WebSphere® eXtreme Scale versión 7.1

Visión general del producto

IBM

Esta edición se aplica a la versión 7, release 1, de WebSphere eXtreme Scale y a todos los releases y modificaciones posteriores hasta que se indique lo contrario en nuevas ediciones.

Contenido

Figuras

Tablas

Acerca de la *Visión general del producto*

El conjunto de documentación de WebSphere eXtreme Scale incluye tres volúmenes que proporcionan la información necesaria para utilizar, programar y administrar el producto WebSphere eXtreme Scale.

Biblioteca de WebSphere eXtreme Scale

La biblioteca de WebSphere eXtreme Scale contiene las siguientes publicaciones:

- La *Guía de administración* contiene la información necesaria para los administradores del sistema, incluido cómo planificar despliegues de aplicaciones, planificar la capacidad, instalar y configurar el producto, iniciar y detener servidores, supervisar el entorno y proteger el entorno.
- La *Guía de programación* contiene información dirigida a los desarrolladores de aplicaciones que indica cómo desarrollar aplicaciones para WebSphere eXtreme Scale utilizando la información de API incluida.
- El *Visión general del producto* contiene una vista de nivel superior de los conceptos de WebSphere eXtreme Scale, incluidos casos de ejemplo y guías de aprendizaje.

Para descargar las publicaciones, vaya a la página de la biblioteca WebSphere eXtreme Scale.

También puede acceder a la misma información de esta biblioteca en el centro de información de WebSphere eXtreme Scale.

Quién debe utilizar esta publicación

Esta publicación va dirigida a cualquier usuario que le interese obtener conocimientos sobre WebSphere eXtreme Scale.

Estructura de esta publicación

La publicación contiene información sobre los siguientes temas principales:

- **Capítulo 1** incluye una visión general de WebSphere eXtreme Scale
- **Capítulo 2** incluye información sobre conceptos de la memoria caché en el producto.
- **Capítulo 3** incluye información sobre la integración de la memoria caché.
- **Capítulo 4** incluye información sobre la escalabilidad.
- **Capítulo 5** incluye información sobre la disponibilidad.
- **Capítulo 6** incluye información sobre la seguridad.
- **Capítulo 7** incluye información sobre el proceso de transacciones.
- **Capítulo 8** incluye guías de aprendizaje para los conceptos básicos del producto.
- **Capítulo 9** incluye el glosario del producto.

Cómo obtener actualizaciones de esta publicación

Puede obtener actualizaciones para esta publicación descargando la versión más reciente desde la página de la biblioteca de WebSphere eXtreme Scale.

Envío de comentarios

Póngase en contacto con el equipo de documentación. ¿Ha encontrado lo que necesita? ¿Ha sido la información precisa y completa? Envíe sus comentarios sobre esta documentación mediante correo electrónico a wasdoc@us.ibm.com.

Capítulo 1. WebSphere eXtreme Scalevisión general

WebSphere eXtreme Scale es una cuadrícula de datos elástica, escalable y en memoria Coloca en la memoria caché, realiza particiones y réplicas y gestiona de forma dinámica los datos de aplicaciones y la lógica empresarial entre varios servidores. WebSphere eXtreme Scale realiza volúmenes masivos de proceso de transacciones con un alta eficacia y una escalabilidad lineal. Con WebSphere eXtreme Scale, también puede obtener calidades de servicio como, por ejemplo, integridad transaccional, alta disponibilidad y tiempos de respuesta predecibles.

La escalabilidad elástica es posible gracias al uso del almacenamiento en memoria caché de objetos distribuidos. Con la escalabilidad flexible, la cuadrícula de datos se supervisa y se gestiona sola. Puede realizar un aumento o una reducción de la topología añadiendo y eliminando servidores, que aumenta o disminuye la memoria, el rendimiento de red y la capacidad de proceso según sea necesario. Cuando se inicia el proceso de aumento, se añade la capacidad a la cuadrícula de datos mientras se ejecuta, sin necesidad de un reinicio. Por el contrario, el proceso de reducción elimina inmediatamente la capacidad. La cuadrícula de datos también se auto-arregla recuperándose automáticamente de anomalías.

WebSphere eXtreme Scale se puede utilizar de distintas formas. Se puede utilizar como una memoria caché muy potente o como una forma de un espacio de proceso de base de datos en memoria para gestionar el estado de la aplicación o como una plataforma para crear aplicaciones XTP (Extreme Transaction Processing) potentes.

Sin embargo, es importante tener en cuenta que no se puede considerar a eXtreme Scale como una base de datos de memoria real, en su mayor parte porque el último es demasiado sencillo para manejar algunas de las complejidades que puede gestionar eXtreme Scale. Tendría algunas de las mismas ventajas con ambos escenarios porque están en la memoria, pero si una base de datos de memoria tiene una máquina que falla, no es capaz de reparar el problema de forma inmediata. Dicho suceso sería particularmente desastroso si todo el entorno está en dicha máquina.

Para enfrentarse al problema de este tipo de anomalía, eXtreme Scale divide el conjunto de datos proporcionado en particiones, equivalentes a tres esquemas limitados. Los esquemas de árbol limitado describen la relación entre entidades. Cuando utiliza particiones, las relaciones de entidad deben modelar una estructura de datos de árbol, donde la cabeza del árbol es la entidad raíz y es la única entidad que está particionada. Todos los demás hijos de la entidad raíz se almacenan en la misma partición que la entidad raíz. Cada partición existe como una copia primaria o fragmento. Una partición contienen también fragmentos replica para hacer una copia de seguridad de los datos. Una base de datos en memoria no puede proporcionar este tipo de funcionalidad porque no está estructurada ni es dinámica de esta forma, lo que le obliga a realizar manualmente lo que eXtreme Scale realiza automáticamente. De forma adicional, puesto que una base de datos en memoria es una base de datos, puede permitir operaciones SQL y funciona como una mejora en términos de velocidad de proceso, en comparación con las bases de datos que no están en la memoria. WebSphere eXtreme Scale tiene su propio lenguaje de consulta en lugar del soporte SQL, pero es significativamente más elástico, lo que permite la partición de datos y proporciona una recuperación tras anomalía fiable.

Con la característica de memoria caché de grabación diferida, WebSphere eXtreme Scale puede actuar como una memoria caché frontal para una base de datos. Gracias al uso de esta memoria caché frontal, el rendimiento aumenta mientras se reduce la carga de la base de datos y los conflictos. WebSphere eXtreme Scale proporciona una escalada horizontal y una escalada vertical a un coste de proceso predecible.

La siguiente imagen muestra que en un entorno distribuido de memoria caché coherente, los clientes de eXtreme Scale envían y reciben datos de la cuadrícula de datos, que se pueden sincronizar automáticamente con un almacén de datos de fondo. La memoria caché es coherente porque todos los clientes ven los mismos datos en la memoria caché. Cada dato se almacena exactamente en un servidor grabable de la memoria caché, lo que impide tener copias inútiles de registros que podrían contener posiblemente distintas versiones de los datos. Una memoria caché coherente contiene más datos a medida que se añaden más servidores a la cuadrícula de datos, y se amplía de forma lineal, a medida que la cuadrícula de datos crece en tamaño. De forma opcional, también se pueden duplicar los datos para la tolerancia al error adicional.

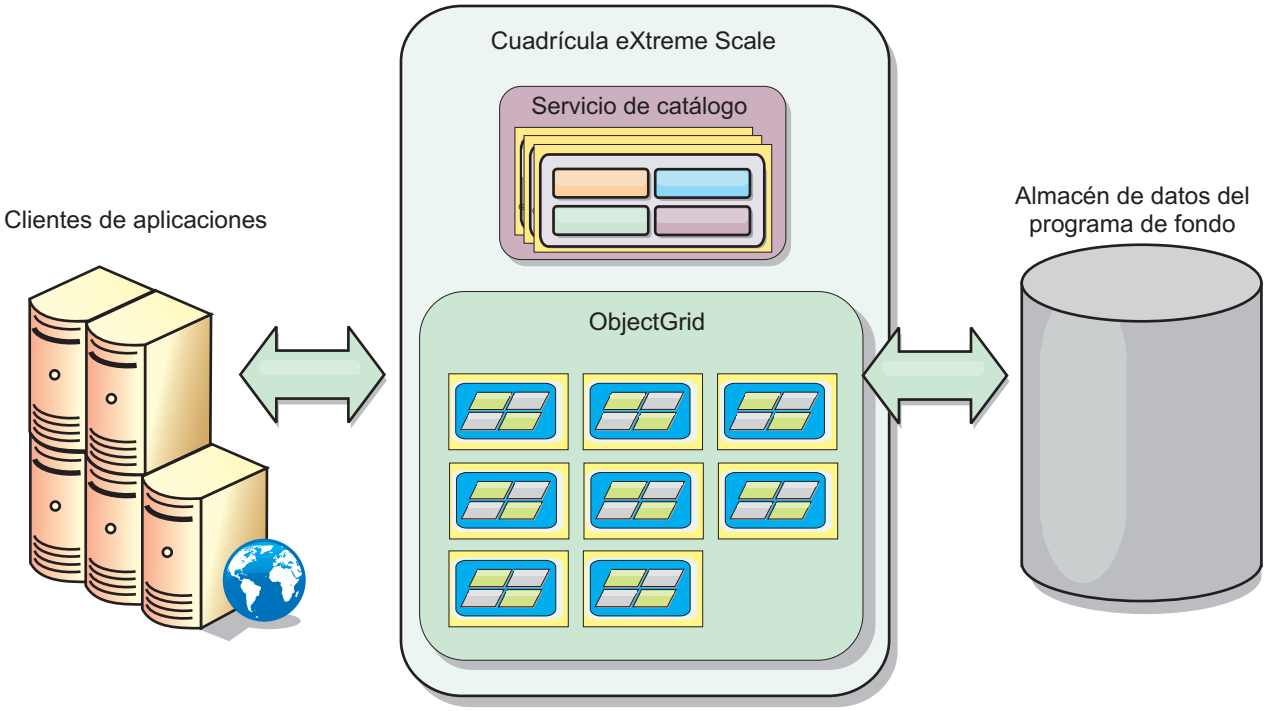


Figura 1. Topología de alto nivel

WebSphere eXtreme Scale tiene servidores que proporcionan su cuadrícula de datos en la memoria. Estos servidores se pueden ejecutar dentro de WebSphere Application Server, o en máquinas virtuales Java™ sencillas Java Standard Edition (J2SE), lo que permite tener más de uno estos por máquina física. Por lo tanto, la cuadrícula de datos en memoria debe ser bastante grande. La cuadrícula de datos no está limitada por, ni repercute sobre, la memoria ni el espacio de dirección de la aplicación o del servidor de aplicaciones. La memoria puede ser la suma de la memoria de cientos o miles máquinas virtuales Java que se ejecutan en muchas máquinas distintas.

Como un espacio de proceso de base de datos en memoria, WebSphere eXtreme Scale puede seguir respaldado por el disco, la base de datos, o ambos.

Mientras que eXtreme Scale proporciona varias API Java, muchos usos no requieren la programación de usuario, sino que sólo la configuración y el despliegue en la infraestructura WebSphere.

Paradigma básico

El paradigma de cuadrícula de datos básico es un par de clave-valor, donde la cuadrícula de datos almacena valores (objetos Java), con una clave asociada (otro objeto Java). La clave se utiliza posteriormente para recuperar el valor. En eXtreme Scale, una correlación consta de entradas de estos pares clave-valor.

WebSphere eXtreme Scale ofrece una serie de configuraciones de cuadrícula de datos, desde una única, una memoria caché local simple a una memoria caché distribuida grande, utilizando varias máquinas virtuales Java o servidores.

Además de almacenar objetos Java simples, puede almacenar objetos con relaciones. Puede utilizar un lenguaje de consulta que sea como SQL, con sentencias `SELECT ... FROM ... WHERE` para recuperar estos objetos. Por ejemplo, un objeto order (pedido) puede tener un objeto customer (cliente) y varios objetos item (artículo) asociados a éste. WebSphere eXtreme Scale soporta las relaciones de uno a uno, de uno a varios y de varios a varios.

WebSphere eXtreme Scale también soporta una interfaz de programación EntityManager, para almacenar entidades de la memoria caché. Esta interfaz de programación es muy similar a las entidades de Java Enterprise Edition. Las relaciones de entidad se pueden descubrir automáticamente desde un archivo XML de descriptor de entidad o anotaciones en las clases Java. De esta forma, se puede recuperar una entidad de la memoria caché mediante la clave primaria utilizando el método find de la interfaz EntityManager. Las entidades pueden persistir o eliminarse de la cuadrícula de datos dentro de un límite de transacción.

WebSphere eXtreme Scale proporciona prestaciones de XTP (Extreme Transaction Processing) que garantizan una infraestructura de aplicaciones inteligente para soportar la mayoría de las aplicaciones empresariales más importantes. Puede superar las limitaciones tradicionales del rendimiento de TI para generar niveles de ampliación, ventajas de proceso y la inteligencia empresarial necesaria para obtener resultados inteligentes y un beneficio empresarial competitivo sostenible.

Con el soporte para WebSphere Real Time, la oferta Java de tiempo real líder del sector, WebSphere eXtreme Scale permite a las aplicaciones XTP tener tiempos de respuesta más coherentes y predecibles. Consulte la información sobre Real Time Support en la *Guía de administración*.

Antes de desplegar eXtreme Scale en un entorno de producción, hay varias opciones que se deben tener en cuenta, incluidos el número de servidores para utilizar, la cantidad de almacenamiento en cada servidor y la duplicación síncrona o asíncrona.

Considere un ejemplo distribuido donde la clave es un simple nombre alfabético. La memoria caché se puede dividir en 4 particiones mediante claves: partición 1 para las claves que empiezan por A-E, partición 2 para las claves que empiezan por F-L, etc. Para la disponibilidad, una partición tiene un fragmento primario (se almacena en) y un fragmento de réplica. Los cambios de los datos de la memoria caché se realizan en el fragmento primario y se duplican en el fragmento secundario. Para una memoria caché distribuida, (o cuadrícula de datos, u ObjectGrid en el vocabulario de eXtreme Scale), configure el número de servidores

eXtreme que contendrá los datos de la cuadrícula de datos y eXtreme Scale distribuye los datos en los fragmentos sobre estas instancias de servidor. Para la disponibilidad, los fragmentos de réplica se colocan en máquinas separadas de fragmentos primarios.

WebSphere eXtreme Scale utiliza un servicio de catálogo para localizar el fragmento primario para cada clave. Maneja el movimiento de fragmentos entre los servidores eXtreme Scale, éstos o las máquinas físicas en las que se encuentran fallan y, posteriormente, se recuperan. Por ejemplo, si el servidor que contiene un fragmento de réplica falla, eXtreme Scale asigna un nuevo fragmento de réplica. Si un servidor que contiene un fragmento primario falla, el fragmento de réplica pasa a ser el fragmento primario y, al igual que antes, se construye un nuevo fragmento de réplica.

La interfaz de programación eXtreme Scale más sencilla es ObjectMap, que es una interfaz de correlación sencilla: map.put(key,value) para colocar un valor en la memoria caché y map.get(key) para recuperar, posteriormente, el valor.

Para obtener una descripción de los métodos recomendados que puede utilizar al diseñar las aplicaciones WebSphere eXtreme Scale, lea el artículo siguiente en developerWorks: Principles and best practices for building high performing and highly resilient WebSphere eXtreme Scale applications (Principios y métodos recomendados para crear aplicaciones de WebSphere eXtreme Scale muy flexibles y de alto rendimiento).

Características nuevas y en desuso de este release

WebSphere eXtreme Scale incluye muchas características nuevas en la versión 7.1, incluida la integración con la memoria caché dinámica, las correlaciones de matrices de bytes y más características.

Novedades en WebSphere eXtreme Scale versión 7.1

Tabla 1. Nuevas características en WebSphere eXtreme Scale versión 7.1

Característica	Descripción
Integración de la información del cliente DB2	Integre los plug-ins del cargador JPA de eXtreme Scale con DB2 de modo que, cuando DB2 se utilice como base de datos de programa de fondo, la información de WebSphere eXtreme Scale (nombre de usuario, nombre de estación de trabajo, nombre de aplicación e información de la contabilidad) se pueda hacer disponible en DB2 Performance Monitor. Esta característica permite habilitar e inhabilitar la configuración de la información del cliente para DB2. Esta función está inhabilitada de forma predeterminada.Para obtener más información, consulte “Cargadores” en la página 43.
Configuración del dominio de servicio de catálogo	Los dominio de servicio de catálogo se pueden configurar con la consola administrativa de WebSphere Application Server o con tareas administrativas. Los dominios de servicio de catálogo definen un grupo. Si desea más información, consulte la información sobre cómo crear los dominios del servicio de catálogo en *Guía de administración*.
Réplica con varios maestros	Se pueden enlazar varios centros de datos juntos de forma asíncrona, con lo que se permite el acceso local del centro de datos a los datos y se permite también mantener la alta disponibilidad. Si desea más información, consulte “Topologías de réplica de cuadrícula con varios maestros (AP)” en la página 24.
Desalojador TTL de última actualización	El desalojador TTL se ha actualizado para realizar un seguimiento de la hora a la que se ha actualizado una entrada, que se extiende al desalojador TimeToLive. Si desea más información, consulte la información sobre el desalojador TimeToLive (TTL) en la *Guía de programación*
Estadísticas usedBytes de cuadrículas en memoria	Se puede realizar un seguimiento de la cantidad de memoria utilizada por las entradas de memoria caché en un BackingMap utilizando todos los proveedores de estadísticas. Si desea más información, consulte la información sobre el dimensionamiento del consumo de memoria caché en la *Guía de administración*.
Estadísticas dinámicas	Ahora se pueden habilitar e inhabilitar las estadísticas según demanda. Si desea más información, consulte la información sobre cómo supervisar con MBeans en la *Guía de administración*.
Consola de supervisión	La consola de supervisión gráfica proporciona vistas históricas y actuales en estadísticas del servidor WebSphere eXtreme Scale. Si desea más información, consulte la información sobre la consola web en la *Guía de administración*.

