche.

```
>> SERVER (id=39132c79, host=9.10.86.47) TRACE START:

>> org.omg.CORBA.NO_PERMISSION : le serveur requiert une
authentification par données d'identification mais aucun contexte de
sécurité n'est fourni par le client. Cela est généralement dû au fait
que le client ne fournit pas de données d'identification au serveur.

vmcid: 0x0

code mineur : 0

terminé : non
```

Vous pouvez également arrêter le serveur de catalogue à l'aide de la commande suivante. Cependant, si vous souhaitez effectuer la prochaine étape du tutoriel, vous pouvez maintenir l'exécution du serveur de catalogue.

- UNIX Linux `stopOgServer.sh catalogServer -catalogServiceEndPoints localhost:2809 -clientSecurityFile ..\security\client.properties`
- Windows `stopOgServer.bat catalogServer -catalogServiceEndPoints localhost:2809 -clientSecurityFile ..\security\client.properties`

Si vous arrêtez le serveur de catalogue, la sortie suivante s'affiche.

```
CWOBJ2512I: ObjectGrid server catalogServer stopped
```

Votre système est à présent partiellement sécurisé grâce à l'activation de l'authentification. Vous avez configuré le serveur pour activer le registre utilisateur, configuré le client pour fournir des données d'identification client et modifié le fichier de propriétés du client et le fichier XML du cluster pour activer l'authentification.

Si vous avez fourni un mot de passe non valide, une exception s'affiche et indique que le nom d'utilisateur ou le mot de passe est incorrect.

Pour plus d'informations sur l'authentification client, voir les informations concernant l'authentification de client de l'application dans le *Guide d'administration*.

Etape suivante du tutoriel

Didacticiel sur la sécurité Java SE - Etape 3

Après avoir authentifié un client, comme dans l'étape précédente, vous pouvez attribuer des privilèges de sécurité par le biais des mécanismes d'autorisation eXtreme Scale.

Avant de commencer

Vous devez avoir terminé «Tutoriel sur la sécurité Java SE - Etape 2», à la page 207 avant d'effectuer cette tâche.

Pourquoi et quand exécuter cette tâche

Au cours de l'étape précédente, vous avez appris à activer l'authentification dans une grille eXtreme Scale. Par conséquent, aucun client non authentifié ne peut se connecter à votre serveur ni soumettre des requêtes à votre système. Toutefois, tous les clients authentifiés possèdent les mêmes permissions ou privilèges liés au serveur, tels que la lecture, l'écriture ou la suppression des données stockées dans les mappes ObjectGrid. Les clients peuvent également soumettre tout type de requête. Cette section explique comment utiliser l'autorisation eXtreme Scale pour attribuer différents privilèges aux utilisateurs authentifiés.

A l'instar de nombreux autres systèmes, eXtreme Scale adopte un mécanisme d'autorisation basé sur les permissions. WebSphere eXtreme Scale permet d'utiliser plusieurs catégories de permission, chacune étant représentée par une classe distincte. Cette rubrique décrit MapPermission. Pour connaître la liste complète des catégories de permissions, voir la section relative à l'autorisation du client dans le *Guide de programmation*.

Dans WebSphere eXtreme Scale, la classe com.ibm.websphere.objectgrid.security.MapPermission représente les permissions liées aux ressources eXtreme Scale, notamment les méthodes des interfaces ObjectMap ou JavaMap. WebSphere eXtreme Scale définit les chaînes de permission suivantes pour accéder aux méthodes des interfaces ObjectMap et JavaMap :

- read : accorde la permission de lire les données de la mappe.
- write : accorde la permission de mettre à jour les données de la mappe.
- insert : accorde la permission d'insérer les données dans la mappe.
- remove : accorde la permission de supprimer les données de la mappe.
- invalidate : accorde la permission d'invalider les données de la mappe.

- all : accorde les permissions de lire, d'écrire, d'insérer, de supprimer et d'invalider.

L'autorisation est accordée lorsque le client invoque une méthode de l'interface ObjectMap ou JavaMap. Le moteur d'exécution eXtreme Scale vérifie différentes permissions de mappe pour différentes méthodes. Si les permissions nécessaires ne sont pas accordées au client, une AccessControlException se produit.

Ce didacticiel indique comment utiliser l'autorisation JAAS (Java Authentication and Authorization Service) afin d'autoriser plusieurs utilisateurs à accéder à la mappe.

Procédure

1. **Activation de l'autorisation eXtreme Scale**. Pour activer l'autorisation sur l'ObjectGrid, vous devez définir l'attribut securityEnabled sur `true` pour cet ObjectGrid spécifique dans le fichier XML. L'activation de la sécurité sur cet ObjectGrid revient à activer l'autorisation. Utilisez les commandes suivantes pour créer un fichier XML ObjectGrid en activant la sécurité.
   a. Accédez au dossier `bin`.
      ```
      cd objectgridRoot/bin
      ```
   b. Copiez le fichier `SimpleApp.xml` dans le fichier `SecureSimpleApp.xml`.
      ```
      cp SimpleApp.xml SecureSimpleApp.xml
      ```
   c. Ouvrez le fichier `SecureSimpleApp.xml` et ajoutez `securityEnabled="true"` au niveau de l'ObjectGrid comme indiqué dans la syntaxe XML 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" securityEnabled="true">
                  <backingMap name="customer" readOnly="false" copyKey="true"/>
              </objectGrid>
          </objectGrids>
      </objectGridConfig>
      ```
2. **Définition de la politique d'autorisation.** Dans la section d'authentification pré-client, vous avez créé trois utilisateurs dans le fichier de clés : cashier (caissier), manager (gestionnaire) et administrator (administrateur). Dans cet exemple, l'utilisateur "cashier" dispose uniquement de la permission de lecture sur toutes les mappes. L'utilisateur "manager", quant à lui, dispose de toutes les permissions. L'autorisation JAAS est utilisée dans cet exemple. L'autorisation JAAS utilise le fichier de politique d'autorisation pour accorder les permissions aux principaux. Le fichier est défini dans le répertoire de sécurité :
   ```
   grant codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
       principal javax.security.auth.x500.X500Principal "CN=cashier,O=acme,OU=OGSample" {
       permission com.ibm.websphere.objectgrid.security.MapPermission "accounting.*", "read ";
   };

   grant codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
       principal javax.security.auth.x500.X500Principal "CN=manager,O=acme,OU=OGSample" {
       permission com.ibm.websphere.objectgrid.security.MapPermission "accounting.*", "all";
   };
   ```

   Remarque :

   - La `base de code "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"` est une URL réservée spécifiquement à l'ObjectGrid. Toutes les permissions ObjectGrid accordées aux principaux utilisent cette base de code spécifique.
   - La première déclaration d'attribution accorde la permission de lecture ("read") de mappe à l'utilisateur principal `"CN=cashier,O=acme,OU=OGSample"`,

de sorte que le caissier dispose uniquement de la permission de lecture sur toutes les mappes de l'ObjectGrid accounting.

- La deuxième déclaration d'attribution accorde toutes ("all") les permissions à l'utilisateur principal `"CN=manager,O=acme,OU=OGSample"`, de sorte que le gestionnaire dispose de toutes les permissions sur toutes les mappes de l'ObjectGrid accounting.

Vous pouvez désormais démarrer un serveur avec une politique d'autorisation. Le fichier de politique d'autorisation JAAS peut être défini à l'aide de la propriété -D standard : -Djava.security.auth.policy=../security/ogAuth.policy

3. **Exécutez l'application.**

   Après avoir créé les fichiers mentionnés ci-dessus, vous pouvez exécuter l'application.

   Démarrez le serveur de catalogue à l'aide des commandes suivantes. Pour de plus amples informations sur le démarrage du service de catalogue, voir les informations relatives au démarrage d'un catalogue de service dans le *Guide d'administration*.

   a. Accédez au répertoire bin : `cd objectgridRoot/bin`

   b. Démarrez le serveur de catalogue.

      - UNIX Linux
      ```
      startOgServer.sh catalogServer
      -clusterSecurityFile ../security/security.xml -serverProps
      ../security/server.properties -jvmArgs
      -Djava.security.auth.login.config="../security/og_jaas.config"
      ```
      - Windows
      ```
      startOgServer.bat catalogServer -clusterSecurityFile
      ../security/security.xml -serverProps ../security/server.properties
      -jvmArgs -Djava.security.auth.login.config="../security/
      og_jaas.config"
      ```

      Vous avez créé les fichiers `security.xml` et `server.properties` au cours de l'étape précédente de ce didacticiel.

      T

   c. Vous pouvez démarrer un serveur de conteneur sécurisé à l'aide du script suivant. Exécutez le script suivant à partir du répertoire bin :

      - UNIX Linux
      ```
      # startOgServer.sh c0 -objectGridFile
      ../xml/SecureSimpleApp.xml -deploymentPolicyFile
      ../xml/SimpleDP.xml -catalogServiceEndpoints localhost:2809
      -serverProps ../security/server.properties -jvmArgs
      -Djava.security.auth.login.config="../security/og_jaas.config"
      -Djava.security.auth.policy="../security/og_auth.policy"
      ```
      - Windows
      ```
      startOgServer.bat c0 -objectGridFile ../xml/
      SecureSimpleApp.xml -deploymentPolicyFile ../xml/SimpleDP.xml
      -catalogServiceEndpoints localhost:2809 -serverProps
      ../security/server.properties -jvmArgs
      -Djava.security.auth.login.config="../security/og_jaas.config"
      -Djava.security.auth.policy="../security/og_auth.policy"
      ```

   Veuillez noter les différences suivantes concernant la commande de démarrage de serveur de conteneur précédente :

   - Utilisez le fichier `SecureSimpleApp.xml` au lieu du fichier `SimpleApp.xml`.
   - Ajoutez un autre argument `-Djava.security.auth.policy` pour définir le fichier de police d'autorisation JAAS sur le processus de serveur de conteneur.

   Utilisez la même commande que dans l'étape précédente du didacticiel :

a. Accédez au dossier bin.

b. `java -classpath ../lib/objectgrid.jar;../applib/secsample.jar com.ibm.websphere.objectgrid.security.sample.guide.SecureSimpleApp ../security/client.properties manager manager1`

   L'application s'exécute correctement car l'utilisateur "manager" dispose de toutes les permissions sur les mappes de l'ObjectGrid accounting.

   Désormais, utilisez l'utilisateur "cashier" et non "manager" pour lancer l'application client.

c. Accédez au répertoire bin.

d. `java -classpath ../lib/objectgrid.jar;../applib/secsample.jar com.ibm.ws.objectgrid.security.sample.guide.SecureSimpleApp ../security/client.properties cashier cashier1`