Tabla 1. Nuevas características en WebSphere eXtreme Scale versión 7.1 (continuación)

Característica	Descripción
Gestor de sesiones HTTP mejorado	La configuración del gestor de sesiones HTTP se ha simplificado. Puede configurar ahora el gestor de sesiones HTTP en la consola administrativa de WebSphere Application Server. Si desea más información, consulte la información sobre cómo configurar el gestor de sesiones HTTP en la *Guía de administración*.
Soporte de cliente con varias conexiones a red (multi-homed)	Los clientes se pueden configurar para utilizar un adaptador de red concreto. Si desea más información, consulte la información sobre el archivo de propiedades de cliente en la *Guía de administración*.
ISA Lite	IBM® Support Assistant Lite para WebSphere eXtreme Scale proporciona una recopilación automática de los datos y soporte de análisis de síntomas para los casos de determinación de problemas. Si desea más información, consulte la información sobre IBM support assistant para WebSphere eXtreme Scale en la *Guía de administración*.
REST	El servicio de datos REST proporciona acceso a los clientes no Java a los datos de eXtreme Scale, que admite el protocolo OData (Open Data Protocol), con lo que se proporciona una compatibilidad completa con Microsoft® WCF Data Services. Para obtener más información, consulte Capítulo 8, "Visión general de los servicios de datos REST", en la página 181.
Instalación solo del cliente	Los clientes de WebSphere eXtreme Scale se pueden instalar ahora de forma individual, con lo que disminuye el espacio de instalación utilizado por las aplicaciones WebSphere eXtreme Scale. Si desea más información, consulte la información sobre cómo instalar y desplegar WebSphere eXtreme Scale en la *Guía de administración*.

Características en desuso

Tabla 2. Características en desuso

En desuso	Acción de migración recomendada
	Cree un dominio de servicio de catálogo en la consola administrativa de WebSphere Application Server, que crea la misma configuración que si se utiliza la propiedad personalizada. Si desea más información, consulte la información sobre cómo crear los dominios del servicio de catálogo en *Guía de administración*.
	Utilice CatalogServiceManagementMBean en su lugar.
Recurso de partición (WPF): el recurso de partición es un conjunto de API de programación que permiten a las aplicaciones Java EE dar soporte a la agrupación en clúster asimétrico.	Las prestaciones de WPF se pueden realizar de forma alternativa en WebSphere eXtreme Scale.
StreamQuery: una consulta continua sobre los datos en curso almacenados en correlaciones ObjectGrid.	Ninguna
Configuración de cuadrícula estática: una topología estática basada en clúster que utiliza el archivo XML de despliegue de clúster.	Se sustituye por la topología mejorada de despliegue dinámico para la gestión de grandes cuadrículas de datos.
Propiedades del sistema en desuso: las propiedades del sistema para especificar los archivos de propiedades de servidor y de clientes están en desuso.	Puede seguir utilizando estos argumentos, pero cambie las propiedades del sistema por los valores nuevos. Consulte la información sobre los archivos de propiedades en *Guía de administración* si desea más información.

Trabajar con WebSphere eXtreme Scale

WebSphere eXtreme Scale es una cuadrícula de datos elástica, escalable y en memoria Coloca en la memoria caché, realiza particiones y réplicas y gestiona de forma dinámica los datos de aplicaciones y la lógica empresarial entre varios servidores.

Puesto que no es una base de datos en memoria, debe considerar los requisitos de configuración específicos para eXtreme Scale. El primer paso para desplegar una cuadrícula de datos de eXtreme Scale es iniciar un grupo principal y un servicio de catálogo, que actuará como coordinador para todas las demás máquinas virtuales Java que participan en la cuadrícula y gestionar la información de configuración. Los procesos de WebSphere eXtreme Scale se inician con unas sencillas invocaciones de script de mandato desde la línea de mandatos.

El siguiente paso es iniciar los procesos del servidor WebSphere eXtreme Scale para la cuadrícula para almacenar y recuperar datos. Cuando se inician los servidores, se registran automáticamente ellos mismos con el grupo principal y el servicio de catálogo que les permite cooperar en el suministro de servicios de cuadrícula. Más servidores aumentan tanto la capacidad como la fiabilidad de la cuadrícula.

Una cuadrícula local es una cuadrícula sencilla y de instancia única donde están todos los datos. Para utilizar de forma eficaz eXtreme Scale como un espacio de proceso de base de datos en memoria, puede configurar y desplegar una cuadrícula distribuida. Los datos de la cuadrícula distribuida se extienden sobre los distintos servidores eXtreme Scale que los contienen, de forma que cada servidor sólo contiene algunos de los datos, llamados partición.

El parámetro de configuración de cuadrícula distribuida de clave es el número de particiones de la cuadrícula. Los datos de cuadrícula se particionan en este número de subconjuntos, cada uno de los cuales recibe el nombre de partición. El servicio de catálogo localiza la partición para un dato determinado basado en su clave. El número de particiones afecta directamente a la capacidad y escalabilidad de la cuadrícula. Un servidor puede contener una o más particiones de cuadrícula. Por lo tanto, el espacio de la memoria del servidor limita el tamaño de una partición. Por el contrario, aumentar el número de particiones, aumenta la capacidad de la cuadrícula. La capacidad máxima de una cuadrícula es el número de particiones que programa el tamaño de la memoria utilizable de un servidor, que puede ser una JVM.

Los datos de una partición se almacenan en un fragmento. Para la disponibilidad, se puede configurar una cuadrícula con réplicas, que pueden ser síncronas o asíncronas. Los cambios en los datos de cuadrícula se realizan en el fragmento primario y se duplican en los fragmentos duplicados. La memoria total consumida/necesaria por una cuadrícula es el tamaño de las veces de cuadrícula (1 (para la primaria) + el número de réplicas).

WebSphere eXtreme Scale distribuye fragmentos de una cuadrícula sobre el número de servidores que contienen la cuadrícula. Estos servidores podrían estar en las mismas y/o en máquinas físicas separadas. Para la disponibilidad, los fragmentos de réplica se colocan en máquinas separadas de fragmentos primarios.

WebSphere eXtreme Scale supervisa el estado de sus servidores y mueve los fragmentos entre ellos, si éstos y/o sus máquinas físicas fallan y se recuperan, posteriormente. Por ejemplo, si el servidor que contiene un fragmento de réplica falla, eXtreme Scale asignará un nuevo fragmento de réplica y duplicará los datos

del fragmento primario a la réplica nueva. Si un servidor que contiene un fragmento primario falla, el fragmento de réplica pasa a ser el fragmento primario y, al igual que antes, se construye un nuevo fragmento de réplica. Si inicia un servidor adicional para la cuadrícula, los fragmentos se distribuirán entre todos los servidores, de forma que la carga en cada uno de ellos se equilibra cuando es posible. Esto recibe el nombre de escala hacia fuera. De forma similar, para la escalada hacia dentro, puede detener uno de los servidores para reducir los recursos consumidos por una cuadrícula, y los fragmentos se volverán a equilibrar entre los servidores restantes, simplemente al igual que en una situación de anomalía.

Visión general del despliegue de aplicaciones

Antes de utilizar WebSphere eXtreme Scale en un entorno de producción, tenga en cuenta las siguientes cuestiones para optimizar el despliegue.

Planificación del despliegue de la aplicación

A continuación se enumeran los elementos que deben tenerse en cuenta:

- Número de sistemas y procesadores: ¿cuántas máquinas físicas y procesadores se necesitan en el entorno?
- Número de servidores: cuántos servidores eXtreme Scale para alojar correlaciones de eXtreme Scale?
- Número de particiones: el volumen de datos almacenado en las correlaciones es un factor fundamental para determinar el número de particiones que se necesita.
- Número de réplicas: ¿cuántas replicas se necesitan en cada fragmento primario en el dominio?
- Réplica síncrona o asíncrona: ¿son vitales los datos de modo que la réplica síncrona es necesaria? ¿Es el rendimiento, en cambio, una prioridad mayor, por lo que la opción es la réplica asíncrona?
- Tamaño de almacenamiento dinámico: ¿cuántos datos se almacenarán en cada servidor?

Integración con otros productos WebSphere Application Server

Puede integrar WebSphere eXtreme Scale con otros productos de servidor como, por ejemplo, WebSphere Application Server y WebSphere Application Server Community Edition.

Configuración del gestor de sesiones HTTP para trabajar con WebSphere Application Server Community Edition

WebSphere Application Server Community Edition puede compartir el estado de sesión, pero no de una forma eficaz y escalable. WebSphere eXtreme Scale proporciona un alto rendimiento, una capa de persistencia distribuida que puede utilizarse para replicar el estado, pero que no se integra fácilmente con otro servidor de aplicaciones fuera de WebSphere Application Server. Puede integrar estos dos productos para proporcionar una solución de gestión de sesiones escalable. Consulte *Guía de administración* si desea más detalles.

Configuración del gestor de sesiones de WebSphere eXtreme Scale para que funcione con WebSphere Application Server

El primer gestor de sesiones HTTP se entregaba con WebSphere Extended Deployment DataGrid versión 6.1.0.0. Las versiones posteriores hasta la versión 6.1.0.5 no han cambiado los métodos de uso, puesto que cumple con la especificación J2EE (Java 2 Enterprise Edition) para la integración y la recuperación de sesiones, pero se han ido produciendo mejoras de rendimiento y de calidad de servicio con cada release. Para asegurarse de que obtiene la mejora calidad de servicio, la recomendación es aplicar el fixpaxk de WebSphere eXtreme Scale versión 6.1.0.5.

Consulte *Guía de administración* si desea más detalles.

Cambios en el nombre del producto

Tenga en cuenta que WebSphere eXtreme Scale era conocido anteriormente por otros nombres.

Cambios en el nombre del producto

Al hacer referencia a otra documentación, materiales de marketing o presentaciones, recuerde que eXtreme Scale también se conocía anteriormente con los siguientes nombres.

- ObjectGrid
- WebSphere Extended Deployment Data Grid

Aunque el propio producto ahora es conocido como WebSphere eXtreme Scale, el término ObjectGrid aparece en la documentación y en otros sitios porque es el nombre del artefacto que habilita la tecnología de la cuadrícula.

Versión de prueba gratuita

Para empezar a utilizar WebSphere eXtreme Scale, descargue una versión de prueba gratuita. Puede desarrollar aplicaciones innovadores y de alto rendimiento ampliando el concepto de almacenamiento en memoria caché de datos utilizando características avanzadas.

Descarga de versión de prueba

Puede descargar una versión de prueba gratuita de eXtreme Scale, desde Descargar prueba de eXtreme Scale.

Después de descargar y descomprimir la versión de prueba de eXtreme Scale, vaya al directorio gettingstarted y lea el archivo GETTINGSTARTED_README.txt. Esta guía de aprendizaje le ayuda a empezar utilizando eXtreme Scale, crear una cuadrícula en varios servidores y ejecutar algunas aplicaciones sencillas para almacenar y recuperar los datos de una cuadrícula. Antes de desplegar eXtreme Scale en un entorno de producción, hay varias opciones que se deben tener en cuenta, incluidos el número de servidores para utilizar, la cantidad de almacenamiento en cada servidor y la duplicación síncrona o asíncrona.

Guías de programación y administración

*Visión general del producto*describe los conceptos fundamentales para comprender WebSphere eXtreme Scale. Hay disponibles dos guías adicionales que amplían los conceptos descritos en esta guía.

Utilice *Guía de administración* para las tareas de configuración y las administrativas generales y *Guía de programación* para ver descripciones de las API Java para acceder y configurar la cuadrícula eXtreme Scale.

Capítulo 2. Visión general de memoria caché

WebSphere eXtreme Scale puede funcionar como un espacio de proceso de base de datos en memoria, que puede utilizar para proporcionar el almacenamiento en memoria caché en línea para un programa de fondo de la base de datos o para funcionar como una memoria caché secundaria. El almacenamiento en memoria caché en línea utiliza eXtreme Scale como el medio principal para interactuar con los datos. Cuando eXtreme Scale se utiliza como memoria caché complementaria, el programa de fondo se utiliza junto con la cuadrícula de datos. En esta sección se describen distintos conceptos y escenarios de almacenamiento en memoria caché y se explican las topologías disponibles para desplegar una cuadrícula de datos.

Arquitectura de memoria caché: correlaciones, contenedores, clientes y catálogos

Con WebSphere eXtreme Scale, la arquitectura puede utilizar el almacenamiento en memoria caché de datos en memoria local o el almacenamiento en memoria caché de datos de cliente-servidor distribuido.

Para poder funcionar, WebSphere eXtreme Scale necesita una mínima infraestructura adicional. La infraestructura se compone de scripts que instalan, inician y detienen una aplicación Java Platform, Enterprise Edition en un servidor. Los datos colocados en memoria caché se almacenan en el servidor eXtreme Scale, y los clientes se conectan de forma remota al servidor.

Las memorias caché distribuidas ofrecen un mejor rendimiento, disponibilidad y escalabilidad y se puede configurar mediante topologías dinámicas, en los que los servidores se equilibran automáticamente. También puede añadir servidores adicionales sin reiniciar los servidores eXtreme Scale existentes. Puede crear despliegues sencillos o grandes despliegues con terabytes en los que son necesarios miles de servidores.

Correlaciones

Una correlación es un contenedor de pares de clave-valor, que permite a una aplicación almacenar un valor indexado por una clave. Las correlaciones soportan los índices que se pueden añadir a atributos de índice en la clave o el valor. Estos índices son utilizados automáticamente por el tiempo de ejecución de la consulta para determinar el método más eficaz para ejecutar una consulta.

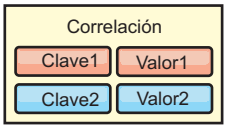


Figura 2. Correlación

Un conjunto de correlaciones es una colección de correlaciones con un algoritmo de particionamiento común. Los datos de las correlaciones se replican en función de la política definida en el conjunto de correlaciones. Un conjunto de relaciones sólo se utiliza en topologías distribuidas y no es necesario en topologías locales.

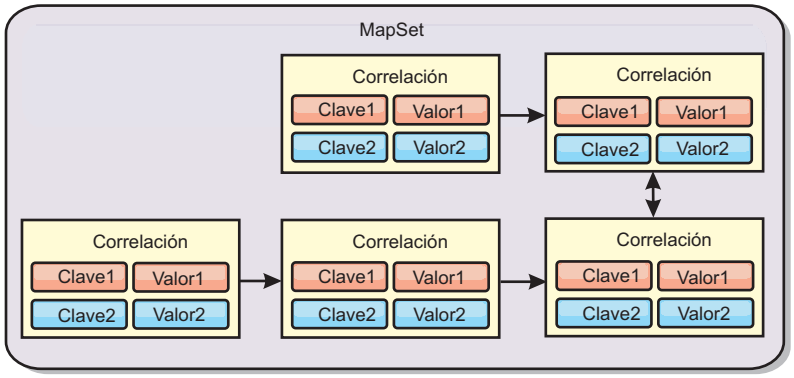


Figura 3. Conjuntos de correlaciones

Un conjunto de correlaciones puede tener asociado un esquema. Un esquema son los metadatos que describen las relaciones entre cada correlación, cuando se utilizan tipos Object homogéneos, o entidad.

WebSphere eXtreme Scale puede almacenar objetos Java serializables en cada una de las correlaciones utilizando la API ObjectMap. Un esquema se puede definir en las correlaciones para identificar la relación entre los objetos de las correlaciones donde cada correlación incluye objetos de un único tipo. La definición de un esquema para las correlaciones es obligatoria para consultar el contenido de los objetos de la correlación. WebSphere eXtreme Scale puede tener varios esquemas de correlación definidos. Consulte la información de la API ObjectMap en *Guía de programación* si desea detalles adicionales.

WebSphere eXtreme Scale también puede almacenar entidades mediante la API EntityManager. Cada entidad se asocia con una correlación. El esquema de un conjunto de correlaciones de entidad se descubre automáticamente utilizando un archivo XML de descriptor de entidad o clases Java anotadas. Cada entidad tiene un conjunto de atributos clave y un conjunto de atributos no clave. Una entidad también puede tener relaciones con otras entidades. WebSphere eXtreme Scale admite relaciones de una a una, de una a varias, de varias a una, y de varias a varias. Cada entidad se correlaciona físicamente con una única correlación del conjunto de correlaciones. Las entidades permiten que las aplicaciones tengan fácilmente gráficos de objetos complejos que abarquen varias correlaciones. Una topología distribuida puede tener varios esquemas de entidad. Consulte la información de la API EntityManager en *Guía de programación* si desea detalles adicionales.

Contenedores, particiones y fragmentos

El contenedor es un servicio que almacena datos de la aplicación para la cuadrícula. Estos datos normalmente se dividen en partes, que se denominan particiones, y se alojan en numerosos contenedores. A cambio, cada contenedor aloja un subconjunto de datos completos. JVM puede alojar uno o más contenedores, y cada contenedor puede alojar varios fragmentos.

Recuerde: planifique el tamaño de almacenamiento dinámico de los contenedores, que albergan todos los datos. Configure los valores de almacenamiento dinámico según corresponda.

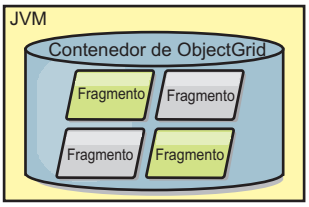


Figura 4. Contenedor

Las particiones alojan un subconjunto de los datos en la cuadrícula. WebSphere eXtreme Scale coloca automáticamente varias particiones en un único contenedor y propaga las particiones a medida que pasan a estar disponibles más contenedores.

Importante: Elija cuidadosamente el número de particiones antes del despliegue final, ya que el número de particiones no se puede cambiar de forma dinámica. Se utiliza un mecanismo hash para localizar las particiones en la red y eXtreme Scale no puede volver a realizar hash de todo el conjunto de datos, una vez que se haya desplegado. Como regla general, se puede sobrestimar el número de particiones

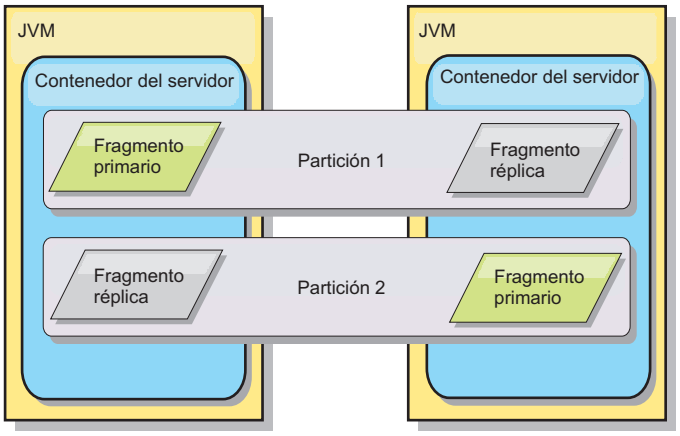


Figura 5. Partición

Los fragmentos son instancias de particiones y tienen uno de los roles siguientes: primario o réplica. El fragmento primario y sus réplicas conforman la manifestación física de la partición. Cada partición tiene varios fragmentos que cada uno de éstos incluye todos los datos contenidos en dicha partición. Un fragmento es el primario y las demás son réplicas, que son copias redundantes de los datos del fragmento primario. Un fragmento primario es la única instancia de partición que permite que las transacciones se graben en la memoria caché. Un fragmento réplica es una instancia "duplicada" de la partición. Recibe actualizaciones de forma síncrona o asíncrona del fragmento primario. El fragmento réplica sólo permite a las transacciones leer de la memoria caché. Las réplicas nunca se alojan en el mismo contenedor que el fragmento primario y por norma no se alojan en la misma máquina que el fragmento primario.

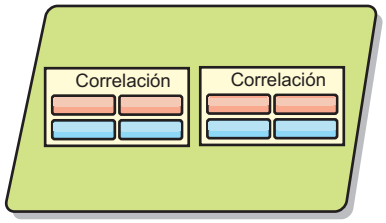


Figura 6. Fragmento

Para aumentar la disponibilidad de los datos, o para aumentar las garantías de persistencia, replique los datos. No obstante, la réplica supone coste en la transacción y cambia rendimiento por disponibilidad. Con eXtreme Scale, puede controlar el coste, ya que se admiten la réplica síncrona y asíncrona, así como modelos de réplica híbridos que usan modalidades de réplica síncrona y asíncrona. Un fragmento réplica síncrona recibe actualizaciones como parte de la transacción del fragmento primario para garantizar la coherencia de los datos. Una réplica síncrona puede doblar el tiempo de respuesta porque la transacción debe confirmar en el fragmento primario y la réplica síncrona antes de que se complete la transacción. Un fragmento réplica asíncrona recibe actualizaciones después de la confirmación de la transacción para limitar el impacto en el rendimiento, pero puede haber pérdida de datos ya que la réplica asíncrona puede estar varias transacciones por detrás del fragmento primario.

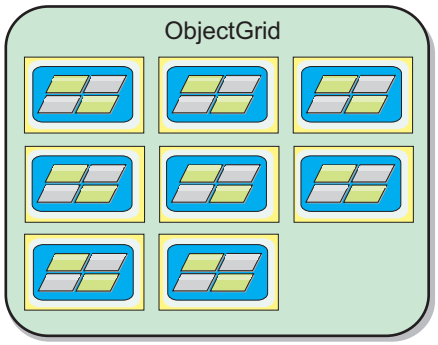


Figura 7. ObjectGrid

Clientes

Los clientes se conectan a un servicio de catálogo, recuperan una descripción de la topología del servidor y se comunican directamente con cada servidor según precisen. Cuando la topología del servidor cambia porque se añaden servidores nuevos o porque han fallado servidores existentes, el servicio de catálogo dinámico direcciona el cliente al servidor apropiado que alberga los datos. Los clientes deben examinar las claves de los datos de la aplicación para determinar a qué partición direccionar la petición. Los clientes pueden leer los datos de diversas particiones en una única transacción. No obstante, los clientes pueden actualizar sólo una única partición en una transacción. Después de que el cliente actualice algunas entradas, la transacción del cliente debe usar esa partición para las actualizaciones.

Las combinaciones posibles de despliegue son las siguientes:

- Existe un servicio de catálogo en su propia cuadrícula de máquinas virtuales Java. Se puede utilizar un único servicio de catálogo para gestionar diversos clientes o servidores de eXtreme Scale.

- Se puede iniciar un contenedor en una JVM por sí mismo o se puede cargar en una JVM arbitraria con otros contenedores para instancias de ObjectGrid distintas.
- Un cliente puede existir en cualquier JVM y comunicarse con una o más instancias de ObjectGrid. También puede existir un cliente en la misma JVM que la de un contenedor.

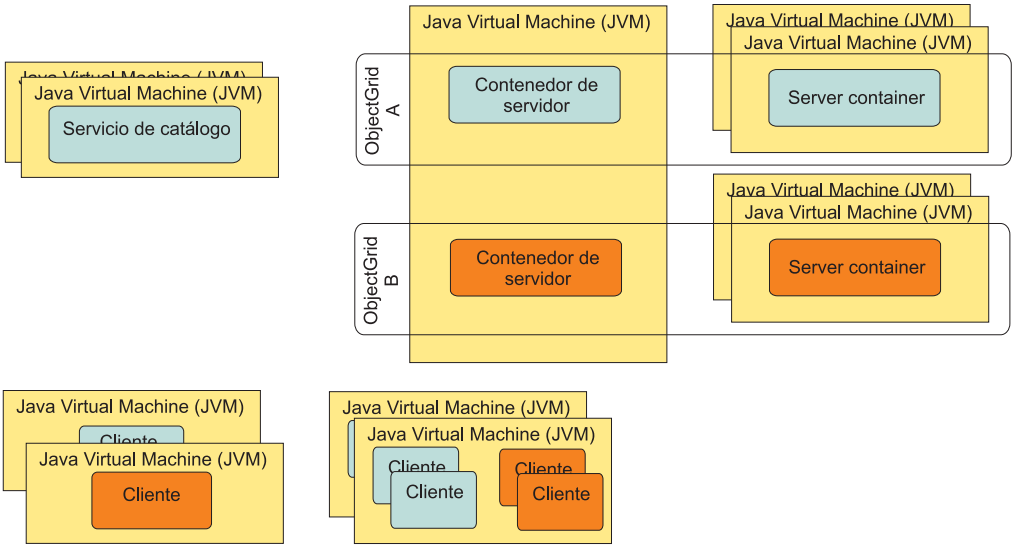


Figura 8. Topologías posibles

Servicios de catálogo (servidores de catálogo)

El servicio de catálogo alberga lógica que debería estar inactiva durante un estado fijo y tiene poca influencia en escalabilidad. El servicio de catálogo se crea para dar servicio a cientos de contenedores que pasan a estar disponibles de forma simultánea y servicios de ejecuciones para gestionar los contenedores.

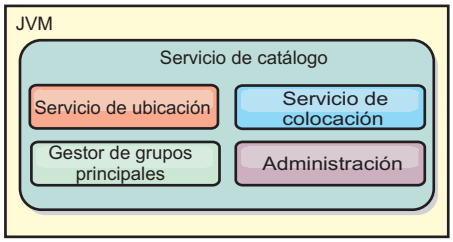


Figura 9. Servicio de catálogo

El catálogo se ocupa de los servicios siguientes:

Servicio de ubicación
El servicio de ubicación proporciona localidad para los clientes que buscan contenedores que alojan aplicaciones y contenedores que buscan registrar aplicaciones alojadas con el servicio de colocación. El servicio de ubicación se ejecuta en todos los miembros de la cuadrícula para escalar hacia fuera esta función.

Servicio de colocación
El servicio de colocación es el sistema nervioso central de la cuadrícula y

es el responsable de asignar fragmentos individuales al contenedor host. El servicio de colocación se ejecuta como uno de los N servicios elegidos en el clúster. Dado que se utiliza la política uno de N, siempre hay exactamente una instancia del servicio de colocación en ejecución. Si se detuviera esa instancia, tomaría el control otro proceso. Todos los estados del servicio de catálogo se replican en todos los servidores que alojan el servicio de catálogo para favorecer la redundancia.

Gestor de grupos principales
El gestor de grupos principales gestiona el agrupamiento de igual para la supervisión de salud, organiza los contenedores en pequeños grupos de servidores y federa automáticamente los grupos de servidores. Cuando un contenedor se pone en contacto en primer lugar con el servicio de catálogo, el contenedor espera a ser asignado a un grupo nuevo o existente de varias máquinas virtuales Java (JVM). Cada grupo de máquinas virtuales Java supervisa la disponibilidad de cada uno de sus miembros a través de las pulsaciones. Uno de los miembros del grupo transmite información de disponibilidad del servicio de catálogo para reaccionar ante anomalías mediante la reasignación y el reenvío de rutas.

Administración
Las cuatro fases que componen la administración del entorno de WebSphere eXtreme Scale son planificación, despliegue, gestión y supervisión. Consulte la publicación *Guía de administración* para obtener más información sobre cada fase.

Para favorecer la disponibilidad, configure un dominio de servicio de catálogo. Un dominio de servicio de catálogo está formado por varias máquinas virtuales Java, incluida una JVM maestra y una serie de máquinas virtuales Java de copia de seguridad.

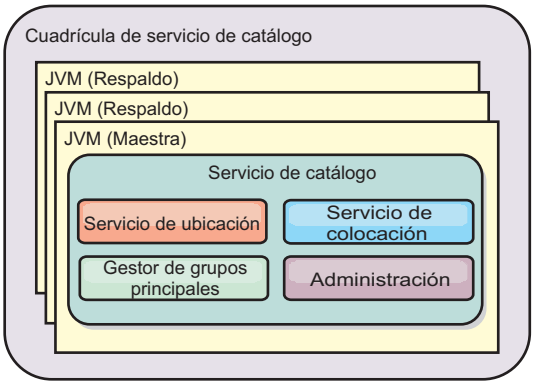


Figura 10. Dominio de servicio de catálogo

Topología de memoria caché: memoria caché en memoria y distribuida

Con WebSphere eXtreme Scale, la arquitectura puede utilizar el almacenamiento en memoria caché de datos en memoria local o el almacenamiento en memoria caché de datos de cliente-servidor distribuido.

Para poder funcionar, WebSphere eXtreme Scale necesita una mínima infraestructura adicional. La infraestructura se compone de scripts que instalan, inician y detienen una aplicación Java Platform, Enterprise Edition en un servidor. Los datos colocados en memoria caché se almacenan en el servidor eXtreme Scale, y los clientes se conectan de forma remota al servidor.

Las memorias caché distribuidas ofrecen un mejor rendimiento, disponibilidad y escalabilidad y se puede configurar mediante topologías dinámicas, en los que los servidores se equilibran automáticamente. También puede añadir servidores adicionales sin reiniciar los servidores eXtreme Scale existentes. Puede crear despliegues sencillos o grandes despliegues con terabytes en los que son necesarios miles de servidores.

Almacenamiento local de memoria caché en memoria

En el caso más sencillo, eXtreme Scale se puede utilizar como una memoria caché de cuadrícula de datos en memoria local (no distribuida). El caso local beneficia especialmente a las aplicaciones de simultaneidad alta donde varias hebras necesitan acceder y modificar los datos transitorios. Los datos conservados en una cuadrícula local de eXtreme Scale se pueden indexar y recuperar mediante el soporte de consulta de WebSphere eXtreme Scale. La capacidad de consultar los datos puede ayudar a los desarrolladores en gran medida cuando trabajan con grandes conjuntos de datos de memoria respecto al soporte limitado de estructura de datos proporcionado con Máquina virtual Java (JVM), que está preparado para ser utilizado tal cual.

La topología de la memoria caché en memoria local para eXtreme Scale se utiliza para proporcionar un acceso coherente y transaccional a los datos temporales de una única máquina virtual Java.

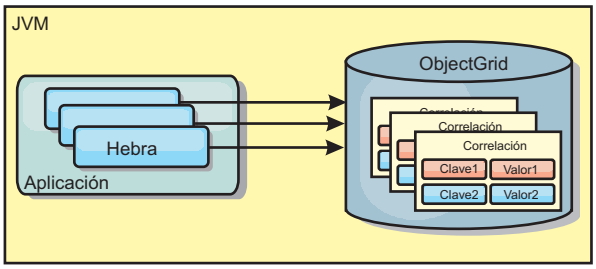


Figura 11. Escenario de memoria caché en memoria local

Ventajas

- Fácil configuración: se puede crear un ObjectGrid a través de un programa o de forma declarativa con el archivo XML descriptor de ObjectGrid o con otras infraestructuras como, por ejemplo, Spring.
- Rápido: cada BackingMap puede adaptarse de forma independiente de modo que la utilización de la memoria y la simultaneidad sean óptimas.
- Es ideal para las topologías de máquina virtual Java única con conjuntos de datos pequeños o para almacenar en memoria caché los datos de acceso frecuente.
- Es transaccional. Las actualizaciones de BackingMap se pueden agrupar en una única unidad de trabajo y se pueden integrar como último participante en transacciones de 2 fases como, por ejemplo, transacciones JTA (Java Transaction Architecture).

Desventajas

- No es tolerante a errores.
- Los datos no se replican. Las memorias caché en memoria son la mejor solución para los datos de referencia de sólo lectura.

- No es escalable. La cantidad de memoria necesaria para la base de datos podría desbordar la máquina virtual Java.
- Se producen problemas al añadir máquinas virtuales Java:
  - Los datos no se pueden particionar fácilmente.
  - Se debe replicar manualmente el estado entre las máquinas virtuales Java o cada instancia podría tener distintas versiones de los mismos datos.
  - La operación de invalidación es muy costosa.
  - Cada memoria caché se debe calentar de forma independientemente. El calentamiento es el periodo de carga de un conjunto de datos, de forma que la memoria caché se rellena con datos válidos.

Cuándo se debe utilizar

La topología de despliegue de la memoria caché en memoria local sólo se debe utilizar cuando la cantidad de datos que se deben almacenar en memoria caché es pequeña (cabe en una única máquina virtual Java) y es relativamente estable. Los datos obsoletos deben tolerarse con este acercamiento. El uso de desalojadores para mantener en la memoria caché los datos usados con más frecuencia o los más recientes puede ayudar a mantener pequeño el tamaño de la memoria caché y a aumentar la relevancia de los datos.

Memoria caché en memoria local replicada por un igual

Para una memoria caché de WebSphere eXtreme Scale local, debe asegurarse de que la memoria caché esté sincronizada si hay varios procesos con instancias de memoria caché independientes. Para hacerlo, habilite una memoria caché replicada por un igual con JMS.

WebSphere eXtreme Scale incluye dos plug-ins que propagan automáticamente los cambios de las transacciones entre instancias de ObjectGrid de un igual. El plug-in JMSObjectGridEventListener propaga automáticamente los cambios de eXtreme Scale utilizando JMS (Java Messaging Service).

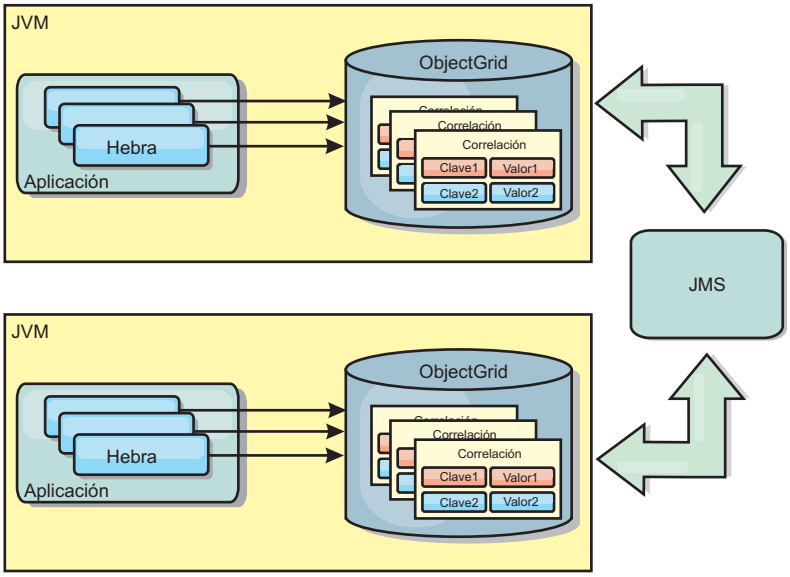


Figura 12. La memoria caché duplicada por un igual con los cambios que se propagan con JMS

Si ejecuta un entorno de WebSphere Application Server, el plug-in TranPropListener también está disponible. El plug-in TranPropListener utiliza el gesto de alta disponibilidad (HA) para propagar los cambios en cada instancia de memoria caché eXtreme Scale de igual.

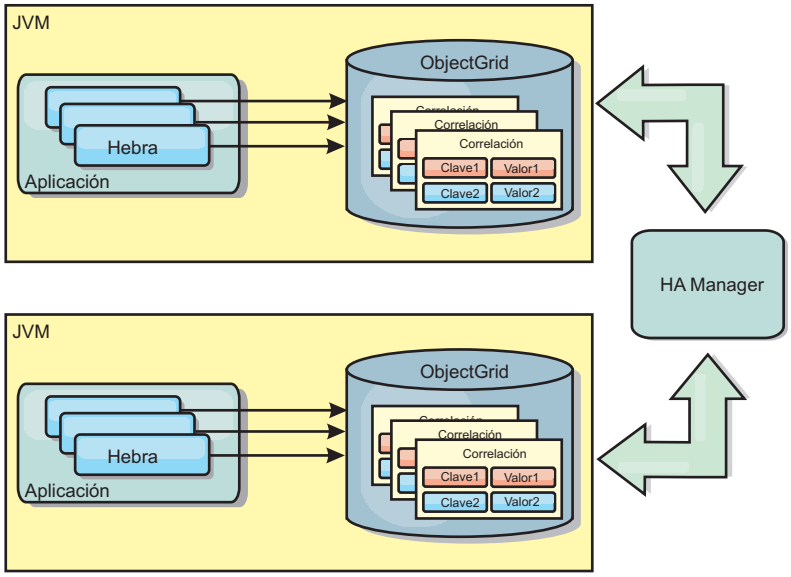


Figura 13. La memoria caché duplicada por un igual con los cambios propagados con el High Availability Manager.

Ventajas

- Los datos son más válidos porque se actualizan con más frecuencia.
- Con el plug-in TranPropListener, igual que el entorno local, eXtreme Scale se puede crear a través de programa o de forma declarativa con el archivo XML de descriptor de despliegue de eXtreme Scale o con otras infraestructuras como, por ejemplo, Spring. La integración con el High Availability Manager se realiza de forma automática.
- Cada BackingMap se puede ajustar independientemente para obtener un uso y una simultaneidad óptimos de la memoria.
- Las actualizaciones de BackingMap se pueden agrupar en una única unidad de trabajo y se pueden integrar como último participante en transacciones de 2 fases como, por ejemplo, transacciones JTA (Java Transaction Architecture).
- Ideal para topologías de pocas JVM con un conjunto de datos razonablemente pequeño o para almacenar en memoria caché datos de acceso frecuente.
- Los cambios en eXtreme Scale se duplican en todas las instancias de eXtreme Scale de igual. Los cambios son coherentes mientras se utilice una suscripción duradera.