Vous obtenez l'exception suivante :

```
Exception in thread "P=387313:O=0:CT" com.ibm.websphere.objectgrid.TransactionException:
rolling back transaction, see caused by exception
 at com.ibm.ws.objectgrid.SessionImpl.rollbackPMapChanges(SessionImpl.java:1422)
  at com.ibm.ws.objectgrid.SessionImpl.commit(SessionImpl.java:1149)
  at com.ibm.ws.objectgrid.SessionImpl.mapPostInvoke(SessionImpl.java:2260)
  at com.ibm.ws.objectgrid.ObjectMapImpl.update(ObjectMapImpl.java:1062)
  at com.ibm.ws.objectgrid.security.sample.guide.SimpleApp.run(SimpleApp.java:42)
 at com.ibm.ws.objectgrid.security.sample.guide.SecureSimpleApp.main(SecureSimpleApp.java:27)
Caused by: com.ibm.websphere.objectgrid.ClientServerTransactionCallbackException:
   Client Services - received exception from remote server:
     com.ibm.websphere.objectgrid.TransactionException: transaction rolled back,
   see caused by Throwable
        at com.ibm.ws.objectgrid.client.RemoteTransactionCallbackImpl.processReadWriteResponse(
            RemoteTransactionCallbackImpl.java:1399)
        at com.ibm.ws.objectgrid.client.RemoteTransactionCallbackImpl.processReadWriteRequestAndResponse(
            RemoteTransactionCallbackImpl.java:2333)
        at com.ibm.ws.objectgrid.client.RemoteTransactionCallbackImpl.commit(RemoteTransactionCallbackImpl.java:557)
        at com.ibm.ws.objectgrid.SessionImpl.commit(SessionImpl.java:1079)
        ... 4 more
Caused by: com.ibm.websphere.objectgrid.TransactionException: transaction rolled back, see caused by Throwable
        at com.ibm.ws.objectgrid.ServerCoreEventProcessor.processLogSequence(ServerCoreEventProcessor.java:1133)
        at com.ibm.ws.objectgrid.ServerCoreEventProcessor.processReadWriteTransactionRequest
     (ServerCoreEventProcessor.java:910)
        at com.ibm.ws.objectgrid.ServerCoreEventProcessor.processClientServerRequest(ServerCoreEventProcessor.java:1285)

        at com.ibm.ws.objectgrid.ShardImpl.processMessage(ShardImpl.java:515)
        at com.ibm.ws.objectgrid.partition.IDLShardPOA._invoke(IDLShardPOA.java:154)
        at com.ibm.CORBA.poa.POAServerDelegate.dispatchToServant(POAServerDelegate.java:396)
        at com.ibm.CORBA.poa.POAServerDelegate.internalDispatch(POAServerDelegate.java:331)
        at com.ibm.CORBA.poa.POAServerDelegate.dispatch(POAServerDelegate.java:253)
        at com.ibm.rmi.iiop.ORB.process(ORB.java:503)
        at com.ibm.CORBA.iiop.ORB.process(ORB.java:1553)
        at com.ibm.rmi.iiop.Connection.respondTo(Connection.java:2680)
        at com.ibm.rmi.iiop.Connection.doWork(Connection.java:2554)
        at com.ibm.rmi.iiop.WorkUnitImpl.doWork(WorkUnitImpl.java:62)
        at com.ibm.rmi.iiop.WorkerThread.run(ThreadPoolImpl.java:202)
        at java.lang.Thread.run(Thread.java:803)
Caused by: java.security.AccessControlException: Access denied (
   com.ibm.websphere.objectgrid.security.MapPermission accounting.customer write)
        at java.security.AccessControlContext.checkPermission(AccessControlContext.java:155)
        at com.ibm.ws.objectgrid.security.MapPermissionCheckAction.run(MapPermissionCheckAction.java:141)
        at java.security.AccessController.doPrivileged(AccessController.java:275)
        at javax.security.auth.Subject.doAsPrivileged(Subject.java:727)
        at com.ibm.ws.objectgrid.security.MapAuthorizer$1.run(MapAuthorizer.java:76)
        at java.security.AccessController.doPrivileged(AccessController.java:242)
        at com.ibm.ws.objectgrid.security.MapAuthorizer.check(MapAuthorizer.java:66)
        at com.ibm.ws.objectgrid.security.SecuredObjectMapImpl.checkMapAuthorization(SecuredObjectMapImpl.java:429)
        at com.ibm.ws.objectgrid.security.SecuredObjectMapImpl.update(SecuredObjectMapImpl.java:490)
        at com.ibm.ws.objectgrid.SessionImpl.processLogSequence(SessionImpl.java:1913)
        at com.ibm.ws.objectgrid.SessionImpl.processLogSequence(SessionImpl.java:1805)
        at com.ibm.ws.objectgrid.ServerCoreEventProcessor.processLogSequence(ServerCoreEventProcessor.java:1011)
        ... 14 more
```

Cette exception se produit car l'utilisateur "cashier" ne dispose pas de permission d'écriture et ne peut donc mettre à jour le client de mappe.

Votre système prend désormais en charge l'autorisation. Vous pouvez définir des politiques d'autorisation pour accorder différentes permissions à différents

utilisateurs. Pour de plus amples informations sur l'autorisation, voir les informations relatives à l'autorisation du client d'application dans le *Guide de programmation*.

Que faire ensuite

Effectuez l'étape suivante du didacticiel. Voir «Tutoriel sur la sécurité Java SE - Etape 4».

Tutoriel sur la sécurité Java SE - Etape 4

L'étape suivante vous explique comment activer une couche de sécurité pour la communication entre les nœuds finaux de votre environnement.

Avant de commencer

Assurez-vous d'avoir terminé l'«Didacticiel sur la sécurité Java SE - Etape 3», à la page 214 avant de commencer cette étape.

Pourquoi et quand exécuter cette tâche

La topologie eXtreme Scale prend en charge les protocoles TLS (Transport Layer Security) et SSL (Secure Sockets Layer) pour sécuriser la communication entre les nœuds finaux de l'ObjectGrid (client, serveurs de conteneur et serveurs de catalogue). Cette étape du tutoriel se base sur les étapes précédentes pour activer la sécurité du transport.

Procédure

1. **Création de clés et de fichiers de clés TLS/SSL**

   Afin d'activer la sécurité du transport, vous devez créer un fichier de clés et un fichier de clés certifiées. Cet exercice crée une seule clé et une seule paire fichier de clés-fichier de clés certifiées. Ces fichiers sont utilisés pour les clients, les serveurs de conteneur et les serveurs de catalogue ObjectGrid et sont créés par le biais de la commande keytool du Java Development Kit.

   - *Créer une clé privée dans le fichier de clés*

     ```
     keytool -genkey -alias ogsample -keystore key.jks -storetype JKS
     -keyalg rsa -dname "CN=ogsample, OU=Your Organizational Unit, O=Your
     Organization, L=Your City, S=Your State, C=Your Country" -storepass
     ogpass -keypass ogpass -validity 3650
     ```

     Cette commande permet de créer un fichier de clés key.jks contenant une clé "ogsample". Ce fichier de clés key.jks sera utilisé en tant que fichier de clés SSL.

   - *Exporter le certificat public*

     ```
     keytool -export -alias ogsample -keystore key.jks -file temp.key
     -storepass ogpass
     ```

     Cette commande permet d'extraire et de stocker le certificat public de la clé "ogsample" dans le fichier temp.key.

   - *Importer le certificat public du client dans le fichier de clés certifiées*

     ```
     keytool -import -noprompt -alias ogsamplepublic -keystore trust.jks
     -file temp.key -storepass ogpass
     ```

     Cette commande permet d'ajouter le certificat public au fichier de clés trust.jks. Ce fichier trust.jks est utilisé en tant que fichier de clés certifiées SSL.

2. **Configuration des fichiers de propriétés d'ObjectGrid**

   Au cours de cette étape, vous devez configurer le fichier de propriétés ObjectGrid pour activer la sécurité du transport.

   Tout d'abord, copiez les fichiers key.jks et trust.jks dans le répertoire objectgridRoot/security.

   Définissez les propriétés suivantes dans le fichier client.properties et server.properties.

   ```
   transportType=SSL-Required

   alias=ogsample
   contextProvider=IBMJSSE2
   protocol=SSL
   keyStoreType=JKS
   keyStore=../security/key.jks
   keyStorePassword=ogpass
   trustStoreType=JKS
   trustStore=../security/trust.jks
   trustStorePassword=ogpass
   ```

   transportType : la valeur de transportType est définie sur "SSL-Required", ce qui signifie que le transport requiert la couche Secure Sockets Layer. La configuration SSL doit être définie pour tous les points de contact ObjectGrid (clients, serveurs de catalogue et serveurs de conteneur). Tous les transports seront chiffrés.

   Les autres propriétés sont utilisées pour définir les configurations SSL. Pour plus d'informations, voir les informations sur le protocole TLS et la couche de connexion sécurisée dans le *Guide d'administration*. Vous devez suivre les instructions de cette rubrique pour mettre à jour le fichier orb.properties.

   Assurez-vous de suivre les instructions de cette page pour mettre à jour le fichier `orb.properties`.

   Dans le fichier `server.properties`, vous devez ajouter une propriété supplémentaire clientAuthentication et la définir sur false. Du côté serveur, vous n'avez pas besoin de certifier le client.

   ```
   clientAuthentication=false
   ```

3. **Exécution de l'application**

   Ces commandes sont identiques aux commandes de la section «Didacticiel sur la sécurité Java SE - Etape 3», à la page 214.

   Utilisez les commandes suivantes pour démarrer un serveur de catalogue.

   a. Placez-vous dans le répertoire bin : `cd objectgridRoot/bin`

   b. Démarrez le serveur de catalogue :

   - Linux UNIX

     ```
     startOgServer.sh catalogServer -clusterSecurityFile ../security/security.xml
     -serverProps ../security/server.properties -JMXServicePort 11001
     -jvmArgs -Djava.security.auth.login.config="../security/og_jaas.config"
     ```

   - Windows

     ```
     startOgServer.bat catalogServer -clusterSecurityFile ../security/security.xml
     -serverProps ../security/server.properties -JMXServicePort 11001 -jvmArgs
     -Djava.security.auth.login.config="../security/og_jaas.config"
     ```

     Les fichiers `security.xml` et `server.properties` ont été créés au cours de la procédure de la section «Tutoriel sur la sécurité Java SE - Etape 2», à la page 207.

     Utilisez l'option -JMXServicePort pour spécifier explicitement le port JMX pour le serveur. Cette option est requise pour utiliser la commande xsadmin.

     Démarrez le serveur de conteneur ObjectGrid sécurisé :

c. Placez-vous de nouveau dans le répertoire bin : `cd objectgridRoot/bin`

d.

- Linux UNIX

```
startOgServer.sh c0 -objectGridFile ../xml/SecureSimpleApp.xml
-deploymentPolicyFile ../xml/SimpleDP.xml -catalogServiceEndpoints
localhost:2809 -serverProps ../security/server.properties
-JMXServicePort 11002 -jvmArgs
-Djava.security.auth.login.config="../security/og_jaas.config"
-Djava.security.auth.policy="../security/og_auth.policy"
```

- Windows

```
startOgServer.bat c0 -objectGridFile ../xml/SecureSimpleApp.xml
-deploymentPolicyFile ../xml/SimpleDP.xml -catalogServiceEndpoints localhost:2809
-serverProps ../security/server.properties -JMXServicePort 11002
-jvmArgs -Djava.security.auth.login.config="../security/og_jaas.config"
-Djava.security.auth.policy="../security/og_auth.policy"
```

Notez les différences par rapport à la commande start du précédent serveur de conteneur :

- Utilisez SecureSimpleApp.xml à la place de SimpleApp.xml
- Ajoutez un autre argument -Djava.security.auth.policy pour définir le fichier de règles de l'autorisation JAAS sur le processus du serveur de conteneur.

Exécutez la commande suivante pour l'authentification de client :

a. `cd objectgridRoot/bin`

b.

```
javaHome/java -classpath ../lib/objectgrid.jar;../applib/secsample.jar
com.ibm.websphere.objectgrid.security.sample.guide.SecureSimpleApp
../security/client.properties manager manager1
```

Etant donné que l'utilisateur "manager" bénéficie de droits d'accès à toutes les mappes de l'ObjectGrid accounting, l'application s'exécute correctement.

Vous pouvez également utiliser xsadmin pour afficher la taille des mappes de la grille "accounting".

- Placez-vous dans le répertoire `objectgridRoot/bin`.
- Utilisez la commande `xsadmin` avec l'option -mapSizes comme suit.
  - UNIX Linux

    ```
    xsadmin.sh -g accounting -m mapSet1 -mapsizes -p 11001 -ssl
    -trustpath ..\security\trust.jks -trustpass ogpass -trusttype jks
    -username manager -password manager1
    ```

  - Windows

    ```
    xsadmin.bat -g accounting -m mapSet1 -mapsizes -p 11001 -ssl
    -trustpath ..\security\trust.jks -trustpass ogpass -trusttype jks
    -username manager -password manager1
    ```

  Notez que nous spécifions ici le port JMX du service de catalogue par le biais de -p 11001.

  La sortie suivante s'affiche.

```
Cet utilitaire administratif est fourni à titre d'exemple uniquement
et ne doit pas être considéré en tant que composant pris en charge par WebSphere eXtreme Scale.
Connexion au service de catalogue au localhost:1099
********** Résultats pour la grille - accounting, MapSet - mapSet1 **********
*** Liste des mappes pour c0 ***
Nom de la mappe : customer N° de partition #: 0 Taille de la mappe : 1 Type de fragment : principal
Nombre de serveurs : 1
Nombre de domaines : 1
```

Exécution de l'application avec un fichier de clés incorrect

Si votre fichier de clés certifiées ne contient pas le certificat public de la clé publique dans le fichier de clés, une exception s'affiche et vous indique que la clé n'est pas certifiée.