Desventajas

- La configuración y el mantenimiento de JMSObjectGridEventListener pueden ser complejos. eXtreme Scale puede crearse mediante programa o de forma declarativa con el archivo XML de descriptor de despliegue de eXtreme Scale o con otras infraestructuras como Spring.
- No es escalable: el volumen de memoria que requiere la base de datos puede desbordar la JVM.
- Funciona de forma incorrecta cuando se añade Máquinas virtuales Java:
  - Los datos no se pueden particionar fácilmente.

- La operación de invalidación es muy costosa.
- Cada memoria caché debe calentarse de manera independiente.

Cuándo se debe utilizar

Esta topología de despliegue sólo se debe utilizar cuando la cantidad de datos que se va a almacenar en la memoria caché es pequeña (cabe en una única JVM) y es relativamente estable.

Memoria caché distribuida

WebSphere eXtreme Scale se usa con más frecuencia como una memoria caché compartida, para proporcionar acceso transaccional a los datos en varios componentes donde, de lo contrario, se utilizará una base de datos tradicional. La memoria caché compartida elimina la necesidad de configurar una base de datos.

La memoria caché es coherente porque todos los clientes ven los mismos datos en la memoria caché. Cada dato se almacena exactamente en un servidor de la memoria caché, lo que evita tener copias innecesarias que podrían contener posiblemente distintas versiones de los datos. Además una memoria caché coherente puede contener más datos, a medida que se añadan más servidores a la cuadrícula y se amplía de forma lineal, a medida que la cuadrícula crece en tamaño. Puesto que los clientes acceden a los datos desde esta cuadrícula con llamadas de procedimiento remotas, también se conoce como memoria caché remota (o memoria caché lejana). A través de la partición de datos, cada proceso contiene un subconjunto exclusivo del conjunto de datos total. Las cuadrículas más grandes pueden contener más datos y dar servicio a más solicitudes de esos datos. La coherencia también elimina la necesidad de pasar los datos de invalidación alrededor de la cuadrícula porque no hay datos obsoletos. La memoria caché coherente sólo contiene la copia más reciente de cada dato.

Si ejecuta un entorno WebSphere Application Server, el plug-in TranPropListener también está disponible. El plug-in TranPropListener utiliza el componente de alta disponibilidad (HA Manager) de WebSphere Application Server para propagar los cambios en cada instancia de memoria caché de ObjectGrid de igual.

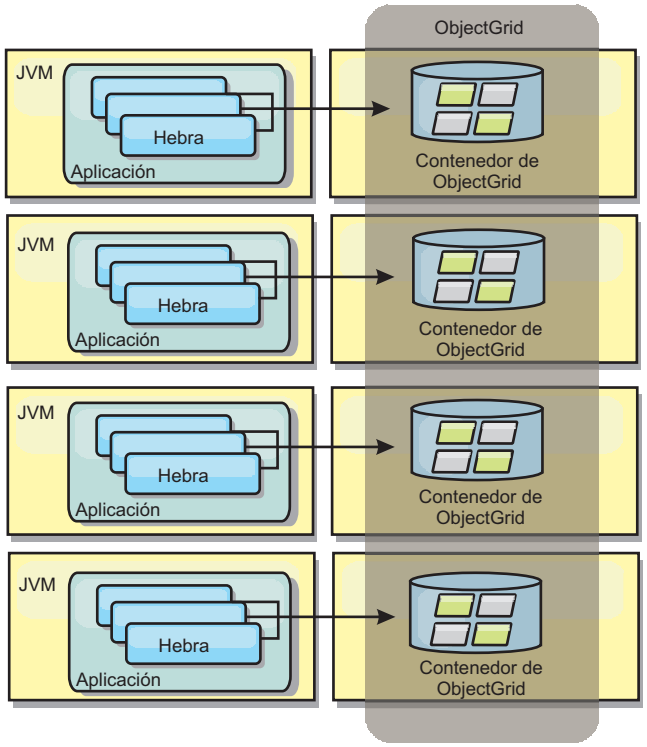


Figura 14. Memoria caché distribuida

Memoria caché cercana

De forma opcional, los clientes pueden tener una memoria caché local en línea cuando se utiliza eXtreme Scale en una topología distribuida. Esta memoria caché opcional se llama memoria caché cercana, es un ObjectGrid independiente en cada cliente, que sirve como memoria caché para la memoria caché remota del lado del servidor. La memoria caché cercana se habilita de manera predeterminada al configurar el bloqueo como optimista o ninguno, y no puede utilizarse si se configura como pesimista.

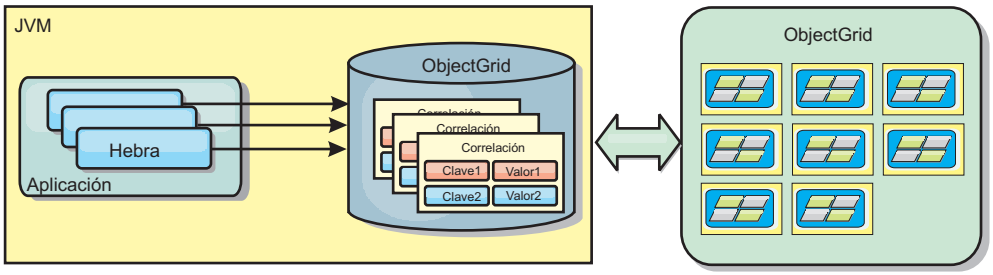


Figura 15. Memoria caché cercana

Una memoria caché cercana es muy rápida porque proporciona un acceso en memoria a un subconjunto de todos los conjuntos de datos almacenados en memoria caché que se almacenan de forma remota en los servidores eXtreme Scale. La memoria caché cercana no está particionada y contiene datos de cualquiera de las particiones eXtreme Scale remotas. WebSphere eXtreme Scale puede tener hasta tres niveles de memoria caché del modo siguiente.

1. La memoria caché de nivel de transacción contiene todos los cambios de una única transacción. La memoria caché de transacción contiene una copia de

trabajo de los datos hasta que la transacción se confirma. Cuando una transacción de cliente solicita datos de un objeto ObjectMap, primero se comprueba la transacción.

2. La memoria caché cercana en el nivel de cliente contiene un subconjunto de datos del nivel de servidor. Cuando un nivel de transacción no tiene los datos, los datos se captan de la memoria caché cercana si están disponibles y se insertan en la memoria caché de transacción.
3. La cuadrícula del nivel del servidor contiene la mayoría de los datos y se comparte entre todos los clientes. El nivel de servidor puede particionarse, lo que permite almacenar en memoria caché un gran volumen de datos. Cuando la memoria caché cercana de cliente no tiene los datos, éstos se captan del nivel de servidor y se insertan en la memoria caché de cliente. El nivel de servidor también tiene un plug-in Loader. Si la cuadrícula no tiene los datos solicitados, se invoca el Loader y los datos resultantes se insertan del almacén de datos de proceso de fondo en la cuadrícula.

Para inhabilitar la memoria caché cercana, establezca el atributo numberOfBuckets en 0 en la configuración de descriptor de eXtreme Scale de alteración temporal del cliente. Consulte el tema sobre el bloqueo de entrada de correlación para ver detalles sobre las estrategias de bloqueo de eXtreme Scale. La memoria caché cercana también se puede configurar para tener una política de desalojo separada y distintos plug-ins mediante una configuración de descriptor de eXtreme Scale de alteración temporal del cliente.

Ventaja

- Un tiempo de respuesta rápido porque todos los accesos a los datos son locales.

Desventajas

- Aumenta la duración de los datos obsoletos.
- Debe usar un desalojador para invalidar los datos con el fin de evitar quedarse sin memoria.

Cuándo se debe utilizar

Debe usarse cuando el tiempo de respuesta sea importante y puedan tolerarse los datos obsoletos.

Memoria caché incorporada

Las cuadrículas de eXtreme Scale pueden ejecutarse dentro de procesos existentes como servidores eXtreme Scale incorporados o pueden gestionarse como procesos externos. Las cuadrículas incorporadas son útiles cuando se ejecutan en un servidor de aplicaciones como, por ejemplo, WebSphere Application Server. Puede iniciar los servidores eXtreme Scale que no están incorporados utilizando los scripts de la línea de mandatos y ejecutarlos en un proceso Java.

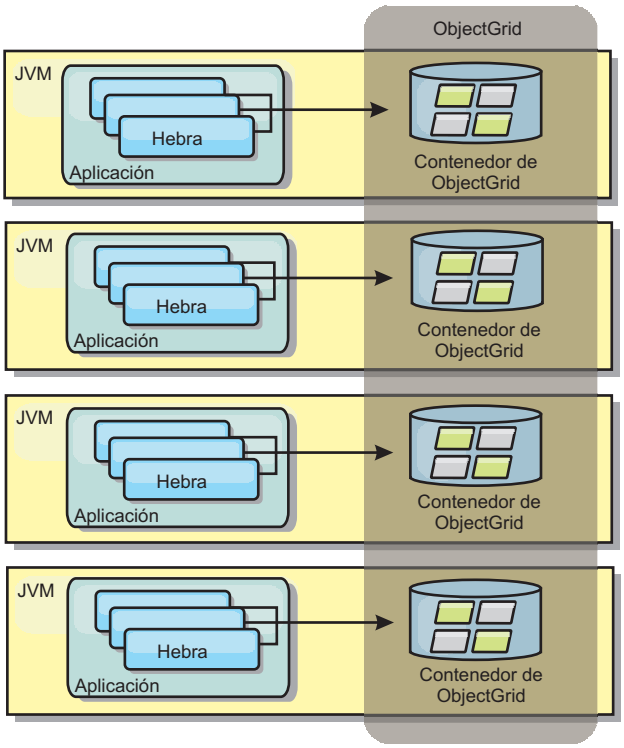


Figura 16. Memoria caché incorporada

Ventajas

- Administración simplificada ya que hay menos procesos que deban gestionarse.
- Despliegue de aplicaciones simplificado ya que la cuadrícula utiliza el cargador de clases de la aplicación cliente.
- Admite particionamiento y alta disponibilidad.

Desventajas

- Aumenta el uso de la memoria en procesos de cliente ya que todos los datos se colocan en el proceso.
- Aumenta el uso de la CPU para dar servicio a las solicitudes de los clientes.
- Es más difícil manejar las actualizaciones de las aplicaciones ya que los clientes utilizan los mismos archivos JAR (Java Archive) de aplicación que los servidores.
- Menos flexible. Escalar clientes y servidores de cuadrícula no puede aumentar a la misma velocidad. Si los servidores se definen externamente, puede tener más flexibilidad al gestionar el número de procesos.

Cuándo se debe utilizar

Utilice cuadrículas incorporadas cuando haya suficiente memoria libre en el proceso de cliente para datos de cuadrícula y posibles datos de sustitución por anomalía.

Si desea más información, consulte el tema sobre cómo habilitar el mecanismo de invalidación de cliente en *Guía de administración*.

Topologías de réplica de cuadrícula con varios maestros (AP)

Con la característica réplica asíncrona con varios maestros, dos o más cuadrículas pueden pasar a ser duplicados exactos las unas de las otras. Esta duplicación se lleva a cabo con la réplica asíncrona entre enlaces que conectan juntas las cuadrículas. Cada cuadrícula se hospeda en un "dominio" completamente independiente, que procesa su propio servicio de catálogo, los servidores de contenedor y un nombre de dominio único. Con la característica réplica asíncrona con varios maestros, puede utilizar los enlaces para interconectar una colección de estos dominios y luego sincronizarlos, mediante la réplica en los enlaces.eXtreme Scale permite construir casi cualquier topología, porque la definición de enlaces entre dominios le corresponde a usted.

Dominios: cuadrículas con características específicas

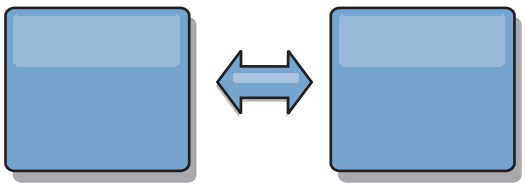
Las cuadrículas utilizadas en topologías de réplica con varios maestros se conocen como *dominios*. Todos los dominios deben tener estas características:

- Tener un servicio de catálogo privado con un nombre de dominio único
- Tener el mismo nombre de cuadrícula que otras cuadrículas del dominio
- Tener el mismo número de particiones que otras cuadrículas del dominio
- Ser una cuadrícula FIXED_PARTITION (no se puede hacer una réplica de las cuadrículas PER_CONTAINER)
- Tener el mismo número de particiones (podrían tener o no tener el mismo número y tipos de réplicas)
- Tener los mismos tipos de datos de los que se va a hacer una réplica que otras cuadrículas del dominio
- Tener los mismos nombres de conjunto de correlaciones, nombres de correlación y plantillas de correlación dinámica que otras cuadrículas del dominio

Después de que se han iniciado los dominios de la topología, se hará una réplica de todas las cuadrículas con las características anteriores. Recuerde que se omitirá su política de réplica.

Enlaces que conectan los dominios

Una infraestructura de cuadrícula de réplica es un gráfico de dominios conectados con enlaces bidireccionales entre ellos. Un enlace permite que dos dominios intercambien cambios de datos. Por ejemplo, la topología más sencilla es un par de dominios con un solo enlace entre ellos. Se nombran los dominios comenzando por una “A” y seguidamente una “B”, etc., desde la izquierda. En enlace podría atravesar una red de área amplia (WAN), que abarca largas distancias. Si se interrumpe el enlace, se pueden realizar todavía cambios en los datos en cualquier dominio. Más tarde se reconcilian los cambios, cuando el enlace se vuelve a conectar a los dominios. Los enlaces intentan volverse a conectar automáticamente si se interrumpe la conexión de red.

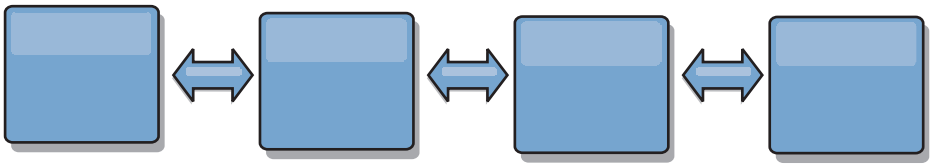


Después de haber configurado los enlaces, eXtreme Scale intentará primero hacer que cada dominio sea idéntico y luego intentará mantener las condiciones idénticas

cuando se produzcan cambios en algún dominio. El objetivo de eXtreme Scale es que cada dominio sea un duplicado exacto de cada otro dominio conectado mediante los enlaces. Los enlaces de réplica entre los dominios ayudan a garantizar que los cambios realizados en un dominio se copian en los otros dominios.

Topologías de línea

Aunque está entre las topologías más sencillas, una topología de línea muestra algunas cualidades de los enlaces. En primer lugar, no es necesario que un dominio esté conectado directamente a todos los demás dominios para recibir los cambios. El Dominio B extraerá los cambios del Dominio A. El Dominio C recibe los cambios del Dominio A a través del Dominio B, que conecta los Dominios A y C. De forma similar, el Dominio D recibe los cambios de los otros dominios a través del Dominio C. Esta capacidad esparce la carga de distribuir los cambios lejos del origen de los cambios.



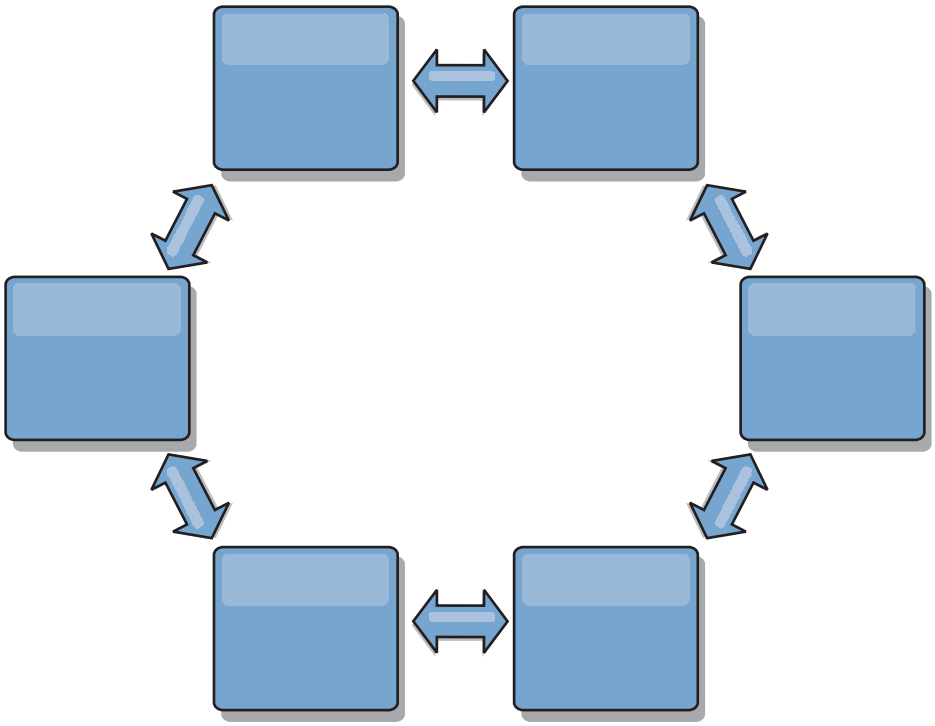
Recuerde que si se produce un error en el Dominio C, se producen los sucesos siguientes.

1. El Dominio D se quedará huérfano hasta que se reinicie el Dominio C
2. El Dominio C se sincronizará a sí mismo con el Dominio B, que es una copia del Dominio A
3. El Dominio D utilizará el Dominio C para sincronizarse a sí mismo con los cambios en los Dominios A y B producidos cuando el Dominio D estaba huérfano (cuando el Dominio C estaba inactivo)

Al final, los Dominios A, B, C y D serán idénticos entre sí de nuevo.