Pour afficher cela, créez un autre fichier de clés key2.jks.

```
keytool -genkey -alias ogsample -keystore key2.jks -storetype JKS
-keyalg rsa -dname "CN=ogsample, OU=Your Organizational Unit, O=Your
```

```
Organization, L=Your City, S=Your State, C=Your Country" -storepass
ogpass -keypass ogpass -validity 3650
```

Modifiez ensuite server.properties pour faire pointer le fichier de clés vers ce nouveau fichier de clés key2.jks:

```
keyStore=../security/key2.jks
```

Exécutez la commande suivante pour démarrer le serveur de catalogue :

a. Placez-vous dans le répertoire bin : `cd objectgridRoot/bin`

b. Démarrez le serveur de catalogue :

Linux UNIX

```
startOgServer.sh c0 -objectGridFile ../xml/SecureSimpleApp.xml
-deploymentPolicyFile ../xml/SimpleDP.xml -catalogServiceEndpoints localhost:2809
-serverProps ../security/server.properties -jvmArgs
-Djava.security.auth.login.config="../security/og_jaas.config"
-Djava.security.auth.policy="../security/og_auth.policy"
```

Windows

```
startOgServer.bat c0 -objectGridFile ../xml/SecureSimpleApp.xml
-deploymentPolicyFile ../xml/SimpleDP.xml -catalogServiceEndpoints localhost:2809
 -serverProps ../security/server.properties -jvmArgs
-Djava.security.auth.login.config="../security/og_jaas.config"
-Djava.security.auth.policy="../security/og_auth.policy"
```

L'exception suivante s'affiche :

```
Caused by: com.ibm.websphere.objectgrid.ObjectGridRPCException:
    com.ibm.websphere.objectgrid.ObjectGridRuntimeException:
        SSL connection fails and plain socket cannot be used.
```

Enfin, modifiez le fichier `server.properties` pour utiliser de nouveau le fichier `key.jks`.

Exemple et tutoriel sur les services de données REST

Nous allons expliquer comment démarrer rapidement les services de données REST d'WebSphere eXtreme Scale. Des instructions sont données pour WebSphere Application Server version 7.0, pour WebSphere Application Server Community Edition et pour Apache Tomcat.

Pourquoi et quand exécuter cette tâche

L'exemple fourni comprend le code source et les fichiers binaires compilés permettant d'exécuter une grille eXtreme Scale partitionnée. Cet exemple montre comment créer une grille simple et comment modéliser les données à l'aide d'entités eXtreme Scale. Il fournit deux applications client de ligne de commande qui permettent d'ajouter et d'interroger les entités à l'aide de Java ou de C# (voir la figure 1).

L'exemple de client Java utilise l'API Java EntityManager d'eXtreme Scale pour conserver et interroger les données dans la grille. Ce client peut être exécuté dans Eclipse ou à l'aide d'un script de ligne de commande. Notez que l'exemple de client Java n'illustre pas le service de données REST, mais permet de mettre à jour les données dans la grille de façon à ce qu'un navigateur Web ou un autre client puisse lire les données. Dans la figure 1, le client Java et le navigateur Web illustrent l'utilisation par des clients HTTP du service de données REST et montrent des clients Java d'eXtreme Scale utilisant la même grille eXtreme Scale et les mêmes données.

L'exemple de client Microsoft WCF Data Services C# communique avec la grille eXtreme Scale par le biais du service de données REST en utilisant .NET Framework. Le client WCF Data Services peut être utilisé à la fois pour mettre à

jour et interroger la grille.

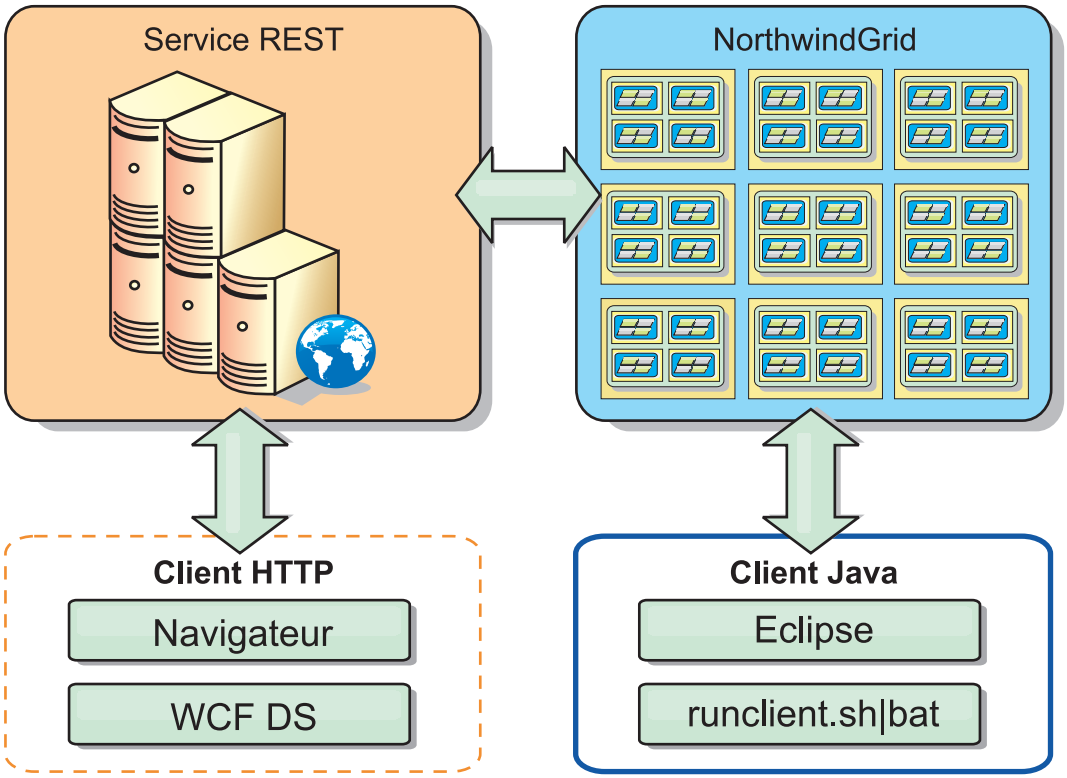


Figure 44. Exemple Mise en route de topologie

Procédure

1. Configurez et démarrez la grille eXtreme Scale. Voir «Activation du service de données REST», à la page 224.
2. Configurez et démarrez le service de données REST dans un navigateur Web. Voir «Configuration de serveurs d'applications pour le service de données REST», à la page 233.
3. Exécutez un client pour interagir avec le service de données REST. Deux options sont disponibles :
   a. Exécutez l'exemple de client Java pour remplir la grille avec les données en utilisant l'API EntityManager et interrogez les données de la grille par le biais d'un navigateur Web et du service de données REST d'eXtreme Scale. Voir «Utilisation d'un client Java avec les services de données REST», à la page 243.
   b. Exécutez l'exemple de client WCF Data Services C#. Voir «Client WCF de Visual Studio 2008 avec le service de données REST», à la page 244.

Conventions concernant les répertoires

A l'aide de nombreux exemples et en exposant la syntaxe de ligne de commande, nous allons expliquer comment faire référence à des répertoires spéciaux comme *racine_install_wxs*, et *rép_base_wxs*. Ces répertoires sont définis comme suit.

racine_install_wxs
Le répertoire *racine_install_wxs* est le répertoire racine où sont installés les

fichiers WebSphere eXtreme Scale. Il peut s'agir du répertoire dans lequel est extrait le fichier zip de la version d'essai ou de celui dans lequel est installé le produit eXtreme Scale.

- exemple où la version d'essai a été extraite :
  `/opt/IBM/WebSphere/eXtremeScale`
- example où eXtreme Scale est installé dans un répertoire autonome :
  `/opt/IBM/eXtremeScale`
- example où eXtreme Scale est intégré à WebSphere Application Server :
  `/opt/IBM/WebSphere/AppServer`

rép_base_wxs
Le répertoire *rép_base_wxs* est le répertoire racine des bibliothèques, des exemples et des composants du produit WebSphere eXtreme Scale. Ce répertoire est le même que *racine_install_wxs* lorsque la version d'essai est extraite. Pour les installations autonomes, il s'agit du sous-répertoire ObjectGrid du répertoire *racine_install_wxs*. Pour les installations intégrées à WebSphere Application Server, ce répertoire est le répertoire `optionalLibraries/ObjectGrid` du répertoire *racine_install_wxs*.

- exemple où la version d'essai a été extraite :
  `/opt/IBM/WebSphere/eXtremeScale`
- example où eXtreme Scale est installé dans un répertoire autonome :
  `/opt/IBM/eXtremeScale/ObjectGrid`
- example où eXtreme Scale est intégré à WebSphere Application Server :
  `/opt/IBM/WebSphere/AppServer/optionalLibraries/ObjectGrid`

racine_was
Le répertoire *racine_was* est le répertoire racine d'une installation WebSphere Application Server :

`/opt/IBM/WebSphere/AppServer`

rép_base_servicerest
Le répertoire *rép_base_servicerest* est le répertoire dans lequel se trouvent les bibliothèques et les exemples du service de données REST d'eXtreme Scale. Ce répertoire porte le nom de *restservice* et c'est un sous-répertoire du répertoire *rép_base_wxs*.

- exemple pour les déploiements autonomes :
  `/opt/IBM/WebSphere/eXtremeScale/ObjectGrid/restservice`
- exemple pour les déploiements intégrés à WebSphere Application Server :
  `/opt/IBM/WebSphere/AppServer/optionalLibraries/ObjectGrid/restservice`

racine_tomcat
Le répertoire *racine_tomcat* est le répertoire racine de l'installation d'Apache Tomcat.

`/opt/tomcat5.5`

racine_wasce
Le répertoire *racine_wasce* est le répertoire racine de l'installation de WebSphere Application Server Community Edition.

`/opt/IBM/WebSphere/AppServerCE`

rép_base_java
Le répertoire *rép_base_java* est le répertoire racine d'une installation de Java Runtime Environment Kit (JRE).

```
/opt/IBM/WebSphere/eXtremeScale/java
```

Activation du service de données REST

Le service de données REST peut représenter les métadonnées d'entités WebSphere eXtreme Scale pour représenter chaque entité sous la forme d'un EntitySet.

Démarrer un exemple de grille eXtreme Scale

En règle générale, démarrez la grille eXtreme Scale avant de lancer le service de données REST. La procédure qui suit va démarrer un processus de service de catalogue eXtreme Scale et deux processus de serveurs conteneurs.

WebSphere eXtreme Scale peut être installé selon trois méthodes différentes :

- installation d'essai
- déploiement autonome
- déploiement intégré à WebSphere Application Server

Evolutivité du modèle de données dans eXtreme Scale

L'exemple Microsoft Northwind utilise la table OrderDetail pour établir une association plusieurs-à-plusieurs entre les commandes et les produits.

Les spécifications ORM (Object to relational mapping) comme l'ADO.NET Entity Framework et JPA (Java Persistence API) peuvent mapper les tables et les relations à l'aide d'entités. Mais cette architecture n'est pas évolutive. Pour bien fonctionner, tout doit se trouver sur la même machine ou sur un cluster coûteux de machines.

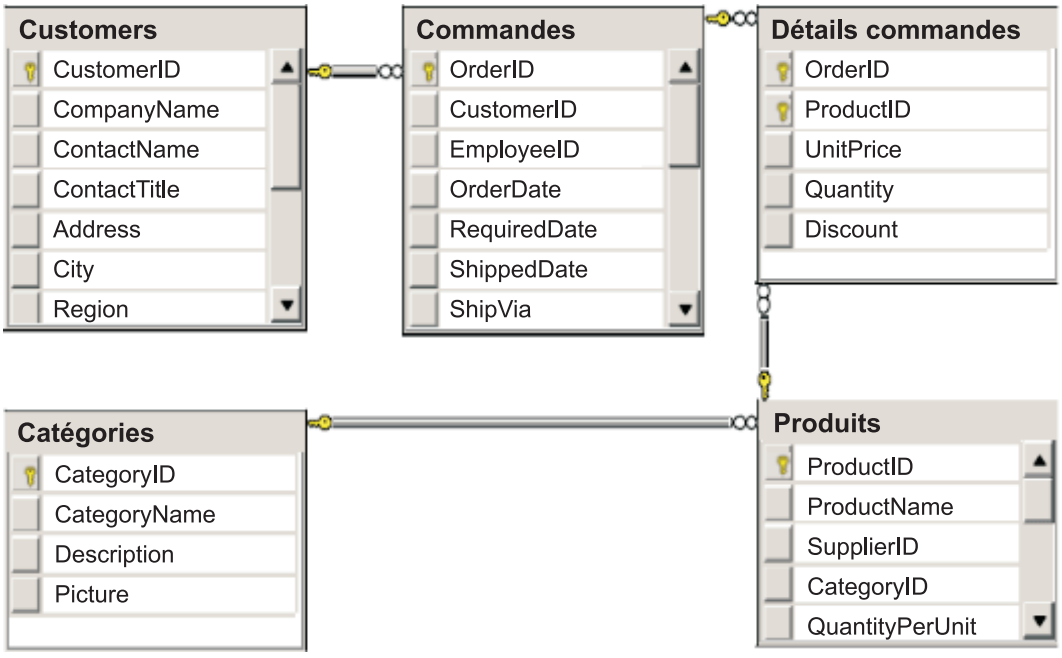


Figure 45. Schéma de l'exemple Microsoft SQL Server Northwind

Pour que puisse être créée une version évolutive de l'exemple, les entités doivent être modélisées de manière à ce que chaque entité ou chaque groupe d'entités en rapport puissent être partitionnées à partir d'une seule clé. De ce fait, les demandes peuvent être réparties entre plusieurs serveurs indépendants. Pour y arriver, les entités ont été divisées en deux arborescences : l'arborescence Customer et

l'arborescence Product. Dans ce modèle, chaque arborescence peut être partitionnée de manière indépendante et peut donc croître à des rythmes différents, d'où une plus grande évolutivité.

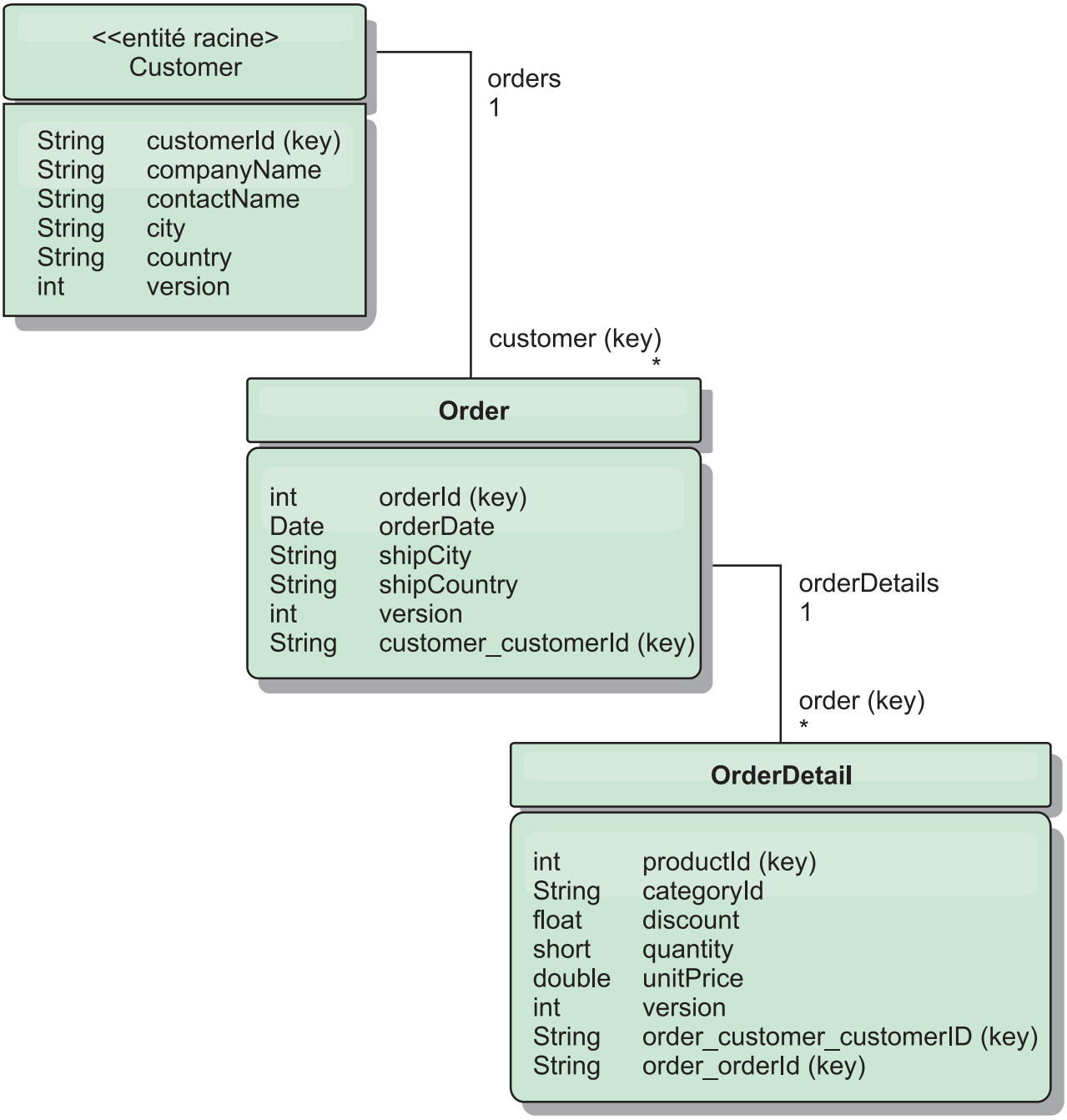


Figure 46. Schéma des entités Customer et Order

Par exemple, Order et Product ont tous les deux comme clés des entiers distincts et uniques. Et de fait, la table Order et la table Product sont deux tables réellement indépendantes l'une de l'autre. Considérons par exemple l'effet de la taille d'un catalogue (le nombre de produits vendus) avec le nombre total de commandes. A première vue, il peut sembler qu'avoir un grand nombre de produits implique d'avoir également un grand nombre de commandes, mais ce n'est pas nécessairement le cas. Si c'était vrai, il suffirait d'ajouter des produits au catalogue

pour augmenter les ventes. Les commandes et les produits ont leurs propres tables indépendantes. L'on peut étendre ce concept en imaginant que les commandes et les produits aient chacun leurs propres grilles de données. Des grilles indépendantes permettent de contrôler séparément le nombre des partitions et des serveurs, en plus de la taille de chaque grille, de manière à ce que votre application puisse évoluer. Si vous doublez la taille de votre catalogue, vous devez doubler la grille des produits, mais la grille des commandes ne bougera pas forcément. L'inverse est vrai pour un afflux de commandes.

Dans le schéma, un client a zéro ou plusieurs commandes et une commande a des articles (OrderDetail), chacun avec un produit spécifique. Un produit est identifié par un ID (la clé Product) dans chaque OrderDetail. La même grille stocke les clients, les commandes et les détails des commandes, Customer étant l'entité racine de la grille. L'on peut extraire les clients à partir de leur ID, mais l'on doit faire partir les commandes des ID clients. L'ID client est donc ajouté à la commande comme partie de sa clé. De la même manière, l'ID client et l'ID commande font partie de l'ID du détail de commande.

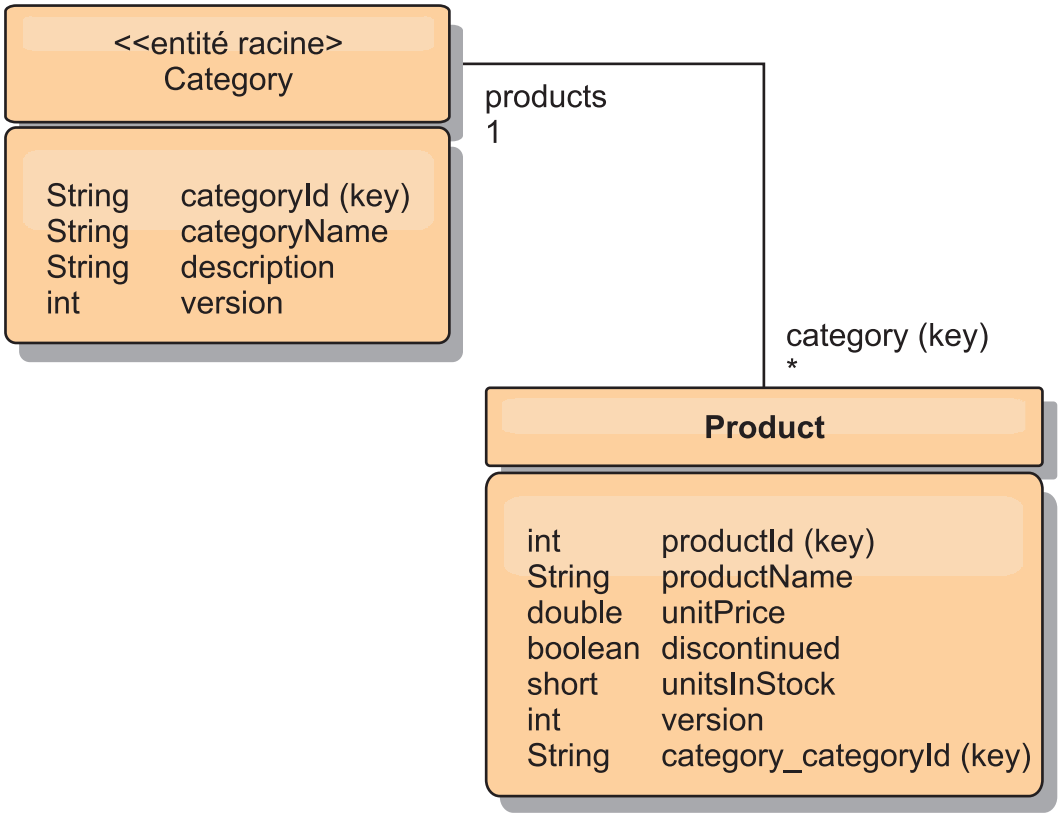


Figure 47. Schéma des entités Category et Product

Dans le schéma Category et Product, Category est la racine du schéma. Avec ce schéma, les clients peuvent rechercher des produits à partir de leur catégorie. Voir «Extraire et actualiser des données avec REST» pour d'autres détails sur les associations de clés et leur importance.

Extraire et actualiser des données avec REST

Le protocole OData requiert que toutes les entités soient adressables à partir de leur forme canonique. Cela signifie que chaque entité doit inclure la clé de l'entité racine partitionnée (la racine du schéma).

Voici un exemple de la manière d'utiliser l'association à partir d'une entité racine pour définir l'adresse d'un enfant dans :

```
/Customer('ACME')/order(100)
```

Dans WCF Data Services, l'entité enfant doit être directement adressable, c'est-à-dire que la clé dans la racine du schéma doit faire partie de la clé de l'enfant : /Order(customer_customerId='ACME', orderId=100). L'on y parvient en créant une association à l'entité racine dans laquelle l'association un-à-un ou plusieurs-à-un à l'entité racine est également identifiée comme une clé. Lorsque des entités sont incluses comme faisant partie de la clé, les attributs de l'entité parent sont exposés comme propriétés de la clé.