Topologías de anillo

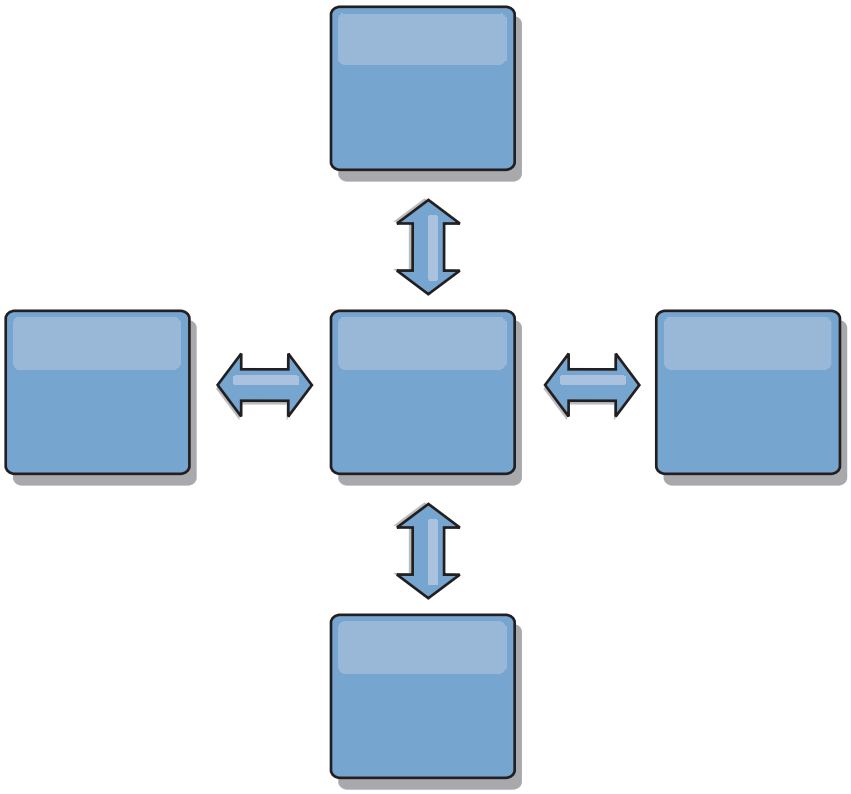
Las topologías de anillo son un ejemplo de una topología más flexible. Aunque un dominio o un enlace único puede producir un error, los dominios supervivientes todavía pueden obtener los cambios viajando por el anillo, lejos de la anomalía. Cada dominio tiene dos enlaces con los otros dominios. Cada dominio tiene a lo sumo dos enlaces, no importa lo grande que sea una topología de anillo. La latencia para propagar los cambios puede ser elevada, porque es posible que los cambios de un dominio en particular tengan que viajar por varios dominios antes de que todos los dominios vean los cambios. Una topología de línea tiene el mismo problema.



Representa una topología de anillo más sofisticada, con un dominio raíz en el dentro del anillo. El dominio raíz funciona como un centro de intercambio de información central, mientras que los demás dominios actúan como centros de intercambio de información remotos para los cambios que se producen en el dominio raíz. El dominio raíz puede arbitrar los cambios entre los dominios. Si una topología de anillo contiene más de un anillo alrededor de un dominio raíz, este último solo puede arbitrar los cambios entre los dominios del anillo más recóndito. No obstante, los resultados del arbitraje se diseminan a los dominios de los demás anillos.

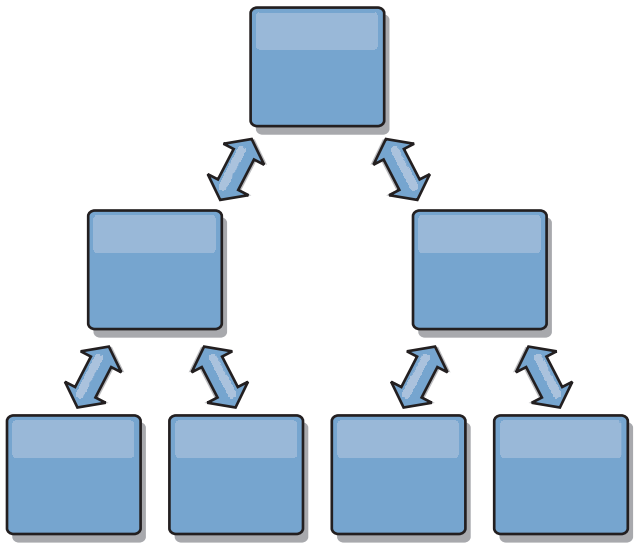
Topologías de hub y radio

Una topología de hub y radio tiene mejor latencia, lo que significa que los cambios viajan a lo sumo por un dominio intermedio (el hub), pero introduce algún otro problema. Tiene un dominio central que actúa como un hub. Este "dominio de hub" se conecta a todos los "dominios de radio" con un enlace. Claramente, la carga de distribuir los cambios entre los dominios recae sobre el hub. El hub actúa como un centro de intercambio de información para las colisiones, una configuración que puede ser importante para determinados casos. En un entorno con una alta tasa de actualizaciones, quizá sea necesario que el hub se ejecute en más hardware que los radios, así puede mantener el ritmo. eXtreme Scale está diseñado para escalar de forma lineal, lo que significa que puede ampliar el hub, según sea necesario, sin dificultad. No obstante, si el hub produce un error, no se distribuirán los cambios hasta que no se reinicie el hub. Los cambios en los dominios de radio se distribuirán después de que se vuelva a conectar el hub.



Topologías de árbol

Un último ejemplo de topología es un árbol dirigido acíclico. Acíclico significa que no hay ciclos ni bucles. Dirigido significa que existen los enlaces solo entre padres e hijos. Esta configuración puede resultar de utilidad para las topologías con tantos dominios que no es práctico tener un hub central conectado a todos los radios posibles, o en el que necesita la capacidad de añadir dominios hijo sin actualizar el dominio raíz.



Esta topología puede tener todavía un centro de intercambio de información central en el dominio raíz pero el segundo nivel puede actuar como centros de intercambio de información remotos para los cambios que se producen en el

dominio por debajo de ellos. El dominio raíz puede arbitrar los cambios entre los dominios sólo en el segundo nivel. También son posibles los árboles N-arios. Un árbol N-ario tiene N hijos en cada nivel. Cada dominio tiene una diseminación de N.

Consideraciones sobre arbitraje en el diseño de topología

Se podrían producir colisiones de cambio si se pueden cambiar en dos lugares a la vez los mismos registros. Configure todos los dominios para que tengan aproximadamente la misma cantidad de recursos de CPU, memoria y red. Podría observar que los dominios que realizan una gestión de colisiones de cambios (arbitraje) utilizan más recursos que otros dominios. Las colisiones se detectan automáticamente. Se resuelven utilizando uno de estos dos mecanismos:

- **Árbitro de colisión predeterminado.** El protocolo predeterminado es utilizar los cambios del dominio con un nombre de dominio menor alfabéticamente. Por ejemplo, si el dominio A y el B generan un conflicto de un registro, se pasa por alto el cambio del dominio B. El dominio A conserva su versión y el registro en el dominio B se cambia para coincidir con el registro del dominio A. Esto se aplica también para las aplicaciones en las que los usuarios o las sesiones se enlazan normalmente o tienen afinidad con una de las cuadrículas.
- **Árbitro de colisión personalizado.** Las aplicaciones pueden proporcionar un árbitro personalizado. Cuando un dominio detecta una colisión, invoca el árbitro. Para obtener información sobre cómo desarrollar un 'buen' árbitro personalizado, consulte Desarrollo de árbitros personalizados para la réplica con varios maestros.

Para las topologías en que son posibles las colisiones, considere utilizar una topología de hub y red o una topología de árbol. Estas dos topologías son propicias para impedir colisiones sin fin, que pueden suceder cuando:

1. Varios dominios sufren una colisión
2. Cada dominio resuelve la colisión de forma local, que produce revisiones
3. Las revisiones colisionan, con lo que se producen revisiones de revisiones
4. Y así sucesivamente, porque las revisiones se propagan entre los distintos dominios intentando alcanzar la sincronicidad

Para evitar las colisiones sin fin, elija un dominio específico – un *dominio de arbitraje* – como manejador de colisiones para un subconjunto de dominios. Por ejemplo, una topología de hub y radio podría utilizar el hub como manejador de colisiones. El manejador de colisiones de radio pasa por alto todas las colisiones detectadas por los dominios de radio. El dominio de hub creará revisiones, con lo que se evitan las revisiones de colisión fuera de control. El dominio asignado para manejar las colisiones debe enlazarse a todos los dominios de los que tiene la responsabilidad de resolver las colisiones. En una topología de árbol, los dominios padre internos resuelven las colisiones de sus hijos inmediatos. A diferencia con esta topología, si utiliza una topología de anillo, no puede designar un dominio en el anillo como solucionador.

En la tabla siguiente se resumen los enfoques de arbitraje que son más compatibles con distintas topologías.

Tabla 3. Enfoques de arbitraje. En esta tabla se indica si el arbitraje de la aplicación es compatible con distintas tecnologías.

Topología	¿Arbitraje de aplicación?	Notas
Una línea de dos dominios	Sí	Seleccione un dominio como árbitro.
Una línea de tres dominios	Sí	El dominio del medio debe ser el árbitro. Piense en el dominio del medio como el hub de una topología sencilla de hub y radio.
Una línea de más de tres dominios	No	No se admite el arbitraje de aplicaciones.
Un hub con N radios	Sí	El hub con enlaces a todos los radios debe ser el dominio de arbitraje.
Un anillo de N dominios	No	No se admite el arbitraje de aplicaciones.
Un árbol dirigido acíclico (árbol N-ario)	Sí	Todos los nodos raíz deben arbitrar sus descendientes directos solo.

Consideraciones sobre enlaces en el diseño de topología

De forma ideal, una topología incluye el número mínimo de enlaces cuando optimiza los compromisos entre las características de latencia de cambios, tolerancia a errores y rendimiento.

- **Latencia de cambios**

  La latencia de cambios viene determinada por el número de dominios intermedios por los que debe pasar un cambio antes de llegar a un dominio concreto.

  Una topología tiene la mejor latencia de cambios cuando elimina los dominios intermedios enlazando cada dominio a todos los demás dominios. No obstante, un dominio debe realizar un trabajo de réplica en proporción a su número de enlaces. Para las topologías de gran tamaño, el verdadero número de enlaces a definir podría representar una carga administrativa.

  La velocidad a la que se copia un cambio en otros dominios depende de factores adicionales, como:

  – CPU y ancho de banda en el dominio de origen
  – El número de dominios intermedios y enlaces entre el dominio de origen y de destino
  – Los recursos de CPU y de red disponibles para los dominios de origen, destino e intermedio

- **Tolerancia a errores**

  La tolerancia a errores viene determinada por el número de vías de acceso que existen entre dos dominios para la réplica de cambios.

  Si solo existe un único enlace entre dominios y el enlace produce un error, no se propagan los cambios. Si el enlace único de un dominio a otro pasa por dominios intermedios, no se propagan los cambios si alguno de los dominios intermedios está inactivo.

  Considere la topología de línea con tres dominios A, B y C:

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

  Si cualquiera de estas condiciones se mantiene, el Dominio C no verá los cambios de A:

- El dominio A está activo y el dominio B está inactivo
- El enlace entre A y B está inactivo
- El enlace entre B y C está inactivo

A diferencia con esta topología, la topología de anillo permite que cada dominio extraiga los cambios de cualquier dirección.

```
A <-> B <-> C <-> de nuevo a A
```

Por ejemplo, si el dominio B está inactivo, el dominio C todavía puede extraer los cambios directamente del dominio A.

Un diseño de hub y radio es propenso a que el hub se quede inactivo porque todos los cambios pasan alrededor a través del hub. No obstante, merece la pena recordar que un solo dominio sigue siendo una cuadrícula totalmente tolerante a errores que podría tener anomalías graves como problemas de WAN o del centro de datos físico.

- **Rendimiento**

  El número de enlaces definidos en un dominio influye en el rendimiento. Si hay más enlaces se utilizan más recursos y a raíz de esto podría disminuir el rendimiento. La capacidad de extraer los cambios de un dominio A a través de otros dominios descarga de forma efectiva el domino A de replicar sus transacciones por todas partes. *La carga de distribución de cambios en un dominio está limitada al número de enlaces que utiliza. No se puede hacer nada con el número de dominios de la topología.* Esta propiedad proporciona escalabilidad y permite que la carga de la distribución de cambios la compartan los dominios de la topología, en lugar de situar la carga en un solo dominio.

  Un dominio puede extraer los cambios de forma indirecta a través de otros dominios. Considere una topología de línea con cinco dominios.

  ```
  A <=> B <=> C <=> D <=> E
  ```

  - A extrae los cambios de B, C, D y E a B
  - B extrae los cambios de A y C directamente y los cambios de D y E a C
  - C realiza los cambios de B y D directamente y los cambios de A a B y de E a D
  - D extrae los cambios de C y E directamente y los cambios de A y B a C
  - E extrae los cambios de D directamente, y los cambios de A, B y C a D

  La carga de distribución en los dominios A y E es menor, porque cada uno tiene un enlace solo a un solo dominio. La carga de distribución en los dominios B, C y D tiene el doble de carga en los dominios A y E porque los dominios B, C y D tienen cada uno un enlace a dos dominios. Esta distribución de cargas permanecería constante, aun cuando la línea contuviera 1000 dominios, porque la carga depende del número de enlaces de cada dominio, no del número global de dominios de la topología.

Consideraciones sobre el rendimiento

Tenga en cuenta estas limitaciones al utilizar las topologías de réplica con varios maestros.

- **Ajuste de la distribución de cambios** (se describe anteriormente)
- **Latencia de réplica** (se describe anteriormente)
- **Rendimiento de enlace de réplica** eXtreme Scale crea un único socket TCP/IP entre cualquier par de JVM. Todo el tráfico entre esas JVM se produce en ese socket, incluida la réplica con varios maestros. Dado que los dominios se alojan en N JVM de contenedor como mínimo, si se proporcionan como mínimo N

enlaces TCP a los dominios de igual, los dominios con mayor número de contenedores tendrán niveles más altos de rendimiento de réplica. Más contenedores significa más recursos de CPU y de red.

- **Ajuste de la ventana deslizante TCP y RFC 1323** Si se habilita el soporte de RFC 1323 en los dos extremos de un enlace se permiten más datos para una transmisión de ida y vuelta, con lo que aumenta el rendimiento. La técnica amplía la capacidad de la ventana en un factor de aproximadamente 16.000.

  Recuerde que los sockets TCP utilizan un mecanismo de ventana deslizante para controlar el flujo de los datos en bloque, que suele limitar el socket a 64 KB para un intervalo de transmisiones de ida y vuelta. Si el intervalo de transmisión de ida y vuelta es de 100 mseg., el ancho de banda está limitado a 640 KB/segundo sin un ajuste adicional. El uso de todo el ancho de banda disponible en un enlace podría requerir un ajuste específico de un sistema operativo. La mayoría de sistemas operativos incluyen parámetros de ajuste, incluidas las opciones de RFC 1323, para mejorar el rendimiento en enlaces de alta latencia.

  Varios factores pueden afectar al rendimiento de la réplica:

  – La velocidad a la que eXtreme Scale puede hacer los cambios.
  – La velocidad a la que eXtreme Scale puede atender la solicitudes de extracción de réplica.
  – La capacidad de la ventana deslizante.
  – El ajuste del almacenamiento intermedio de red en los dos extremos del enlace para permitir que eXtreme Scale extraiga los cambios en el socket lo más ágil posible.
- **Serialización de objetos** Todos los datos deben ser serializables. Si un dominio no utiliza COPY_TO_BYTES, el dominio debe utilizar la serialización de Java u ObjectTransformers para optimizar el rendimiento de la serialización.
- **Compresión** eXtreme Scale comprime todos los datos enviados entre dominios de forma predeterminada. No hay opción para inhabilitar la compresión en el release actual.
- **Ajuste de memoria** *El uso de memoria para una topología de réplica con varios maestros es muy independiente del número de dominios de la topología.*

  La habilitación de la réplica con varios maestros añade una sobrecarga fija por entrada Map para gestionar el mantenimiento de versiones. Cada contenedor realiza un seguimiento también de una cantidad fija de datos de cada dominio de la topología. Una topología con dos dominios utiliza aproximadamente la misma memoria que una topología con 50 dominios. eXtreme Scale no utiliza registros de reproducción o colas similares en su implementación, lo que significa que si un enlace de réplica no está disponible durante un periodo de tiempo sustancial, no se produce un aumento en tamaño de la estructura de datos, a la espera de reanudar la réplica cuando se inicia el enlace.

Varios centros de datos con FIXED_PARTITION

Ahora puede utilizar una cuadrícula FIXED_PARTITION entre dos o más centros de datos. Cada centro de datos necesita su propio dominio, en lo que se refiere a réplica con varios maestros. Cada centro de datos lee y graba los datos en el dominio local. Estos cambios se propagarán a los otros centros de datos con los enlaces que ha definido.

Clientes totalmente replicados

En esta variación de topología interviene un par de servidores eXtreme Scale que se ejecutan como un hub. Todos los clientes crean una cuadrícula de contenedor

única autocontenida con un catálogo en la JVM cliente. El cliente utiliza su cuadrícula para conectarse al catálogo de hub, que hace que el cliente se sincronice con el hub en cuanto el cliente obtiene una conexión al hub.

Los cambios realizados por el cliente son locales al cliente y se hace una réplica de ellos asíncrona en el hub. El hub actúa como un dominio de arbitraje, que distribuye los cambios a todos los clientes conectados. La topología de clientes totalmente replicados proporciona una buena memoria caché L2 para un correlacionador relacional de objetos, como OpenJPA. Los cambios se distribuirán rápidamente entre las JVM cliente por el hub. Siempre que el tamaño de la memoria caché se pueda incluir en el espacio de almacenamiento dinámico disponible de los clientes, esta topología es una buena arquitectura para este estilo de L2.

Utilice varias particiones para escalar el dominio del hub en varias JVM, si es necesario. Dado que todos los datos todavía deben caber en una sola JVM cliente, el uso de varias particiones aumenta la capacidad del hub para distribuir y arbitrar los cambios, pero no cambia la capacidad de un dominio único.

Limitaciones

Tenga en cuenta estas limitaciones al determinar si va a utilizar y cómo utilizar las topologías de réplica con varios maestros.