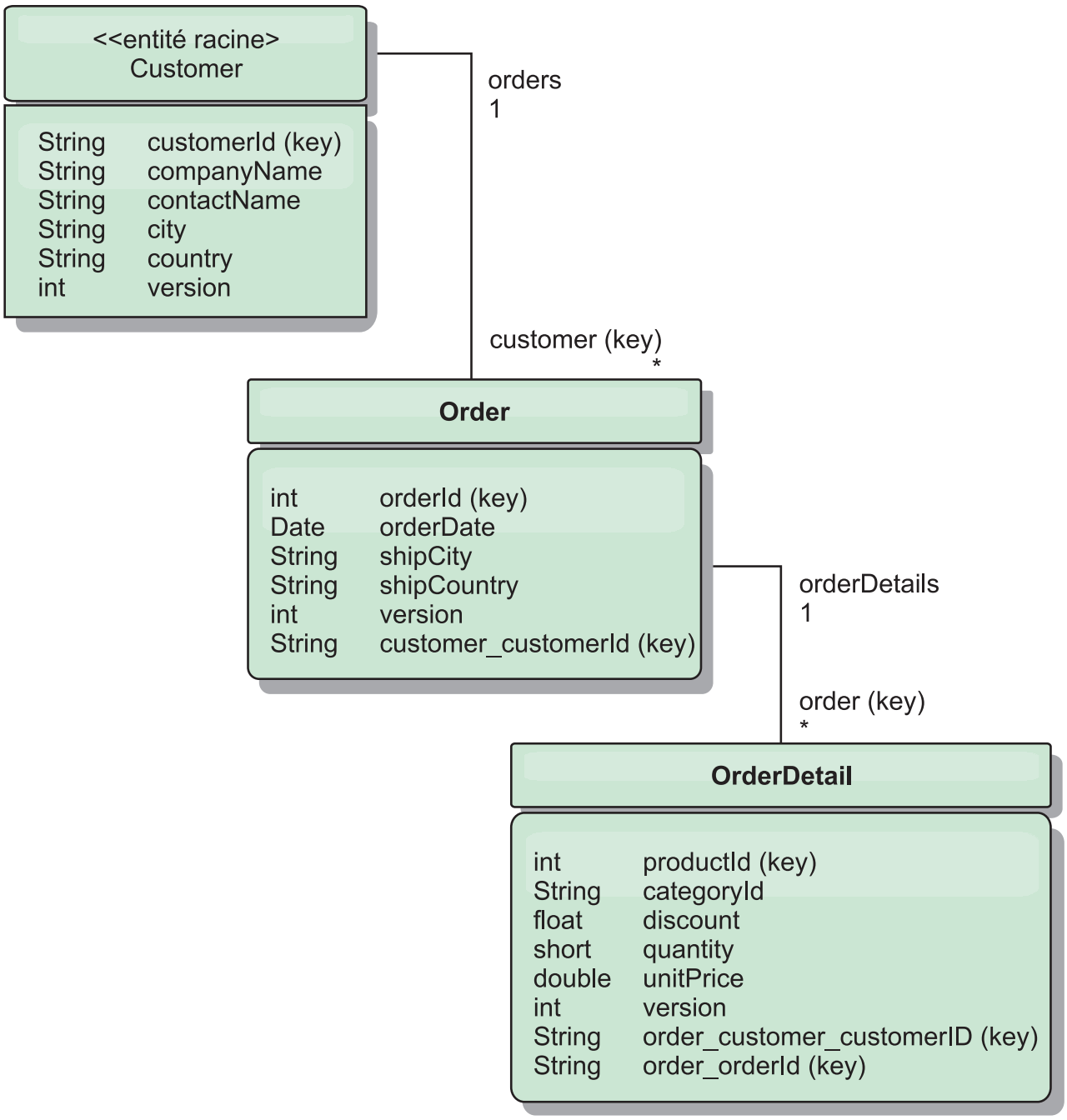


Figure 48. Schéma des entités Customer et Order

Le schéma des entités Customer/Order illustre la manière dont chaque entité est partitionnée à l'aide de Customer. L'entité Order inclut Customer comme partie de sa clé et elle est donc directement accessible. Le service de données REST expose toutes les associations de clés comme des propriétés individuelles : Order a customer_customerId et OrderDetail a order_customer_customerId and order_orderId.

L'API EntityManager permet de trouver la commande avec l'ID du client et celui de la commande :

```
transaction.begin();
// L'on recherche l'Order à l'aide du Customer.  Nous n'incluons que l'Id
// dans la classe Customer lorsqu'on génère l'instance de clé OrderId.
Order order = (Order) em.find(Order.class,
    new OrderId(100, new Customer('ACME')));
...
transaction.commit();
```

Lorsqu'on utilise le service de données REST, la commande peut être extraite avec l'une des URL :

- `/Order(orderId=100, customer_customerId='ACME')`
- `/Customer('ACME')/orders?$filter=orderId eq 100`

L'adresse de la clé Customer est créée avec l'attribut name de l'entité Customer, un caractère de soulignement et l'attribut name de l'ID Customer : customer_customerId.

Une entité peut également inclure une entité non racine comme faisant partie de sa clé si tous les ancêtres de cette entité non racine ont des associations de clés à la racine. Dans notre exemple, OrderDetail a une association de clés à Order et Order a une association de clés à l'entité Customer racine. Avec l'API EntityManager :

```
transaction.begin();
// L'on construit une instance de clé OrderDetailId.  Elle inclut
// Order et Customer avec uniquement le jeu de clés.
Customer customerACME = new Customer("ACME");
Order order100 = new Order(100, customerACME);
OrderDetailId orderDetailKey =
    new OrderDetailId(order100, "COMP");
OrderDetail orderDetail = (OrderDetail)
    em.find(OrderDetail.class, orderDetailKey);
...
```

Le service de données REST permet l'adressage direct d'OrderDetail :

```
/OrderDetail(productId=500, order_customer_customerId='ACME', order_orderId =100)
```

L'association partant de l'entité OrderDetail vers l'entité Product a été rompue pour permettre le partitionnement indépendant des commandes et du stock de produits. L'entité OrderDetail stocke la catégorie et l'ID de produit au lieu d'une relation en dur. En découpant les deux schémas d'entités, il n'est accédé qu'à une seule partition à la fois.

Le schéma Category et Product montre que l'entité racine est Category et que chaque Product a une association à une entité Category. L'entité Category est incluse dans l'identité Product. Le service de données REST expose une propriété de clé : category_categoryId qui permet l'adressage direct de Product.

Category étant l'entité racine, dans un environnement partitionné, Category doit être connu pour que Product puisse être trouvé. Avec l'API EntityManager, la transaction doit être fixée à l'entité Category avant toute recherche de Product.

Avec l'API EntityManager :

```
transaction.begin();
// L'on crée l'entité racine Category avec uniquement la clé. Cela
// nous permet de construire un ProductId sans avoir besoin de trouver
// d'abord la Category.  La transaction est à présent fixée
// à la partition où est stockée la Category "COMP".
```

```
Category cat = new Category("COMP");
Product product = (Product) em.find(Product.class,
    new ProductId(500, cat));
...
```

Le service de données REST permet l'adressage direct de Product :

```
/Product(productId=500, category_categoryId='COMP')
```

Démarrage d'une grille autonome pour les services de données REST

Appliquez la procédure exposée ci-après pour démarrer l'échantillon de grille de service REST de WebSphere eXtreme Scale pour un déploiement eXtreme Scale autonome.

Avant de commencer

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

- Installez la version autonome du produit WebSphere eXtreme Scale 7.1 et appliquez tous les correctifs ultérieurs.
- Téléchargez et extrayez la version d'évaluation de WebSphere eXtreme Scale version 7.1, qui inclut le service de données REST de WebSphere eXtreme Scale.

Pourquoi et quand exécuter cette tâche

Démarrez la grille d'exemple WebSphere eXtreme Scale.

Procédure

1. Démarrez le processus de service de catalogue. Ouvrez une ligne de commande ou une fenêtre de terminal, puis définissez la variable d'environnement JAVA_HOME :
   - Linux UNIX `export JAVA_HOME=base_java`
   - Windows `set JAVA_HOME=base_java`
2. `cd base_servicerest/gettingstarted`
3. Démarrez le processus de service de catalogue. Pour démarrer le service *sans* la sécurité d'eXtreme Scale, utilisez les commandes suivantes :
   - Linux UNIX `./runcat.sh`
   - Windows `runcat.bat`

   Pour démarrer le service *avec* la sécurité d'eXtreme Scale, utilisez les commandes suivantes :
   - Linux UNIX `./runcat_secure.sh`
   - Windows `runcat_secure.bat`
4. Démarrez deux processus de serveur conteneur. Ouvrez une autre ligne de commande ou fenêtre de terminal, puis définissez la variable d'environnement JAVA_HOME :
   - Linux UNIX `export JAVA_HOME=base_java`
   - Windows `set JAVA_HOME=base_java`
5. `cd base_servicerest/gettingstarted`
6. Démarrez un processus de serveur de conteneur :

Pour démarrer le serveur *sans* la sécurité d'eXtreme Scale, utilisez les commandes suivantes :

- Linux UNIX `./runcontainer.sh container0`
- Windows `runcontainer.bat container0`

Pour démarrer le serveur *avec* la sécurité d'eXtreme Scale, utilisez les commandes suivantes :

- Linux UNIX `./runcontainer_secure.sh container0`
- Windows `runcontainer_secure.bat container0`

7. Ouvrez une autre ligne de commande ou fenêtre de terminal, puis définissez la variable d'environnement JAVA_HOME :
   - Linux UNIX `export JAVA_HOME=base_java`
   - Windows `set JAVA_HOME=base_java`
8. `cd base_servicerest/gettingstarted`
9. Démarrez un second processus de serveur conteneur.

   Pour démarrer le serveur *sans* la sécurité d'eXtreme Scale, utilisez les commandes suivantes.

   - Linux UNIX `./runcontainer.sh container1`
   - Windows `runcontainer.bat container1`

   Pour démarrer le serveur *avec* la sécurité d'eXtreme Scale, utilisez les commandes suivantes.

   - Linux UNIX `./runcontainer_secure.sh container1`
   - Windows `runcontainer_secure.bat container1`

Résultats

Attendez que les conteneurs eXtreme Scale soient prêts avant de passer aux étapes suivantes. Les serveurs conteneurs sont prêts lorsque le message suivant s'affiche dans la fenêtre de terminal :

`CWOBJ1001I: Le serveur ObjectGrid nom_conteneur est prêt à traiter les requêtes.`

où *nom_conteneur* correspond au nom du conteneur qui a été démarré.

Démarrage sur WebSphere Application Server d'une grille pour les services de données REST

Appliquez la procédure exposée ici pour démarrer une grille de service REST de WebSphere eXtreme Scale autonome pour un déploiement de WebSphere eXtreme Scale intégré à WebSphere Application Server. Bien que WebSphere eXtreme Scale soit intégré à WebSphere Application Server, cette procédure démarre un processus de service de catalogue WebSphere eXtreme Scale autonome et un conteneur.

Avant de commencer

Installez le produit WebSphere eXtreme Scale version 7.1 dans un répertoire d'installation de WebSphere Application Server version 7.0.0.5 ou plus récente (la sécurité étant désactivée), étendez au moins un profil Application Server.

Pourquoi et quand exécuter cette tâche

Démarrez la grille WebSphere eXtreme Scale d'exemple.

Procédure

1. Démarrez le processus de service de catalogue. Ouvrez une fenêtre de ligne de commande ou de terminal et définissez la variable d'environnement JAVA_HOME :
   - Linux UNIX `export JAVA_HOME=base_java`
   - Windows `set JAVA_HOME=base_java`

   `cd base_servicerest/gettingstarted`
2. Démarrez le processus de service de catalogue.

   Pour démarrer le serveur *sans* la sécurité d'eXtreme Scale, utilisez les commandes suivantes.
   - Linux UNIX `./runcat.sh`
   - Windows `runcat.bat`

   Pour démarrer le serveur *avec* la sécurité d'eXtreme Scale, utilisez les commandes suivantes :
   - Linux UNIX `./runcat_secure.sh`
   - Windows `runcat_secure.bat`
3. Démarrez deux processus de serveur conteneur. Ouvrez une autre fenêtre de ligne de commande ou de terminal et définissez la variable d'environnement JAVA_HOME :
   - Linux UNIX `export JAVA_HOME=base_java`
   - Windows `set JAVA_HOME=base_java`
4. Démarrez un processus de serveur conteneur.

   Pour démarrer le serveur *sans* la sécurité d'eXtreme Scale, utilisez les commandes suivantes.
   a. Ouvrez une fenêtre de ligne de commande.
   b. `cd base_servicerest/gettingstarted`
   c. Pour démarrer le serveur *sans* la sécurité d'eXtreme Scale, utilisez les commandes suivantes :
      - Linux UNIX `./runcontainer.sh container0`
      - Windows `runcontainer.bat container0`
   d. Pour démarrer le serveur *avec* la sécurité d'eXtreme Scale, utilisez les commandes suivantes.
      - Linux UNIX `./runcontainer_secure.sh container0`
      - Windows `runcontainer_secure.bat container0`
5. Démarrez un second processus de serveur conteneur.
   a. Ouvrez une fenêtre de ligne de commande.
   b. `cd base_servicerest/gettingstarted`
   c. Pour démarrer le serveur *sans* la sécurité d'eXtreme Scale, utilisez les commandes suivantes :
      - Linux UNIX `./runcontainer.sh container1`

- Windows `runcontainer.bat container1`

d. Pour démarrer le serveur *avec* la sécurité d'eXtreme Scale, utilisez les commandes suivantes :

- Linux UNIX `./runcontainer_secure.sh container1`
- Windows `runcontainer_secure.bat container1`

Résultats

Attendez que les serveurs conteneurs soient prêts avant de passer aux étapes suivantes. Les serveurs conteneurs sont prêts lorsque le message suivant s'affiche :

`CWOBJ1001I: Le serveur ObjectGrid` *`nom_conteneur`* `est prêt à traiter les requêtes.`

où *nom_conteneur* correspond au nom du conteneur qui a été démarré à l'étape précédente.

Configuration de serveurs d'applications pour le service de données REST

Démarrage des services de données REST sur WebSphere Application Server version 7.0

Nous allons expliquer comment configurer et démarrer le service de données REST d'eXtreme Scale sur WebSphere Application Server version 7.0.

Avant de commencer

Vérifiez que l'exemple de grille eXtreme Scale est bien démarrée. Voir «Activation du service de données REST», à la page 224 pour savoir comment démarrer la grille.

Procédure

1. Téléchargez et installez WebSphere Application Server Version 7.0 for Developers.

   Restriction : N'activez pas la sécurité.
2. Téléchargez et installez WebSphere Application Server version 7.0 groupe de correctifs 5 ou plus récent.
3. Ajoutez au chemin d'accès aux classes du serveur d'applications le fichier JAR d'exécution du client WebSphere eXtreme Scale, le fichier `wsogclient.jar` et le fichier JAR ou le répertoire de configuration du service de données REST :
   a. Ouvrez la console d'administration de WebSphere Application Server.
   b. Accédez à **Environnement → Bibliothèques partagées.**
   c. Cliquez sur **Nouveau**.
   d. Ajoutez les entrées suivantes dans les zones appropriées :
      1) Nom : extremescale_client_v71
      2) Chemin d'accès aux classes : `base_wxs/lib/wsogclient.jar`
   e. Cliquez sur **OK**.
   f. Cliquez sur **Nouveau**.
   g. Ajoutez les entrées suivantes dans les zones appropriées :
      1) Nom : `extremescale_gettingstarted_config`

2) Chemin d'accès aux classes :
   - `base_servicerest/gettingstarted/restclient/bin`
   - `base_servicerest/gettingstarted/common/bin`

   A faire : Ajoutez chaque chemin sur une ligne séparée.

h. Cliquez sur **OK**.

i. Enregistrez les modifications apportées à la configuration principale.

4. Installez le fichier EAR du service de données REST (`wxsrestservice.ear`) sur WebSphere Application Server à partir de la console d'administration :
   a. Ouvrez la console d'administration de WebSphere Application Server.
   b. Placez-vous dans **Applications → Nouvelle application.**
   c. Recherchez `base_servicerest/lib/wxsrestservice.ear`, sélectionnez le fichier, puis cliquez sur **Suivant**.
   d. Sélectionnez les options d'installation, puis cliquez sur **Suivant**.
   e. Sur l'écran des avertissements de sécurité de l'application, cliquez sur **Continuer**.
   f. Sélectionnez les options d'installation par défaut, puis cliquez sur **Suivant**.
   g. Choisissez le serveur à mapper à l'application, puis cliquez sur **Suivant**.
   h. Sur la page de rechargement JSP, conservez les valeurs par défaut, puis cliquez sur **Suivant**.
   i. Dans la page des bibliothèques partagées, mappez le module `wxsrestservice.war` vers les bibliothèques partagées suivantes :
      - extremescale_client_v71
      - extremescale_ gettingstarted _config
   j. Sur la page de la relation de la bibliothèque partagée de la mappe, utilisez les valeurs par défaut, puis cliquez sur **Suivant**.
   k. Sur la page des hôtes virtuels de la mappe, utilisez les valeurs par défaut, puis cliquez sur **Suivant**.
   l. Dans la page des racines de contexte des mappes, spécifiez `wxsrestservice` comme racine de contexte.
   m. Sur l'écran Récapitulatif, cliquez sur **Terminer** pour terminer l'installation.
   n. Enregistrez les modifications apportées à la configuration principale.
5. Démarrez le serveur d'applications et l'application wxsrestservice du service de données REST d'eXtreme Scale. Une fois que l'application a démarré, recherchez dans le journal `SystemOut.log` du serveur d'applications le message suivant : `CWOBJ4000I: Le service de données REST de WebSphere eXtreme Scale a été démarré.`
6. Vérifiez que le service de données REST fonctionne correctement.
   a. Dans une fenêtre de navigateur, allez à `http://localhost:9080/wxsrestservice/restservice/NorthwindGrid`. Le document de service pour NorthwindGrid s'affiche.
   b. Allez à `http://localhost:9080/wxsrestservice/restservice/NorthwindGrid/$metadata`. Le document Entity Model Data Extensions (EDMX) s'affiche.
7. Pour arrêter les processus de grille, utilisez `CTRL+C` dans les fenêtres de commande respectives.

Démarrage des services de données REST avec WebSphere eXtreme Scale eXtreme Scale intégré dans WebSphere Application Server 7.0

Nous allons expliquer comment configurer et démarrer le service de données REST d'eXtreme Scale à l'aide de WebSphere Application Server version 7.0 qui a été intégré et étendu avec WebSphere eXtreme Scale.

Avant de commencer

Vérifiez que l'exemple de grille eXtreme Scale autonome est bien démarré. Voir «Activation du service de données REST», à la page 224 pour savoir comment démarrer la grille.

Pourquoi et quand exécuter cette tâche

Pour commencer avec le service de données REST de WebSphere eXtreme Scale REST à l'aide de WebSphere Application Server, procédez comme suit :

Procédure

1. Ajoutez au chemin d'accès aux classes le fichier JAR d'exemple de configuration du service de données REST de WebSphere eXtreme Scale :
   a. Ouvrez la console d'administration WebSphere
   b. Accédez à Environnement -> Bibliothèques partagées
   c. Cliquez sur Nouveau
   d. Ajoutez les entrées suivantes dans les champs appropriés :
      1) Nom : extremescale_gettingstarted_config
      2) Chemin d'accès aux classes
         - `base_servicerest/gettingstarted/restclient/bin`
         - `base_servicerest/gettingstarted/common/bin`

         A faire : Chaque chemin doit se trouver sur une ligne distincte.
   e. Cliquez sur **OK**
   f. Enregistrez les modifications apportées à la configuration principale.
2. Installez le fichier d'archive d'entreprise du service de données REST sur le serveur à l'aide de la console d'administration WebSphere :
   a. Ouvrez la console d'administration WebSphere
   b. Accédez à Applications -> Nouvelle application
   c. Allez au fichier `base_servicerest/lib/wxsrestservice.ear`. Sélectionnez le fichier, puis cliquez sur **Suivant**.
   d. Sélectionnez les options d'installation détaillées, puis cliquez sur **Suivant**.
   e. Dans l'écran d'avertissements de sécurité de l'application, cliquez sur **Continuer**.
   f. Sélectionnez les options d'installation par défaut, puis cliquez sur **Suivant**.
   g. Choisissez un serveur sur lequel mapper le module wxsrestservice.war, puis cliquez sur **Suivant**.
   h. Sur la page de rechargement JSP, utilisez les valeurs par défaut, puis cliquez sur **Suivant**.
   i. Sur la page des bibliothèques partagées, mappez le module wxsrestservice.war" sur les bibliothèques partagées suivantes, à savoir celles définies au cours de la première étape : `extremescale_ gettingstarted _config`

j. Sur la page de mappe des relations de bibliothèques partagées, utilisez les valeurs par défaut, puis cliquez sur **Suivant**.

k. Sur la page de mappe des hôtes virtuels,utilisez les valeurs par défaut, puis cliquez sur **Suivant**.

l. Sur la page de mappe de racine de contexte, définissez la racine de contexte sur : wxsrestservice.

m. Sur l'écran récapitulatif, cliquez sur **Fin** pour terminer la installation.

n. Enregistrez les modifications apportées à la configuration principale.

3. Si la grille eXtreme Scale a été démarrée avec la sécurité d'eXtreme Scale activée, définissez la propriété suivante dans le fichier `base_servicerest/gettingstarted/restclient/bin/wxsRestService.properties`.

```
ogClientPropertyFile=base_servicerest/gettingstarted/security/security.ogclient.properties
```

4. Démarrez le serveur d'applications et l'application wxsrestservice du service de données REST d'eXtreme Scale.

   Une fois que l'application a démarré, ouvrez le journal SystemOut.log du serveur d'applications et vérifiez que le message suivant est présent : `CWOBJ4000I: Le service de données REST de WebSphere eXtreme Scale a été démarré.`

5. Vérifiez que le service de données REST fonctionne :

   a. Ouvrez un navigateur et rendez-vous à l'adresse suivante :

      http://localhost:9080/wxsrestservice/restservice/NorthwindGrid

      Le document de service de la NorthwindGrid s'affiche.

   b. Rendez-vous à l'adresse suivante :

      http://localhost:9080/wxsrestservice/restservice/NorthwindGrid/$metadata

      Le document Entity Model Data Extensions (EDMX) s'affiche

6. Pour arrêter les processus de grille, utilisez CTRL+C dans les fenêtres de commande respectives des processus.

Démarrage du service de données REST sur WebSphere Application Server Community Edition.

Nous allons expliquer comment configurer et démarrer le service de données REST d'eXtreme Scale avec WebSphere Application Server Community Edition.

Avant de commencer

Vérifiez que la grille d'exemple est bien démarrée. Voir «Activation du service de données REST», à la page 224 pour savoir comment démarrer la grille.

Procédure

1. Téléchargez et installez WebSphere Application Server Community Edition version 2.1.1.3 ou plus récente dans le répertoire `racine_wasce`, par exemple le répertoire `/opt/IBM/wasce`.
2. Démarrez le serveur WebSphere Application Server Community Edition en exécutant la commande suivante :
   - Linux UNIX `racine_wasce/bin/startup.sh`
   - Windows `racine_wasce\bin\startup.bat`
3. Si la grille eXtreme Scale a été démarrée avec la sécurité d'eXtreme Scale activée, définissez les propriétés suivantes dans le fichier `base_servicerest/gettingstarted/restclient/bin/wxsRestService.properties`.

```
ogClientPropertyFile=base_servicerest/gettingstarted/security/security.ogclient.properties
loginType=none
```

4. Installez sur le serveur WebSphere Application Server Community Edition le service de données REST d'eXtreme Scale et l'exemple fourni :
   a. Ajoutez le fichier JAR d'exécution du client ObjectGrid au référentiel WebSphere Application Server Community Edition :
      1) Ouvrez et connectez-vous à la console d'administration de WebSphere Application Server Community Edition.

         Conseil : L'URL par défaut est `http://localhost:8080/console`. L'ID utilisateur par défaut est `system` et le mot de passe `manager`.
      2) Cliquez sur **Référentiel** dans le dossier Services.
      3) Dans la section **Ajouter une archive au référentiel**, entrez les éléments suivants dans les zones de texte :

Tableau 16. Archivage dans le référentiel

Zone de texte	Valeur
Fichier	base_wxs/lib/ogclient.jar
Groupe	com.ibm.websphere.xs
Artefact	ogclient
Version	7.0
Type	jar

      4) Cliquez sur le bouton Installer.