- **Tenga cuidado al configurar los cargadores de clases con varios dominios**

  Los dominios deben tener acceso a todas las clases que se utilizan como claves y valores. Todas las dependencias se deben reflejar en todas las vías de acceso de clases de las JVM de contenedor de cuadrícula para todos los dominios. Si un plug-in CollisionArbiter recupera el valor para una entrada de memoria caché, las clases de los valores deben estar presentes para el dominio que invoca el árbitro.

- **No se recomienda utilizar cargadores**

  Los cargadores se pueden utilizar para hacer de interfaz de los cambios entre una cuadrícula y una base de datos. Es improbable que todas las cuadrículas (dominios) de una topología se den en combinación geográfica con la misma base de datos. La latencia de WAN y otros factores podrían representar este caso de uso no deseable.

  La carga previa de cuadrícula es otro problema que requiere un diseño cuidadoso. Normalmente, cuando se reinicia una cuadrícula, se vuelve a cargar previamente de nuevo. No es necesaria la precarga o incluso no es deseable al utilizar la réplica con varios maestros. En cuanto un dominio está en línea, se vuelve a cargar automáticamente a sí mismo con el contenido de los dominios a los que está enlazado. Como consecuencia, no es necesario iniciar una precarga manual para una cuadrícula que es un dominio de una topología de réplica con varios maestros.

  Los cargadores suelen seguir las reglas de inserción y actualización. Con la réplica con varios maestros, las inserciones se deben tratar como fusiones. Cuando se extraen los datos de forma remota después de que se inicia un dominio, los datos existentes se 'insertarán' en el dominio local. Dado que estos datos podrían estar ya en la base de datos local, una inserción típica producirá un error con una excepción de clave duplicada en la base de datos. En su lugar, se debe utilizar la semántica de fusión.

  eXtreme Scale se puede configurar para hacer una precarga basada en fragmentos, con los métodos de precarga en los plug-in Loader. No utilice esta técnica en una topología de réplica con varios maestros. En su lugar, utilice una

precarga basada en cliente cuando se inicia la topología (inicialmente). Permita que la topología con varios maestros renueve los dominios reiniciados con una copia actual que está almacenada en otros dominios de la topología. Después de que se han iniciado los dominios, la topología con varios maestros se encarga de conservarlos en sincronismo.

- **No se admite EntityManager**

  Un conjunto de correlaciones que contiene una correlación de entidad no se replica entre dominios.

- **No se admiten las correlaciones de matriz de bytes**

  Un conjunto de correlaciones que contiene una correlación que está configurada con COPY_TO_BYTES no se replica entre dominios.

- **No se admite la grabación diferida**

  Un conjunto de correlaciones que contiene una correlación que está configurada con soporte de grabación diferida no se replica entre dominios.

Integración de base de datos: almacenamiento en memoria caché de grabación diferida, en línea y complementaria

WebSphere eXtreme Scale se utiliza para atender una base de datos tradicional y eliminar la actividad de lectura que normalmente se envía a la base de datos. Puede utilizarse una memoria caché coherente con una aplicación mediante el uso directo o indirecto de un correlacionador de objetos relacionales. La memoria caché coherente puede después descargar de lecturas la base de datos o el programa de fondo. En un escenario ligeramente más complejo, como por ejemplo un acceso transaccional a un conjunto de datos donde sólo algunos de los datos necesitan garantías de persistencia tradicional, puede usarse el filtrado para descargar incluso transacciones de grabación.

Puede configurar eXtreme Scale para que funcione como un espacio de proceso de base de datos en memoria muy flexible. No obstante, eXtreme Scale no es un correlacionador de objetos relacionales (ORM). No sabe de dónde proceden los datos de eXtreme Scale. Una aplicación o un ORM puede colocar datos en un servidor eXtreme Scale. Es responsabilidad del origen de datos garantiza que son coherentes con la base de datos de la que proceden los datos. Esto significa que eXtreme Scale no puede invalidar los datos extraídos de una base de datos automáticamente. La aplicación o el correlacionador debe proporcionar esta función y gestionar los datos almacenados en eXtreme Scale.

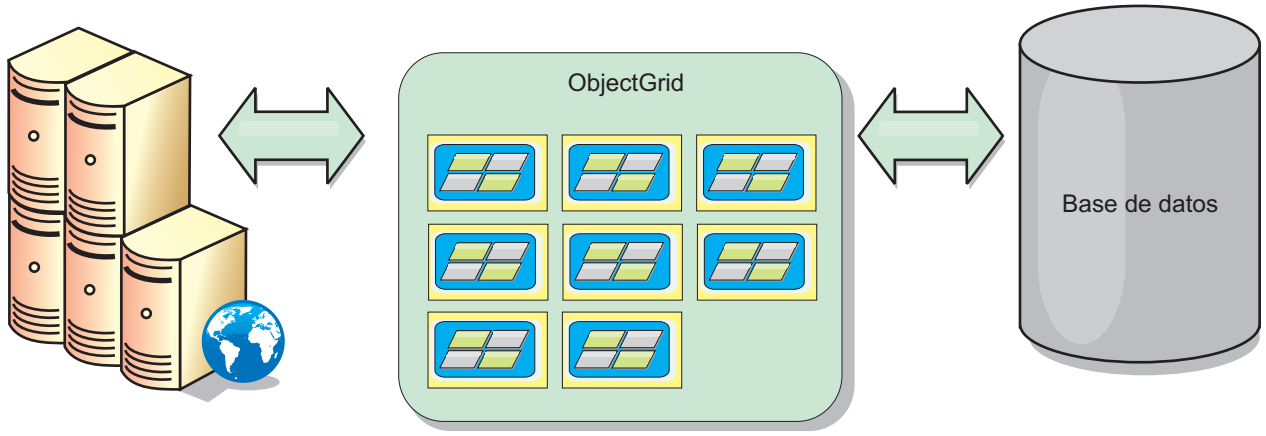


Figura 17. ObjectGrid como un almacenamiento intermedio de base de datos

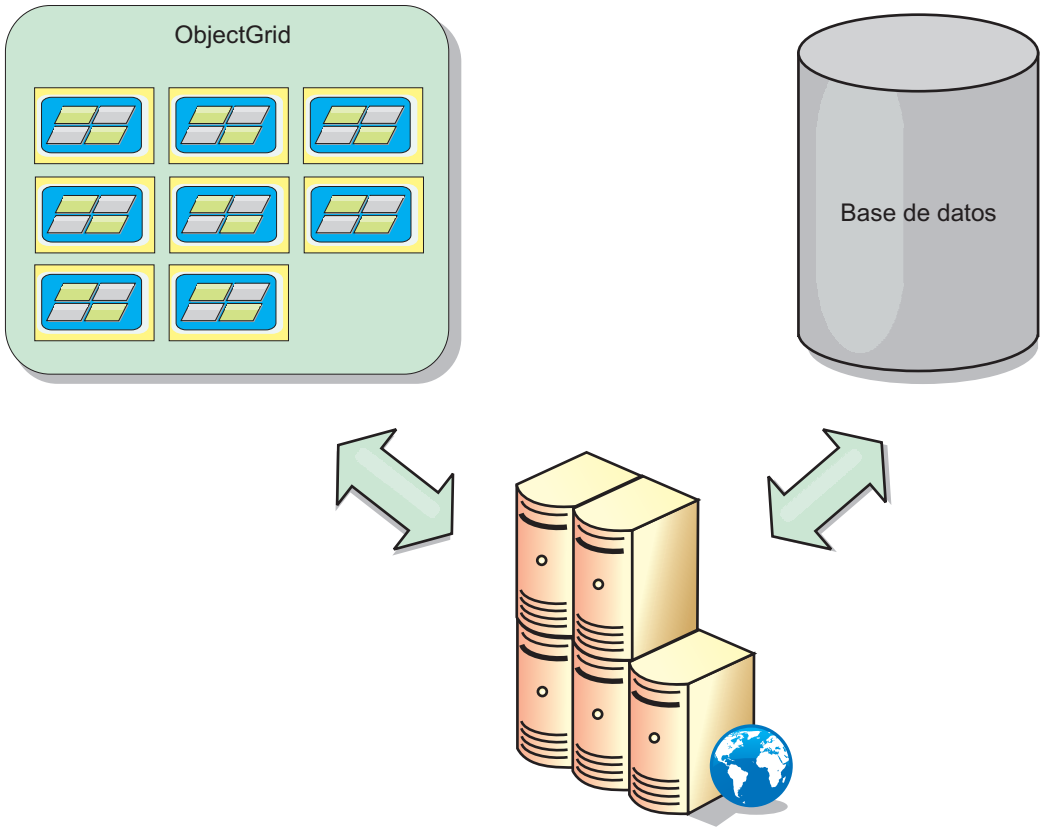


Figura 18. ObjectGrid como una memoria caché secundaria

Memoria caché escasa y completa

WebSphere eXtreme Scale puede utilizarse como una memoria caché escasa o una memoria caché completa. Una memoria caché escasa sólo mantiene un subconjunto de los datos totales, mientras que una memoria caché completa conserva todos los datos y se pueden llenar de forma poca activa y a petición. A las memorias caché escasas normalmente se accede utilizando claves (en lugar de índices o consultas) puesto que los datos sólo están parcialmente disponibles.

Cuando no existe ninguna clave (una falta de memoria caché), se invoca el siguiente nivel y los datos se captan e insertan en el nivel de memoria caché respectivo. Si utiliza una consulta o un índice, sólo se accede a los valores cargados actualmente y las solicitudes no se remiten a los demás niveles. Una memoria caché completa contiene todos los datos necesarios y se puede acceder a la misma utilizando atributos que no son de clave con índices o consultas.

Se precarga una memoria caché completa con los datos anteriores a las aplicaciones que la utilizan, y funciona de forma eficaz como sustituto de la base de datos. Una vez que se han cargado los datos, se puede tratar de forma similar a una base de datos. Puesto que están disponibles todos los datos, las consultas y los índices se pueden utilizar para encontrar y agregar datos.

Memoria caché complementaria y memoria caché en línea

WebSphere eXtreme Scale se utiliza para proporcionar almacenamiento en memoria caché en línea para un programa de fondo de base de datos o como una memoria caché secundaria para una base de datos. El almacenamiento en memoria caché en línea utiliza eXtreme Scale como el medio principal para interactuar con los datos. Cuando eXtreme Scale se utiliza como memoria caché complementaria, el programa de fondo se utiliza junto con eXtreme Scale.

Memoria caché complementaria

eXtreme Scale puede utilizarse como una memoria caché complementaria para una capa de acceso de datos de una aplicación. En este escenario, eXtreme Scale se utiliza para almacenar temporalmente objetos que normalmente se recuperarían de una base de datos de programa de fondo. Las aplicaciones comprueban si eXtreme Scale contiene los datos deseados. Si es así, los datos se devuelven al llamante. Si no contiene los datos, éstos se recuperan del programa de fondo y se insertan en eXtreme Scale de modo que la próxima solicitud pueda utilizar la copia almacenada en la memoria caché. El diagrama siguiente ilustra cómo eXtreme Scale puede usarse como memoria caché complementaria mediante el uso de una capa de acceso de datos arbitrarios como OpenJPA o Hibernate.

Plug-ins de memoria caché para Hibernate y OpenJPA

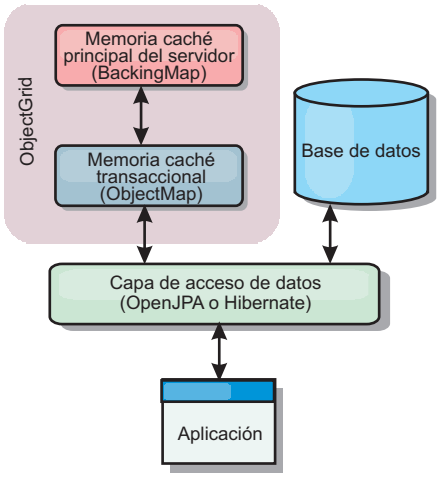


Figura 19. Memoria caché complementaria

Los plug-ins de memoria caché para OpenJPA e Hibernate se incluyen en eXtreme Scale , que le permite utilizar eXtreme Scale como una memoria caché secundaria

automática. El uso de eXtreme Scale como un proveedor de memoria caché aumenta el rendimiento cuando se leen y consultan datos y reduce la carga de la base de datos. Existen ventajas que eXtreme Scale tiene sobre las implementaciones de memoria caché incorporadas, porque la memoria caché se duplica automáticamente entre todos los procesos. Cuando un cliente almacena un valor en la memoria caché, todos los otros clientes podrán utilizar dicho valor.

Memoria caché en línea

Cuando eXtreme Scale se utiliza como memoria caché en línea, interactúa con el programa de fondo mediante un plug-in de cargador. Este escenario puede simplificar el acceso a los datos al permitir que las aplicaciones accedan directamente a las API de eXtreme Scale. Están soportados varios escenarios de almacenamiento en memoria caché en eXtreme Scale para garantizar que los datos de la memoria caché y los datos del programa de fondo están sincronizados. El diagrama siguiente ilustra cómo una memoria caché en línea interactúa con la aplicación y el programa de fondo.

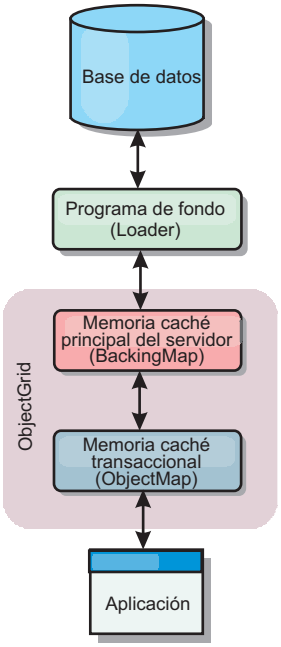


Figura 20. Memoria caché en línea

Almacenamiento en memoria caché en línea

El almacenamiento en memoria caché en línea utiliza eXtreme Scale como el medio principal para interactuar con los datos. Cuando se utiliza eXtreme Scale como una memoria caché en línea, la aplicación interactúa con el programa de fondo mediante un plug-in Loader.

La opción de memoria caché en línea simplifica el acceso de datos, porque permite a las aplicaciones acceder a las API de eXtreme Scale directamente. WebSphere eXtreme Scale soporta varios escenarios de memoria caché en línea, del modo siguiente.

- Lectura directa
- Grabación directa
- Grabación diferida

Caso de ejemplo de almacenamiento en memoria caché de lectura directa

Una memoria caché de lectura directa es una memoria caché escasa que carga de forma poco activa entradas de datos por clave cuando se solicitan. Esto se lleva a cabo sin que el solicitante sepa cómo se llenan las entradas. Si los datos no se pueden encontrar en la memoria caché de eXtreme Scale, eXtreme Scale recuperará los datos que faltan del plug-in Loader, que carga los datos de la base de datos de programa de fondo y los inserta en la memoria caché. Las solicitudes subsiguientes para la misma clave de datos se encontrarán en la memoria caché hasta que se elimina, anula o desaloja.

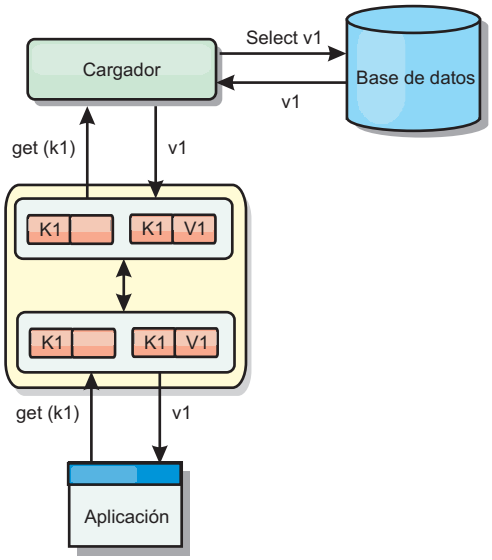


Figura 21. Almacenamiento en memoria caché de lectura directa

Caso de ejemplo de almacenamiento en memoria caché de grabación directa

En una memoria caché de grabación directa, cada grabación en la memoria caché graba de forma síncrona en la base de datos mediante el cargador. Este método proporciona coherencia con el programa de fondo, pero reduce el rendimiento de grabación porque la operación de la base de datos es síncrona. Como que la memoria caché y la base de datos están actualizadas, las lecturas subsiguientes para los mismos datos se encontrarán en la memoria caché, evitando la llamada a la base de datos. Una memoria caché de grabación directa suele utilizarse junto con una memoria caché de lectura directa.

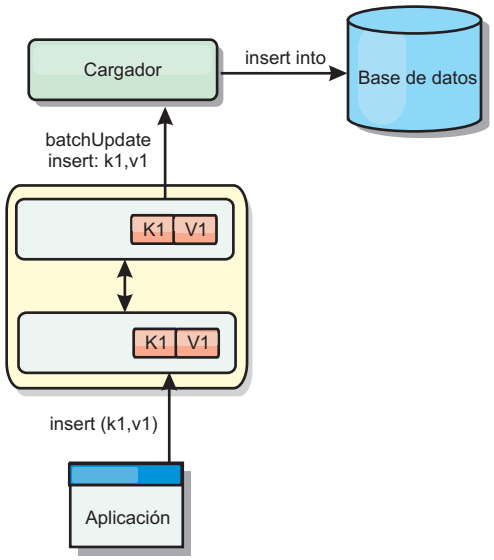


Figura 22. Almacenamiento en memoria caché de grabación directa

Caso de ejemplo de almacenamiento en memoria caché de grabación anticipada

La sincronización de base de datos se puede mejorar grabando los cambios de forma asíncrona. Esto se conoce como memoria caché de grabación diferida o de grabación aplazada. En su lugar, los cambios que normalmente se grabarían de forma síncrona en el cargador se colocarán en el almacenamiento intermedio de eXtreme Scale y se grabarán en la base de datos utilizando una hebra de subordinada. El rendimiento de grabación se mejora de forma significativa porque la operación de la base de datos se elimina de la transacción del cliente y se pueden comprimir las grabaciones de la base de datos. Si desea más información, consulte “Almacenamiento en memoria caché de grabación anticipada” en la página 39.