         Conseil : Reportez-vous à la note technique suivante pour des explications détaillées sur les différentes méthodes de configuration des dépendances de classes et des bibliothèques : Specifying external dependencies to applications running on WebSphere Application Server Community Edition.
   b. Déployez vers le serveur WebSphere Application Server Community Edition le module du service de données REST, le fichier `wxsrestservice.war`.
      1) Editez l'exemple de fichier XML de déploiement `base_servicerest/gettingstarted/wasce/geronimo-web.xml` pour inclure les dépendances de chemin d'accès dans les répertoires du chemin d'accès aux classes de l'exemple Mise en route :

         Modifiez le chemin classesDirs des deux GBeans du client Mise en route :
         - Le chemin de "classesDirs" pour le bean géré GettingStarted_Client_SharedLib doit avoir la valeur `base_servicerest/gettingstarted/restclient/bin`.
         - Le chemin de "classesDirs" pour le bean géré GettingStarted_Common_SharedLib doit avoir la valeur `base_servicerest/gettingstarted/restclient/bin`.
      2) Ouvrez et connectez-vous à la console d'administration de WebSphere Application Server Community Edition.

         Conseil : L'URL par défaut est `http://localhost:8080/console`. L'ID utilisateur par défaut est `system` et le mot de passe `manager`.
      3) Cliquez sur **Déployer nouveau**.

4) Entrez les valeurs suivantes dans les zones de texte de la page **Installer de nouvelles applications** :

Tableau 17. Valeurs d'installation

Zone de texte	Valeur
Archive	`base_servicerest/lib/wxsrestservice.war`
Plan	`base_servicerest/gettingstarted/wasce/geronimo-web.xml`

5) Cliquez sur le bouton Installer.

   La page de la console indique que l'application est installée et a démarré.

6) Recherchez le message ci-après sur la console ou dans le journal de sortie système de WebSphere Application Server Community Edition pour vérifier que le service de données REST a bien démarré :

   `CWOBJ4000I : Le service de données REST de WebSphere eXtreme Scale a été démarré.`

5. Vérifiez que le service de données REST fonctionne :
   a. Ouvrez le lien suivant dans une fenêtre de navigateur : `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid`. Le document de service de la grille NorthwindGrid s'affiche.
   b. Ouvrez le lien suivant dans une fenêtre de navigateur : `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid`. Le document Entity Model Data Extensions (EDMX) s'affiche.
6. Pour arrêter les processus de grille, utilisez `CTRL+C` dans les fenêtres de commande respectives.
7. Pour arrêter WebSphere Application Server Community Edition, utilisez la commande suivante :
   - UNIX Linux `racine_wasce/bin/shutdown.sh`
   - Windows `racine_wasce\bin\shutdown.bat`

   Conseil : L'ID utilisateur par défaut est `system` et le mot de passe `manager`. Si vous utilisez un port personnalisé, utilisez l'option `-port`.

Démarrage des services de données REST dans Apache Tomcat

Nous allons expliquer comment configurer et démarrer le service de données REST d'eXtreme Scale avec Apache Tomcat version 5.5 ou plus récente.

Avant de commencer

Vérifiez que l'exemple de grille eXtreme Scale est bien démarrée. Voir «Activation du service de données REST», à la page 224 pour savoir comment démarrer la grille.

Procédure

1. Téléchargez et installez dans racine_tomcat Apache Tomcat version 5.5 ou ultérieure. Par exemple : `/opt/tomcat`
2. Installez sur le serveur Tomcat le service de données REST d'eXtreme Scale et l'exemple fourni :
   a. Si vous utilisez un JRE ou un JDK Sun, vous devez installer l'ORB IBM sur Tomcat :

- Pour Tomcat version 5.5

  Copiez tous les fichiers JAR de :

  `base_wxs/lib/endorsed`

  vers

  `racine_tomcat/common/endorsed`
- Pour Tomcat version 6.0
  1) Créez un répertoire "validé".
     - UNIX Linux `mkdir racine_tomcat/endorsed`
     - Windows `md racine_tomcat/endorsed`
  2) Copiez tous les fichiers JAR de :

     `base_wxs/lib/endorsed`

     vers

     `racine_tomcat/endorsed`

b. Déployez le module de service de données REST : wxsrestservice.war vers le serveur Tomcat.

   Copiez le fichier wxsrestservice.war depuis :

   `base_servicerest/lib`

   vers :

   `racine_tomcat/webapps`

c. Ajoutez le fichier JAR d'exécution client ObjectGrid et le fichier JAR d'application au chemin d'accès aux classes dans Tomcat :
   1) Modifiez le fichier `racine_tomcat/conf/catalina.properties`.
   2) Ajoutez les noms de chemins suivants à la fin de la propriété shared.loader sous forme de liste séparée par des virgules :
      - base_wxs/lib/ogclient.jar
      - base_servicerest/gettingstarted/restclient/bin
      - base_servicerest/gettingstarted/common/bin

      Important : Le séparateur de chemin doit être une barre **oblique**.

3. Si la grille eXtreme Scale a été démarrée avec la sécurité d'eXtreme Scale activée, définissez les propriétés suivantes dans le fichier `base_servicerest/gettingstarted/restclient/bin/wxsRestService.properties`.

```
ogClientPropertyFile=base_servicerest/gettingstarted/security/security.ogclient.properties
loginType=none
```

4. Démarrez le serveur Tomcat avec le service de données REST :
   - Si vous utilisez Tomcat 5.5 sous UNIX® ou Windows®, ou Tomcat 6.0 sous UNIX :
     a. `cd racine_tomcat/bin`
     b. Démarrez le serveur :
        - UNIX Linux `./catalina.sh run`
        - Windows `catalina.bat run`
     c. La console affiche ensuite les journaux d'Apache Tomcat. Au démarrage du service de données REST, le message suivant s'affiche dans la console d'administration :

        `CWOBJ4000I : Le service de données REST de WebSphere eXtreme Scale a été démarré.`
   - Si vous utilisez Tomcat 6.0 sous Windows :

a. `cd racine_tomcat/bin`
b. Démarrez l'outil de configuration Apache Tomcat 6 avec la commande suivante : `tomcat6w.exe`
c. Dans la fenêtre des propriétés d'Apache Tomcat 6, cliquez sur le bouton Démarrer pour démarrer le serveur Tomcat.
d. Révisez les journaux suivants pour vérifier que le serveur Tomcat a démarré correctement :
   - `racine_tomcat/bin/catalina.log`
     Affiche l'état du moteur du serveur Tomcat.
   - `racine_tomcat/bin/stdout.log`
     Affiche le journal de sortie système.
e. Au démarrage du service de données REST, le message suivant s'affiche dans le journal de sortie système : `CWOBJ4000I : Le service de données REST de WebSphere eXtreme Scale a été démarré.`

5. Vérifiez que le service de données REST fonctionne :
   a. Ouvrez un navigateur et accédez à :
      http://localhost:8080/wxsrestservice/restservice/NorthwindGrid
      Le document de service pour NorthwindGrid s'affiche.
   b. Accédez à :
      http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/$metadata
      Le document Entity Model Data Extensions (EDMX) s'affiche.
6. Pour arrêter les processus de grille, utilisez CTRL+C dans les fenêtres de commande respectives.
7. Pour arrêter Tomcat, utilisez CTRL +C dans la fenêtre depuis laquelle vous l'avez démarré.

Utilisation d'un navigateur avec les services de données REST

Par défaut, le service de données REST d'eXtreme Scale crée des flux ATOM lorsqu'il utilise un navigateur Web. Le format des flux ATOM risque de ne pas être compatible avec les navigateurs plus anciens. Il y a également le risque d'une mauvaise interprétation des données qui ne seront pas affichées comme des données XML. Les rubriques qui suivent expliquent comment configurer Internet Explorer version 8 et Firefox version 3 pour afficher les flux ATOM et le XML dans le navigateur.

Pourquoi et quand exécuter cette tâche

Par défaut, le service de données REST d'eXtreme Scale crée des flux ATOM lorsqu'il utilise un navigateur Web. Le format des flux ATOM risque de ne pas être compatible avec les autres navigateurs ou d'être interprété de sorte que les données ne puissent pas être consultées comme des données XML. Pour les anciens navigateurs, vous serez invité à sauvegarder les fichiers sur le disque. Une fois que les fichiers ont été téléchargés, utilisez votre lecteur XML favori pour consulter les fichiers. Le XML généré n'étant pas formaté pour être affiché, tout est imprimé sur une seule ligne. La plupart des programmes de lecture XML, tels qu'Eclipse, prennent en charge le reformatage du XML dans un format lisible.

Pour les navigateurs modernes, tels que Microsoft Internet Explorer Version 8 et Firefox Version 3, les fichiers XML ATOM peuvent être affichés de manière native dans le navigateur. Les rubriques ci-après fournissent des détails sur la manière de

configurer Internet Explorer Version 8 et Firefox Version 3 pour afficher les flux ATOM et le XML dans le navigateur.

Procédure

Configuration d'Internet Explorer Version 8

- Pour permettre à Internet Explorer de lire les flux ATOM générés par le service de données REST, procédez comme suit :
  1. Cliquez sur **Outils → Options Internet**
  2. Sélectionnez l'onglet **Contenu**
  3. Cliquez sur le bouton **Paramètres** de la section **Flux et composants Web Slice**
  4. Désélectionnez la case "Activer le mode Lecture du flux"
  5. Cliquez sur **OK** pour retourner au navigateur.
  6. Redémarrez Internet Explorer.

Configuration de Firefox Version 3

- Firefox n'affiche pas automatiquement les pages avec le type de contenu suivant : application/atom+xml. La première fois qu'une page est affichée, Firefox vous invite à sauvegarder le fichier. Pour afficher la page, ouvrez le fichier avec Firefox, comme suit :
  1. Dans la boîte de dialogue de sélection de l'application, sélectionnez le bouton d'option "Ouvrir avec" et cliquez sur le bouton **Parcourir**.
  2. Accédez au répertoire d'installation de Firefox. Par exemple : `C:\Program Files\Mozilla Firefox`
  3. Sélectionnez `firefox.exe`, puis cliquez sur le bouton **OK**.
  4. Cochez la case "Toujours utiliser ce programme pour ouvrir ce type de fichier".
  5. Cliquez sur le bouton **OK**.
  6. Firefox affiche ensuite la page XML ATOM dans une nouvelle fenêtre ou page de navigateur
- Firefox affiche automatiquement les flux ATOM dans un format lisible. Toutefois, les flux créés par le service de données REST incluent XML. Firefox ne peut pas afficher le XML à moins que vous ne désactiviez le présentateur de flux. Contrairement à Internet Explorer, dans Firefox, le plug-in d'affichage des flux ATOM doit être édité de manière explicite. Pour configurer Firefox afin qu'il puisse lire les flux ATOM comme des fichiers XML, procédez comme suit :
  1. Ouvrez le fichier suivant dans un éditeur de texte : `<firefoxInstallRoot>\components\FeedConverter.js`. Dans le chemin d'accès, `<firefoxInstallRoot>` correspond au répertoire principal dans lequel Firefox est installé.

     Pour les systèmes d'exploitation Windows, le répertoire par défaut est le suivant : `C:\Program Files\Mozilla Firefox`.
  2. Recherchez le fragment de code similaire au suivant :

     ```
     // montre la page de flux si elle n'a pas été reniflée et que nous avons
     un document,
     // ou que nous avons un document, un titre et un lien ou un ID
     if (result.doc && (!this._sniffed ||
         (result.doc.title && (result.doc.link || result.doc.id)))) {
     ```
  3. Placez les deux lignes commençant par `if` et `result` en commentaire, en les précédant de deux barres obliques (//).
  4. Ajoutez l'instruction suivante au fragment de code : `if(0) {`.
  5. Le texte résultant doit ressembler au suivant :

```
// montre la page de flux si elle n'a pas été reniflée et que nous avons
un document,
// ou que nous avons un document, un titre et un lien ou un id
//if (result.doc && (!this._sniffed ||
//    (result.doc.title && (result.doc.link || result.doc.id)))) {
if(0) {
```

6. Enregistrez le fichier.
7. Redémarrez Firefox
8. Firefox peut maintenant afficher automatiquement tous les flux dans le navigateur.

- Testez votre configuration en essayant quelques URL.