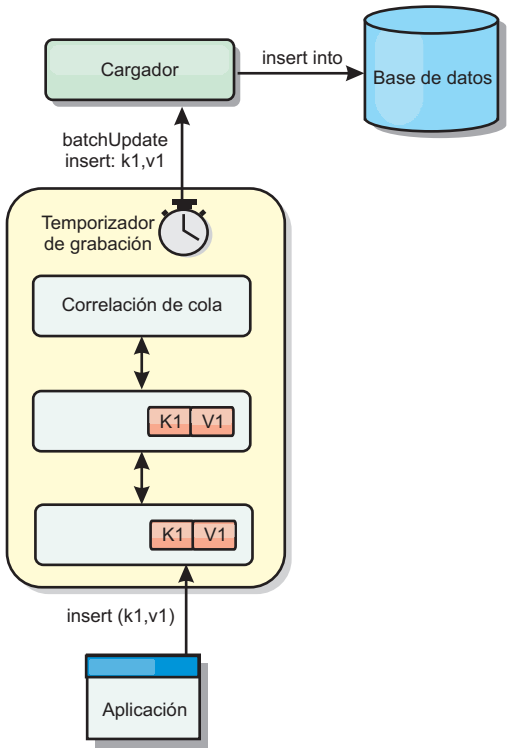


Figura 23. Almacenamiento en memoria caché de grabación anticipada

Consulte "Almacenamiento en memoria caché de grabación anticipada" si desea información adicional.

Almacenamiento en memoria caché de grabación anticipada

Puede utilizar el almacenamiento en la memoria caché de grabación diferida para reducir la sobrecarga que se produce al actualizar una base de datos utilizada como programa de fondo.

Introducción

El almacenamiento en memoria caché de grabación diferida pone en cola de forma asíncrona actualizaciones del plug-in de cargador (Loader). Puede mejorar el rendimiento mediante la desconexión de actualizaciones, inserciones y eliminaciones de una correlación, la sobrecarga de la actualización de la base de datos de programa de fondo. La actualización asíncrona se realiza después de un retardo basado en la hora (por ejemplo, cinco minutos) o un retardo basado en entradas (1000 entradas).

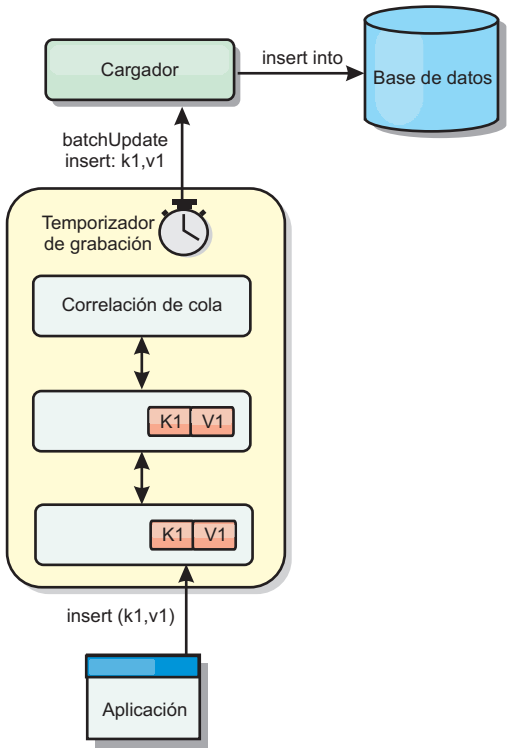


Figura 24. Almacenamiento en memoria caché de grabación anticipada

La configuración de la grabación diferida en BackingMap crea una hebra entre el cargador y la correlación. El cargador delega las solicitudes de datos a través de la hebra de acuerdo con los valores de configuración del método BackingMap.setWriteBehind. Cuando una transacción de eXtreme Scale inserta, actualiza o elimina una entrada de una correlación, se crea un objeto LogElement para cada uno de estos registros. Estos elementos se envían al cargador de grabación diferida y se ponen en cola en un objeto ObjectMap especial llamado correlación de cola. Cada correlación de respaldo con el valor de grabación diferida habilitado tiene sus propias correlaciones de cola. Una hebra de grabación diferida elimina periódicamente los datos en cola de las correlaciones de cola y los envía al cargador de programa de fondo real.

El cargador de grabación diferida sólo envía los tipos de inserción, actualización y eliminación de objetos LogElement al cargador real. Todos los demás tipos de objetos LogElement, por ejemplo el tipo EVICT, se pasan por alto.

Ventajas

La habilitación del soporte de grabación diferida tiene las ventajas siguientes:

- **Aislamiento de anomalía de programa de fondo:** el almacenamiento de grabación diferida proporciona una capa de aislamiento de las anomalías de programa de fondo. Cuando la base de datos de programa de fondo falla, las actualizaciones se ponen en cola en la correlación de cola. Las aplicaciones pueden continuar con las transacciones a eXtreme Scale. Cuando se recupera el programa de fondo, los datos de la correlación de cola se envían al programa de fondo.
- **Carga reducida de programa de fondo** el cargador de grabación diferida fusiona las actualizaciones según una clave, de forma que sólo existe una actualización

fusionada por clave en la correlación de cola. Este procedimiento reduce el número de actualizaciones en la base de datos de programa de fondo.

- **Rendimiento mejorado de transacciones:** los tiempos individuales de las transacciones de eXtreme Scale se reducen porque la transacción no necesita esperar a que los datos se sincronicen con el programa de fondo.

Consideraciones sobre el diseño de aplicaciones

Habilitar el soporte de grabación diferida es sencillo, pero diseñar una aplicación que funcione con el soporte de grabación diferida requiere un cuidado especial. Sin el soporte de grabación diferida, la transacción ObjectGrid encierra la transacción del programa de fondo. La transacción ObjectGrid se inicia antes de que se inicie la transacción de programa de fondo, pero termina después de que termine la transacción de programa de fondo.

Con el soporte de grabación diferida habilitado, la transacción ObjectGrid finaliza antes de que se inicie la transacción de programa de fondo. La transacción ObjectGrid y la transacción del programa de fondo se desacoplan.

Restricciones de la integridad referencial

Cada correlación de respaldo que se configura con soporte de grabación diferida tiene su propia hebra de grabación diferida que envía los datos al programa de fondo. Por lo tanto, los datos que se actualizan en correlaciones diferentes de una transacción ObjectGrid se actualizan en el programa de fondo en diferentes transacciones de programa de fondo. Por ejemplo, la transacción T1 actualiza la clave key1 en la correlación Map1 y la clave key2 en la correlación Map2. La actualización de key1 en la correlación Map1 se actualiza en el programa de fondo en una transacción de programa de fondo, y la clave key2 actualizada en la correlación Map2 se actualiza en el programa de fondo en otra transacción de programa de fondo mediante distintas hebras de grabación diferida. Si los datos almacenados en Map1 y Map2 tienen relaciones, como restricciones de clave foránea en el programa de fondo, puede que se produzca un error en las actualizaciones.

Al diseñar las restricciones de la integridad referencial en la base de datos de programa de fondo, asegúrese de que se permiten las actualizaciones que no funcionan.

Comportamiento de bloqueo de correlaciones de cola

Otra diferencia principal en el comportamiento de las transacciones es el comportamiento de bloqueo. ObjectGrid admite tres estrategias de bloqueo distintas: pesimista (PESSIMISTIC), optimista (OPTIMISTIC) y ninguno (NONE). Las correlaciones de cola de grabación diferida utilizan la estrategia de bloqueo pesimista independientemente de la estrategia de bloqueo configurada en el mapa de respaldo. Existen dos tipos diferentes de operaciones que adquieren un bloqueo en la correlación de cola:

- Cuando se confirma una transacción ObjectGrid, o se produce un vaciado (vaciado de correlación o vaciado de sesión), la transacción lee la clave de la correlación de cola y coloca un bloqueo S en la clave.
- Cuando se confirma una transacción ObjectGrid, la transacción intenta actualizar el bloqueo S a un bloqueo X en la clave.

Debido a este comportamiento de correlación de cola adicional, puede observar algunas diferencias de comportamiento en el bloqueo.

- Si la correlación de usuarios está configurada con la estrategia de bloqueo PESSIMISTIC, apenas existe diferencia en el comportamiento del bloqueo. Cada vez que se llama a una operación de vaciado o confirmación, se coloca un bloqueo S en la misma clave en la correlación de cola. Durante el tiempo de confirmación, no sólo se adquiere un bloqueo X para la clave de la correlación de usuarios, sino que se adquiere para la clave de la correlación de cola.
- Si la correlación de usuarios se configura con la estrategia de bloqueo OPTIMISTIC o NONE, la transacción de usuarios seguirá el patrón de la estrategia de bloqueo PESSIMISTIC. Cada vez que se llama a una operación de vaciado o confirmación, se adquiere un bloqueo S para la misma clave de la correlación de cola. Durante el tiempo de la confirmación, se adquiere un bloqueo X para la clave de la correlación de cola mediante la misma transacción.

Reintentos de la transacción de cargador

ObjectGrid no admite transacciones XA o en dos fases. La hebra de grabación diferida elimina los registros de la correlación de cola y actualiza los registros del programa de fondo. Si se produce una anomalía en el servidor durante la transacción, puede que se pierdan algunas actualizaciones del programa de fondo.

El cargador de grabación diferida reintentará automáticamente la grabación de las transacciones con anomalías y enviará un objeto LogSequence en duda al programa de fondo para evitar la pérdida de datos. Esta acción requiere que el cargador sea idempotente, que significa que cuando Loader.batchUpdate(TxId, LogSequence) se llama dos veces con el mismo valor, el resultado es como si se aplicara sólo una vez. Las implementaciones de cargador deben implementar la interfaz RetryableLoader para habilitar esta característica. Consulte la documentación de la API para obtener información detallada.

Anomalías del cargador

El plug-in de cargador puede fallar cuando no puede comunicarse con el programa de fondo de la base de datos. Esto puede suceder si el servidor de bases de datos o la conexión de red está inactivo. El cargador de grabación diferida pondrá en cola las actualizaciones e intentará enviar los cambios de los datos al cargador de forma periódica. El cargador debe notificar al tiempo de ejecución de ObjectGrid que hay un problema de conectividad de base de datos; para ello, emitirá una excepción LoaderNotAvailableException.

Por lo tanto, la implementación del cargador debe distinguir entre una anomalía de datos o un anomalía física del cargador. La anomalía de datos debe emitirse o volver a emitirse como excepción LoaderException o OptimisticCollisionException, pero una anomalía física del cargador debe emitirse o volver a emitirse como excepción LoaderNotAvailableException. ObjectGrid maneja estas dos excepciones de manera diferente:

- Si el cargador de grabación diferida obtiene una excepción LoaderException, el cargador de grabación diferida considerará la anomalía como un error de los datos, como por ejemplo un error de clave duplicada. El cargador de grabación diferida anulará el proceso por lotes de la actualización, e intentará actualizar un registro cada vez para aislar la anomalía de los datos. Si se vuelve a obtener una excepción {{LoaderException}} durante la actualización de un registro, se crea un registro de actualización con errores y se anota en la correlación de actualizaciones con errores.

- Si el cargador de grabación diferida obtiene una excepción LoaderNotAvailableException, el cargador de grabación diferida la considerará como un error porque no puede conectase a la base de datos, por ejemplo, el programa de fondo de la base de datos está inactivo, una conexión de base de datos no está disponible, o la red no está activa. El cargador de grabación diferida esperará 15 segundos y después volverá a intentar realizar la actualización de proceso por lotes en la base de datos.

El error habitual es emitir una excepción LoaderException cuando debería emitirse una excepción LoaderNotAvailableException. Todos los registros puestos en cola en el cargador de grabación diferida pasan a ser registros de actualizaciones con anomalías, que anula el propósito del aislamiento de anomalías de programa de fondo.

Consideraciones sobre el rendimiento

El soporte de almacenamiento en memoria caché de grabación diferida aumenta el tiempo de respuesta al eliminar la actualización del cargador de la transacción. Aumenta además el rendimiento de la base de datos ya que las actualizaciones de las bases de datos se combinan. Es importante comprender la sobrecarga que supone la hebra de grabación diferida, que extrae los datos de la correlación de cola y los envía al cargador.

El número máximo de actualizaciones o el tiempo máximo de actualización debe ajustarse en función del entorno y de los patrones de uso esperados. Si el valor del número máximo de actualizaciones o el tiempo máximo de actualización es demasiado pequeño, la sobrecarga de la hebra de grabación diferida puede sobrepasar las ventajas. Si se especifica un valor elevado para estos dos parámetros, podría aumentarse el uso de memoria al poner en cola los datos y aumentarse el tiempo obsoleto de los registros de la base de datos.

Para obtener un rendimiento óptimo, ajuste los parámetros de grabación diferida de acuerdo con los factores siguientes:

- Índice de transacciones de lectura y grabación.
- Misma frecuencia de actualización de registros.
- Latencia de actualización de la base de datos.

Cargadores

Con un plug-in Loader de eXtreme Scale, una correlación de eXtreme Scale se puede comportar como una memoria caché para los datos que, normalmente, se conservan en un almacén persistente en el mismo sistema o en otro. Una base de datos o un sistema de archivos suele utilizarse como el almacén persistente. Una máquina virtual Java (JVM) remota también se puede utilizar como el origen de datos, lo que permite crear memorias caché basadas en hub utilizando eXtreme Scale. Un cargador tiene la lógica para leer y escribir datos en un almacén persistente.

Visión general

Los cargadores son plug-ins de correlaciones de respaldo que se invocan cuando se realizan cambios en la correlación de respaldo o ésta no puede satisfacer una solicitud de datos (una falta de memoria caché). Se invoca el cargador cuando la memoria caché no puede satisfacer la solicitud de una clave, proporcionando la capacidad de lectura a través y el relleno poco activo de la memoria caché. Un cargador también permite actualizar la base de datos cuando los valores de la

memoria caché cambian. Todos los cambios dentro de una transacción se agrupan para permitir minimizar el número de interacciones de la base de datos. Se utiliza un plug-in TransactionCallback junto con el cargador para desencadenar la demarcación de la transacción de fondo. Utilizar este plug-in es importante cuando se incluyen varias correlaciones en una única transacción, o cuando se desechan los datos de una transacción en la memoria caché sin confirmar.

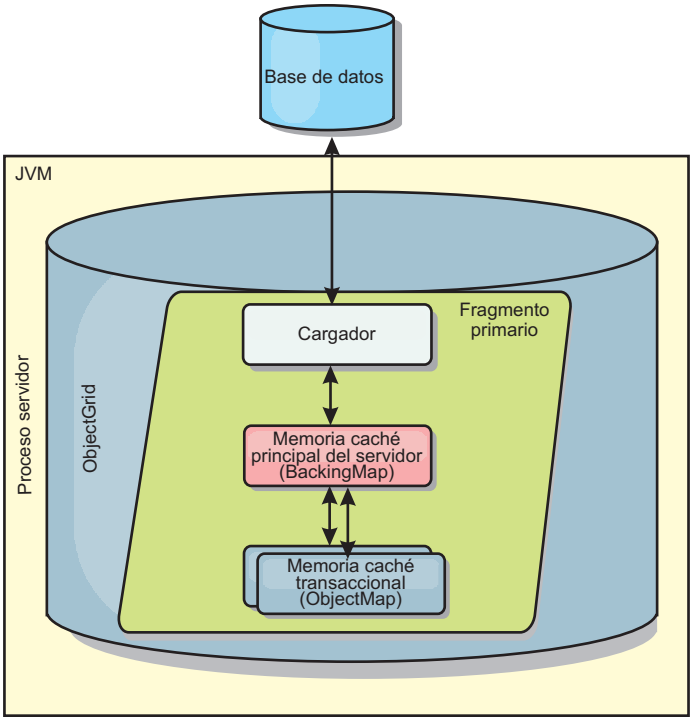


Figura 25. Cargador

El cargador también puede utilizar las actualizaciones sobrecualificadas para evitar mantener los bloqueos de base de datos. Al almacenar un atributo de versión en el valor de memoria caché, el cargador puede ver la imagen antes y después del valor tal como se actualiza en la memoria caché. Este valor se puede utilizar cuando se actualiza la base de datos o cuando se realiza un programa de fondo para verificar que los datos no se han actualizado. Asimismo, se puede configurar un cargador para precargar la cuadrícula cuando se inicia. Cuando se realizan particiones, se asocia una instancia de cargador con cada partición. Si la correlación "Company" tiene diez particiones, hay diez instancias de cargador, una por partición primaria. Cuando se activa el fragmento primario de la correlación, se invoca el método preloadMap para el cargador de forma síncrona o asíncrona, que permite cargar automáticamente la partición de la correlación con los datos procedentes del programa de fondo. Cuando se invoca de forma síncrona, se bloquean todas las transacciones de cliente, lo que impide un acceso incoherente a la cuadrícula. De forma alternativa, se puede utilizar un cargador previo de cliente para cargar toda la cuadrícula.

Dos cargadores incorporados pueden simplificar en gran medida la integración con los programas de fondo de la base de datos relacional. Los cargadores JPA utilizan las funciones de correlación de objetos relacionales (ORM) de ambas implementaciones, OpenJPA e Hibernate, de la especificación de JPA (Java

Persistence API). Consulte la información sobre los cargadores JPA en *Visión general del producto* si desea más información.

Uso de un cargador

Para añadir un cargador a la configuración de BackingMap, puede utilizar la configuración mediante programa o la configuración del archivo XML. Un cargador tiene la siguiente relación con una correlación de respaldo.

- Una correlación de respaldo sólo puede tener un cargador.
- Una correlación de respaldo de cliente (memoria caché cercana) no puede tener un cargador.
- Una definición de cargador se puede aplicar a varias correlaciones de respaldo, pero cada una de éstas tiene su propia instancia de cargador.

Si desea más información, consulte el tema sobre cómo escribir un cargador en *Visión general del producto*.

Conexión de un cargador mediante la configuración del XML

Un cargador proporcionado por la aplicación se puede conectar utilizando un archivo XML. El siguiente ejemplo demuestra cómo conectar el cargador "MyLoader" en la correlación de respaldo "map1". Debe especificar el className para el cargador, el nombre de la base de datos y los detalles de la conexión, y las propiedades del nivel de aislamiento. Puede utilizar la misma estructura XML si sólo utiliza un cargador previo especificando el nombre de clase del cargador previo, en lugar de un nombre de clase de cargador completo.