Exemple

Nous allons décrire quelques exemples d'URL utilisables pour visualiser les données ajoutées par l'exemple d'initiation fourni avec le service de données REST d'eXtreme Scale. Avant d'utiliser ces URL, ajoutez le fichier par défaut à l'échantillon de grille eXtreme Scale en utilisant l'un des clients fournis à titre d'exemple : le client Java ou le client Visual Studio WCF Data Services.

Dans les exemples qui suivent, l'on part du principe que le port utilisé est le 8080, mais cela peut varier. Reportez-vous à la section pour des explications détaillées sur la manière de configurer le service de données REST sur différents serveurs d'applications.

- Visualiser un seul client dont l'ID est "ACME" :

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')
  ```

- Visualiser toutes les commandes du client "ACME" :

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')/orders
  ```

- Visualiser le client "ACME" et les commandes :

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')?$expand=orders
  ```

- Visualiser la commande 1000 du client "ACME" :

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=1000,customer_customerId='ACME')
  ```

- Visualiser la commande 1000 du client "ACME" et le Customer qui lui est associé :

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
  Order(orderId=1000,customer_customerId='ACME')?$expand=customer
  ```

- Visualiser la commande 1000 du client "ACME" et le Customer et les OrderDetails associés à ce client :

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
  Order(orderId=1000,customer_customerId='ACME')?$expand=customer,orderDetails
  ```

- Visualiser toutes les commandes du client "ACME" pour le mois d'octobre 2009 (GMT) :

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
  Customer(customerId='ACME')/orders?$filter=orderDate
  ge datetime'2009-10-01T00:00:00'
  and orderDate lt datetime'2009-11-01T00:00:00'
  ```

- Visualiser les trois premières commandes et les trois premiers orderDetails du client "ACME" pour le mois d'octobre 2009 (GMT) :

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
  Customer(customerId='ACME')/orders?$filter=orderDate
  ge datetime'2009-10-01T00:00:00'
  and orderDate lt datetime'2009-11-01T00:00:00'
  &$orderby=orderDate&$top=3&$expand=orderDetails
  ```

Utilisation d'un client Java avec les services de données REST

L'application client Java utilise l'API EntityManager d'eXtreme Scale pour insérer des données dans la grille.

Pourquoi et quand exécuter cette tâche

Dans les sections précédentes, nous avons vu comment créer une grille eXtreme Scale et configurer et démarrer le service de données REST d'eXtreme Scale. L'application client Java utilise l'API EntityManager d'eXtreme Scale pour insérer des données dans la grille. Elle ne montre pas comment utiliser les interfaces REST. Ce client a pour utilité de montrer comment l'API EntityManager sert à interagir avec la grille eXtreme Scale et autoriser les modifications de données dans la grille. Pour afficher des données dans la grille avec le service de données REST, utilisez un navigateur Web ou l'application client Visual Studio 2008.

Procédure

Pour ajouter rapidement du contenu à la grille eXtreme Scale, exécutez la commande suivante :

1. Ouvrez une ligne de commande ou une fenêtre de terminal, puis définissez la variable d'environnement JAVA_HOME :
   - Linux UNIX `export JAVA_HOME=base_java`
   - Windows `set JAVA_HOME=base_java`
2. `cd base_servicerest/gettingstarted`
3. Insérez des données dans la grille. Les données insérées seront extraites ultérieurement à l'aide d'un navigateur Web et du service de données REST.

   Si la grille a été démarrée *sans*la sécurité d'eXtreme Scale, utilisez les commandes suivantes :
   - UNIX Linux `./runclient.sh load default`
   - Windows `runclient.bat load default`

   Si la grille a été démarrée *avec*la sécurité d'eXtreme Scale, utilisez les commandes suivantes :
   - UNIX Linux `./runclient_secure.sh load default`
   - Windows `runclient_secure.bat load default`

   Pour un client Java, utilisez la syntaxe suivante :
   - UNIX Linux `runclient.sh commande`
   - Windows `runclient.bat commande`

   Les commandes suivantes sont disponibles :
   - `load default`

     Charge un jeu prédéfini d'entités Client, Catégorie et Produit dans la grille et crée un jeu aléatoire de Commandes pour chaque client.
   - `load category IDcatégorie Nomcatégorie IDpremierproduit nbre_produits`

     Crée une catégorie Produit et un nombre fixe d'entités Produit dans la grille. Le paramètre firstProductId identifie le numéro d'identification du premier produit et chaque produit suivant se voit affecter l'identifiant suivant jusqu'à ce que le nombre spécifié de produits soit créé.

- `load customer` *`companyCode contactNamecompanyName numOrders firstOrderIdshipCity maxItems discountPct`*

  Charge un nouveau client dans la grille et crée un jeu fixe d'entités Commande pour tout produit actuellement chargé dans la grille. Le nombre de commandes est déterminé par le paramètre <numOrders>. Chaque commande sera dotée d'un nombre aléatoire d'entités OrderDetail jusqu'à <maxItems>
- `display customer` *`companyCode`*

  Affiche une entité Customer et les entités Order et OrderDetail associées.
- `display category` *`categoryId`*

  Affiche une entité de produit Category et les entités Product associées.

Résultats

- `runclient.bat load default`
- `runclient.bat load customer IBM "John Doe" "IBM Corporation" 5 5000 Rochester 5 0.05`
- `runclient.bat load category 5 "Household Items" 100 5`
- `runclient.bat display customer IBM`
- `runclient.bat display category 5`

Exécution et génération avec Eclipse de la grille échantillon et du client Java

L'échantillon de démarrage de service de données REST peut être mis à jour et amélioré à l'aide d'Eclipse. Pour plus d'informations sur la configuration de votre environnement Eclipse, voir le document : `base_servicerest/gettingstarted/ECLIPSE_README.txt`.

Une fois que le projet WXSRestGettingStarted a été importé dans Eclipse et que sa génération s'effectue correctement, l'échantillon se recompile automatiquement et les fichiers script utilisés pour démarrer le serveur conteneur et le client sélectionnent automatiquement les fichiers de classes et les fichiers XML. Le service de données REST détecte automatiquement toute modification, car le serveur Web est configuré pour effectuer une lecture automatique des répertoires de construction Eclipse.

Important : En cas de modification des fichiers source ou de configuration, le serveur de conteneur eXtreme Scale et l'application du service de données REST doivent tous deux être redémarrés. Le serveur conteneur eXtreme Scale doit être démarré avant l'application Web du service de données REST.

Client WCF de Visual Studio 2008 avec le service de données REST

L'exemple Mise en route du service de données REST d'eXtreme Scale inclut un client WCF Data Services qui peut interagir avec le service de données REST. L'exemple est écrit comme une application de ligne de commande dans C#.

Configuration logicielle requise

L'exemple de client WCF Data Services C# requiert la configuration suivante :

- Système d'exploitation
  - Microsoft Windows XP

  - Microsoft Windows Server 2003
  - Microsoft Windows Server 2008
  - Microsoft Windows Vista
- Microsoft Visual Studio 2008 avec Service Pack 1

  Conseil : Pour les configurations matérielle et logicielle supplémentaires requises, voir le lien précédent.
- Microsoft .NET Framework 3.5 Service Pack 1
- Microsoft Support : une mise à jour de .NET Framework 3.5 Service Pack 1 est disponible

Génération et exécution du client Mise en route

L'exemple de client des services de données WCF inclut un projet et une solution Visual Studio 2008, ainsi que le code source permettant d'exécuter l'exemple. L'exemple doit être chargé dans Visual Studio 2008 et compilé dans un programme exécutable sous Windows pour pouvoir être exécuté. Pour générer et exécuter l'exemple, voir le document texte : `base_servicerest/gettingstarted/VS2008_README.txt`.

Syntaxe des commandes du client WCF Data Services C#

Windows `WXSRESTGettingStarted.exe <URL du service> <commande>`

L'<URL du service> est l'URL du service de données REST d'eXtreme Scale configuré dans la section .

Les commandes suivantes sont disponibles :

- `load default`

  Charge un ensemble prédéfini d'entités Customer, Category et Product dans la grille et crée un ensemble aléatoire de commandes pour chaque client.
- `load category <categoryId> <categoryName> <firstProductId> <numProducts>`

  Crée une catégorie de produits et un nombre fixe d'entités Product dans la grille. Le paramètre firstProductId identifie l'identificateur du premier produit et chaque produit suivant reçoit le prochain ID jusqu'à ce que le nombre de produits spécifié soit créé.
- `load customer <companyCode> <contactName> <companyName> <numOrders> <firstOrderId> <shipCity> <maxItems> <discountPct>`

  Charge un nouveau client dans la grille et crée un ensemble fixe d'entités Order pour tout produit aléatoire actuellement chargé dans la grille. Le nombre de commandes est déterminé par le paramètre <numOrders>. Chaque commande contient un nombre aléatoire d'entités OrderDetail, inférieur à la valeur <maxItems>
- `display customer <companyCode>`

  Affiche un entité Customer et les entités Order et OrderDetail associées.
- `display category <categoryId>`

  Affiche l'entité Category d'un produit et les entités Product associées.
- `unload`

  Supprime toutes les entités chargées à l'aide de la commande "default load".

Les exemples suivants illustrent diverses commandes.

- WXSRestGettingStarted.exe http://localhost:8080/wxsrestservice/restservice/NorthwindGrid load default
- WXSRestGettingStarted.exe http://localhost:8080/wxsrestservice/restservice/NorthwindGrid load customer
- IBM "John Doe" "IBM Corporation" 5 5000 Rochester 5 0.05
- WXSRestGettingStarted.exe http://localhost:8080/wxsrestservice/restservice/NorthwindGrid load category 5 "Household Items" 100 5
- WXSRestGettingStarted.exe http://localhost:8080/wxsrestservice/restservice/NorthwindGrid display customer IBM
- WXSRestGettingStarted.exe http://localhost:8080/wxsrestservice/restservice/NorthwindGrid display category 5

Remarques

Les références aux produits, logiciels et services d'IBM n'impliquent pas qu'ils soient distribués dans tous les pays dans lesquels IBM exerce son activité. Toute référence à un produit, logiciel ou service IBM n'implique pas que seul ce produit, logiciel ou service puisse être utilisé. Tout autre élément fonctionnellement équivalent peut être utilisé, s'il n'enfreint aucun droit d'IBM. L'évaluation et la vérification de son fonctionnement en conjonction avec d'autres produits, hormis ceux expressément désignés par IBM, relèvent de la responsabilité de l'utilisateur.

IBM peut détenir des brevets ou des demandes de brevet couvrant les produits mentionnés dans le présent document. La remise de ce document ne vous donne aucun droit de licence sur ces brevets ou demandes de brevet. Si vous désirez recevoir des informations concernant l'acquisition de licences, veuillez en faire la demande par écrit à l'adresse suivante :

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, New York 10594 USA

Les licenciés souhaitant obtenir des informations permettant : (i) l'échange des données entre des logiciels créés de façon indépendante et d'autres logiciels (dont celui-ci), et (ii) l'utilisation mutuelle des données ainsi échangées, doivent adresser leur demande à :

IBM Corporation
Mail Station P300
522 South Road
Poughkeepsie, NY 12601-5400
USA
Attention: Information Requests

Ces informations peuvent être soumises à des conditions particulières, prévoyant notamment le paiement d'une redevance.

Marques

Les termes suivant sont des marques d'IBM Corporation aux Etats-Unis et/ou dans certains autres pays :

- AIX
- CICS
- Cloudscape
- DB2
- Domino
- IBM
- Lotus
- RACF
- Redbooks
- Tivoli
- WebSphere
- z/OS

Java ainsi que tous les logos et toutes les marques incluant Java sont des marques de Sun Microsystems, Inc. aux Etats-Unis et/ou dans certains autres pays.

LINUX est une marque de Linus Torvalds aux Etats-Unis et/ou dans certains autres pays.

Microsoft, Windows, Windows NT® et le logo Windows sont des marques de Microsoft Corporation aux Etats-Unis et/ou dans certains autres pays.

UNIX est une marque enregistrée de l'Open Group aux Etats-Unis et dans d'autres pays.

Les autres noms de sociétés, de produits et de services peuvent appartenir à des tiers.

Index

T

U

V