Configuración del cargador con 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>
```

Conexión de un cargador mediante programa

Sólo puede utilizar una configuración programática con cuadrículas locales en memoria. El fragmento de código siguiente muestra cómo conectar un cargador proporcionado por la aplicación en la correlación de respaldo map1 mediante ObjectGrid API:

Configuración programática de un cargador

```
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 );
```

Este fragmento de código presupone que la clase MyLoader es la clase proporcionada por la aplicación que implementa la interfaz com.ibm.websphere.objectgrid.plugins.Loader. Puesto que la asociación de un cargador con una correlación de respaldo no se puede modificar después de que se inicialice el ObjectGrid, el código se debe ejecutar antes de invocar el método initialize de la interfaz ObjectGrid que se está llamando. Se produce una excepción IllegalStateException en una llamada de método setLoader, si se llama después de que se haya producido la inicialización.

El cargador que proporciona la aplicación puede tener propiedades establecidas. En el ejemplo, el cargador MyLoader se utiliza para leer y escribir datos de una tabla en una base de datos relacional. El cargador debe especificar el nombre de la base de datos y el nivel de aislamiento de SQL. El cargador MyLoader tiene los métodos setDataBaseName y setIsolationLevel que permiten a la aplicación establecer estas dos propiedades de cargador.

- El usuario 'bob' está autenticado como usuario de eXtreme Scale. La aplicación accede a una cuadrícula denominada 'mygrid' utilizando el nombre de unidad de persistencia 'DB2Hibernate'. El servidor de contenedor es 'XS_Server1'. La información resultante debe ser tal como se indica a continuación:
  - **Usuario**=bob
  - **Nombre de estación de trabajo**=XS_Server1,192.168.1.101
  - **Nombre de aplicación**=mygrid,DB2Hibernate
  - **Información de contabilidad**=1, DEFAULT,FE7954BD-0126-4000-E000-2298094151DB,com.ibm.db2.jcc.t4.b@71787178
- El usuario 'bob' se autentica utilizando una señal de WAS. La aplicación accede a una cuadrícula denominada 'mygrid' utilizando el nombre de unidad de persistencia 'DB2OpenJPA'. El servidor de contenedor es 'XS_Server2'. La información resultante debe ser tal como se indica a continuación:
  - **Usuario**
    =acme.principal.UserPrincipal[Bob],acme.principal.GroupPrincipal[admin]
  - **Nombre de estación de trabajo**=XS_Server2,192.168.1.102
  - **Nombre de aplicación**=mygrid,DB2OpenJPA
  - **Información de contabilidad**=188,DEFAULT,FE72BC63-0126-4000-E000-851C092A4E33,com.ibm.ws.rsadapter.jdbc.WSJccSQLJConnection@2b432b43

Lea la información sobre DB2 Performance Expert para aprender a supervisar el acceso a la base de datos.

Precarga de datos y calentamiento

En muchos casos de ejemplo que incorporan el uso de un cargador, puede preparar su cuadrícula precargando datos en ella.

Cuando se utiliza como una memoria caché completa, la cuadrícula debe contener todos los datos y se debe cargar antes de que los clientes se puedan conectar a ella. Cuando se utiliza una memoria caché escasa, debe calentar la memoria caché con datos de forma que los clientes puedan tener un acceso inmediato a los datos cuando se conectan.

Existen dos enfoques para la precarga de datos en la cuadrícula: utilizar un plug-in Loader o un cargado de clientes, tal como se describe en las siguientes secciones.

Plug-in Loader

El plug-in Loader se asocia a cada correlación y es responsable de sincronizar un fragmento de partición primaria único con la base de datos. El método preloadMap del plug-in Loader se invoca automáticamente cuando se activa un fragmento. Por lo tanto, si tiene 100 particiones, existen 100 instancias de cargador, cada una de estas cargando los datos para su partición. Si se ejecuta de forma síncrona, todos los clientes se bloquean hasta que se complete la precarga.

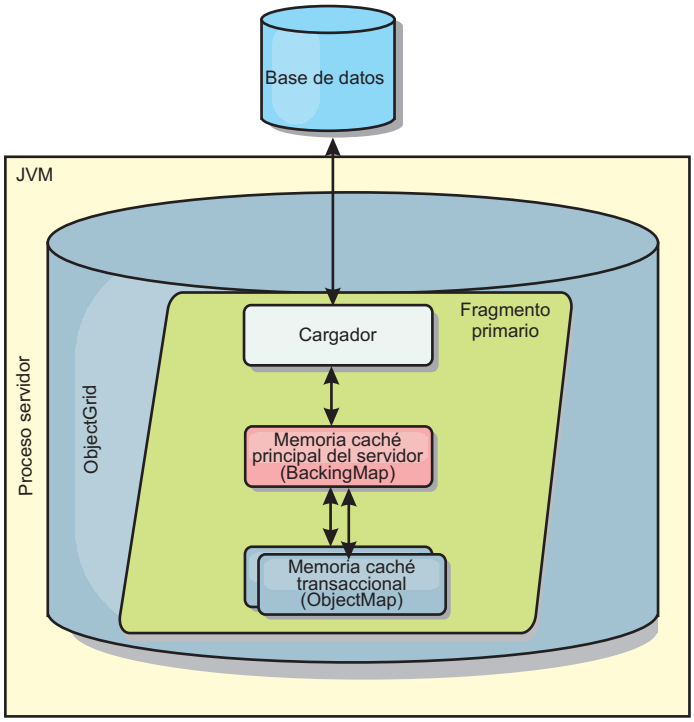


Figura 26. Plug-in Loader

Consulte los detalles sobre cómo utilizar un cargador en *Guía de programación* si desea más información.

Cargador de clientes

Un cargador de clientes es un patrón para utilizar uno o más clientes para carga la cuadrícula con datos. El uso de varios clientes para cargar los datos de cuadrícula puede ser eficaz cuando el esquema de partición no se almacena en la base de datos. Puede invocar los cargadores de clientes manualmente o automáticamente cuando se inicia la cuadrícula. De forma opcional, los cargadores de clientes pueden utilizar StateManager para establecer el estado de la cuadrícula en la modalidad de precarga, de forma que los clientes no pueden acceder a la cuadrícula mientras está precargando los datos. WebSphere eXtreme Scale incluye un cargador basado en JPA (Java Persistence API) que puede utilizar para cargar automáticamente la cuadrícula con los proveedores OpenJPA o Hibernate JPA.

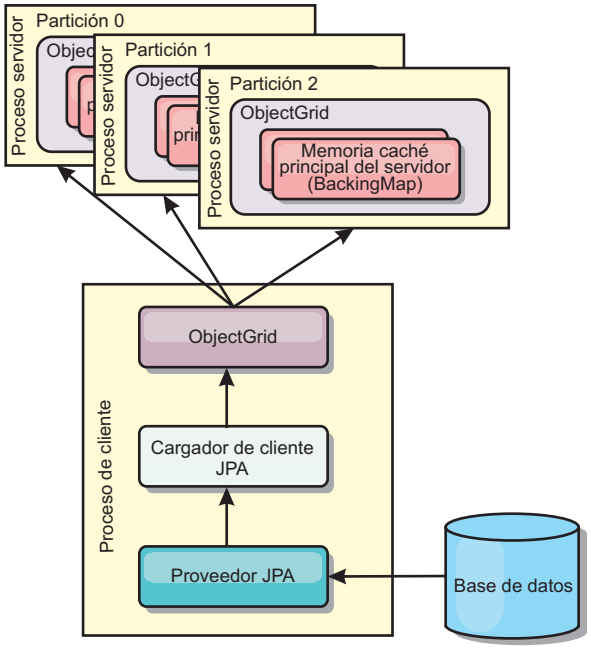


Figura 27. Cargador de clientes

Precarga de correlaciones

Las correlaciones se pueden asociar a cargadores. Un cargador se utiliza para captar objetos cuando no se pueden encontrar en la correlación (una falta de coincidencia) así como para grabar los cambios en un programa de fondo cuando se confirma una transacción. Los cargadores también se pueden utilizar para cargar previamente datos en una correlación. El método preloadMap de la interfaz del cargador se invoca en cada correlación cuando su correspondiente partición de MapSet pasa a ser un fragmento primario. El método preloadMap no se invoca en réplicas. Intenta cargar todos los datos referenciados previstos del programa de fondo en la correlación utilizando la sesión proporcionada. La correlación relevante la identifica el argumento BackingMap que se pasa al método preloadMap.

```
void preloadMap(Session session, BackingMap backingMap) throws LoaderException;
```

Precarga en el MapSet particionado

Las correlaciones puede particionarse en N particiones. Por lo tanto, las correlaciones pueden extenderse por varios servidores, con cada entrada identificada por una clave que sólo se almacena en uno de esos servidores. Las correlaciones muy grandes pueden mantenerse en un eXtreme Scale porque la aplicación ya no está limitada por el tamaño del almacenamiento dinámico de una sola JVM para mantener todas las entradas de una correlación. Las aplicaciones que desea cargar previamente con el método preloadMap de la interfaz del cargador deben identificar el subconjunto de datos que carga previamente. Siempre existe un número fijo de particiones. Puede determinar este número utilizando el siguiente ejemplo de código:

```
int numPartitions = backingMap.getPartitionManager().getNumOfPartitions();
int myPartition = backingMap.getPartitionId();
```

Este ejemplo de código muestra como una aplicación puede identificar un subconjunto de datos que se debe cargar previamente de la base de datos. Las

aplicaciones siempre deben utilizar estos métodos incluso cuando la correlación no está particionada inicialmente. Estos métodos permiten una cierta flexibilidad: si posteriormente los administradores particionan la correlación, el cargador sigue funcionando correctamente.

La aplicación debe emitir consultas para recuperar el subconjunto *myPartition* del programa de fondo. Si se utiliza una base de datos, puede ser más fácil tener una columna con un identificador de partición para un registro dado salvo que haya alguna consulta natural que permita a los datos de la tabla particionarse fácilmente.

Consulte los detalles sobre cómo escribir un cargador con un controlador de precarga de réplica en *Guía de programación* para ver un ejemplo sobre cómo implementar un cargador para un eXtreme Scale duplicado.

Rendimiento

La implementación de la precarga copia datos del programa de fondo en la correlación almacenando varios objetos en la correlación de una única transacción. El número óptimo de registros para almacenar por transacción depende de varios factores, incluidos la complejidad y el tamaño. Por ejemplo, después de que la transacción incluya bloques de más de 100 entradas, se reduce la ventaja del rendimiento a medida que aumenta el número de entradas. Para determinar el número óptimo, empiece con 100 entradas y, a continuación, aumente el número hasta que no se detecte más aumento en el rendimiento. Las transacciones de mayor tamaño dan como resultado un mayor rendimiento de duplicación. Recuerde que sólo el fragmento primario ejecuta el código de precarga. Los datos cargados previamente se duplican desde el fragmento primario hasta todas las réplicas que están en línea.

Precarga de MapSets

Si la aplicación utiliza un MapSet con varias correlaciones, cada correlación tendrá su propio cargador. Cada cargador tiene un método de carga previa. eXtreme Scale carga cada correlación en serie. Será más eficaz precargar todas las correlaciones designando una única correlación como la correlación de precarga. Este proceso es un convenio de aplicación. Por ejemplo, dos correlaciones, departamento y empleado, podrían utilizar el cargador de departamento para cargar previamente las correlaciones de departamento y de empleado. Este procedimiento asegura que, transaccionalmente, si una aplicación desea un departamento los empleados de dicho departamento están en la memoria caché. Cuando el cargador de departamento precarga un departamento desde el programa de fondo, también capta los empleados de dicho departamento. El objeto de departamento y sus objetos de empleados asociados se añadirán a la correlación utilizando una sola transacción.

Precarga recuperable

Algunos clientes tienen conjuntos de datos de gran tamaño que necesitan almacenarse en la memoria caché. La precarga de estos datos puede requerir mucho tiempo. A veces, la precarga debe finalizar para que la aplicación pueda ir en línea. Puede sacar provecho de que la precarga sea recuperable. Suponga que hay un millón de registros que se deben precargar. El fragmento primario los precarga y falla al llegar al registro número 800.000. Normalmente, la réplica elegida como el nuevo fragmento primario borra los estados duplicados y empieza desde el principio. eXtreme Scale puede emplear una interfaz

ReplicaPreloadController. El cargador de la aplicación también necesitará implementar la interfaz ReplicaPreloadController. Este ejemplo añade un solo método al cargador: `Status checkPreloadStatus(Session session, BackingMap bmap);`. Este método lo invoca el tiempo de ejecución de eXtreme Scale antes de que se llame al método de carga previa de la interfaz del cargador. eXtreme Scale comprueba el resultado de este método (estado) para determinar su comportamiento siempre que una réplica pasa a ser un fragmento primario.

Tabla 4. Valor de estado y respuesta

Valor de estado devuelto	Respuesta de eXtreme Scale
Status.PRELOADED_ALREADY	eXtreme Scale no llama al método de precarga porque su valor de estado indica que la correlación se ha precargado completamente.
Status.FULL_PRELOAD_NEEDED	eXtreme Scale borra la correlación y llama de forma normal al método de precarga.
Status.PARTIAL_PRELOAD_NEEDED	eXtreme Scale deja la correlación tal cual y llama a la precarga. Esta estrategia permite al cargador de aplicación seguir realizando la precarga a partir de ese momento.

Evidentemente, cuando un fragmento primario está cargando la correlación, debe dejar algún estado en una correlación del MapSet que se está duplicando para que la réplica determine qué estado debe devolver. Puede utilizar una correlación adicional llamada, por ejemplo, RecoveryMap. Esta RecoveryMap debe formar parte del mismo MapSet que se está precargando para asegurarse de que la correlación se duplica coherentemente con los datos que se están precargando. A continuación se muestra una implementación sugerida.

Cuando la precarga confirma cada bloque de registros, el proceso también actualiza un contador o valor en la RecoveryMap como parte de esa transacción. Los datos precargados y los datos de RecoveryMap se duplican de forma atómica en las réplicas. Cuando la réplica se promociona a fragmento primario, puede comprobar la RecoveryMap para ver qué ha pasado.

RecoveryMap puede mantener una sola entrada con la clave de estado. Si no existe ningún objeto para esta clave, es necesario una precarga completa (checkPreloadStatus devuelve FULL_PRELOAD_NEEDED). Si existe un objeto para esta clave de estado y el valor es COMPLETE, la precarga se completa y el método checkPreloadStatus devuelve PRELOADED_ALREADY. Si no, el objeto de valor indica donde se reinicia la precarga y el método checkPreloadStatus devuelve PARTIAL_PRELOAD_NEEDED. El cargador puede almacenar el punto de recuperación en una variable de instancia para el cargador de forma que, cuando se invoque la precarga, el cargador sepa el punto de partida. RecoveryMap también puede mantener una entrada por correlación si cada correlación se precarga independientemente.

Manejo de la recuperación en modalidad de duplicación síncrona con un cargador

El tiempo de ejecución de eXtreme Scale se ha diseñado para que no pierda datos confirmados cuando el fragmento primario falla. En la siguiente sección se muestran los algoritmos utilizados. Estos algoritmos sólo se aplican cuando un grupo de réplicas utiliza la réplica síncrona. Un cargador es opcional.

El tiempo de ejecución de eXtreme Scale puede configurarse de modo que duplique de forma síncrona todos los cambios de un fragmento primario en las réplicas. Cuando se coloca una réplica síncrona, recibe una copia de los datos

existentes en el fragmento primario. Durante este tiempo, el primario continúa recibiendo transacciones y las copia en la réplica de forma asíncrona. La réplica no se considera en línea en este momento.

Después de que la réplica capte el primario, la réplica entra en la modalidad de igual y se inicia la réplica síncrona. Cada transacción confirmada en el primario se envía a las réplicas síncronas y el primario espera una respuesta de cada réplica. Una secuencia de confirmación síncrona con un cargador en el primario se parece al siguiente conjunto de pasos:

Tabla 5. Secuencia de confirmación del fragmento primario

Paso con cargador	**Paso sin cargador**
Obtener bloqueos para entradas	igual
Desechar cambios para el cargador	no operativo
Guardar cambios en la memoria caché	igual
Enviar cambios a réplicas y esperar el reconocimiento	igual
Confirmar en el cargador a través del plug-in TransactionCallback	Se invoca el plug-in para enviar, pero no sucede nada
Liberar bloqueos para entradas	igual

Tenga en cuenta que los cambios se envían a la réplica antes de que se confirmen en el cargador. Para determinar cuando se confirman los cambios en la réplica, revise esta sentencia: en el momento de la inicialización, inicializar las listas tx en el fragmento primario tal como se indica a continuación.

```
CommitedTx = {}, RolledBackTx = {}
```

Durante el proceso de confirmaciónsíncrono, utilice la siguiente secuencia:

Tabla 6. Proceso de confirmación síncrona

Paso con cargador	**Paso sin cargador**
Obtener bloqueos para entradas	igual
Desechar cambios para el cargador	no operativo
Guardar cambios en la memoria caché	igual
Enviar cambios con una transacción confirmada, retrotraer transacción a la réplica y esperar al reconocimiento	igual
Borrar lista de transacciones confirmadas y transacciones retrotraídas	igual
Confirmar en el cargador a través del plug-in TransactionCallback	Se sigue llamando a la confirmación del plug-in TransactionCallBack, pero normalmente no sucede nada
Si la confirmación es satisfactoria, añada la transacción a las transacciones confirmadas, de lo contrario, añádala a las transacciones retrotraídas	no operativo
Liberar bloqueos para entradas	igual

Para el proceso de réplicas, utilice la siguiente secuencia:

1. Recibir cambios

2. Confirmar todas las transacciones recibidas en la lista de transacciones confirmadas
3. Retrotraer todas las transacciones recibidas en la lista de transacciones retrotraídas
4. Iniciar una transacción o sesión
5. Aplicar cambios en la transacción o sesión
6. Guardar la transacción o sesión en la lista de pendientes
7. Devolver respuesta

Tenga en cuenta que en la réplica, no se produce ninguna interacción de cargador mientras la réplica está en modalidad de réplica. El fragmentoprimario debe pasar todos los cambios a través del cargador. La réplica no realiza ningún cambio. Un efecto secundario de este algoritmo es que la réplica siempre tiene las transacciones, pero éstas no se confirman hasta que la siguiente transacción primaria envía el estado de confirmado de estas transacciones. A continuación, las transacciones se confirman o retrotraen en la réplica. Hasta entonces, las transacciones no están confirmadas. Puede añadir un temporizador en el fragmento primario que envía el resultado de la transacción después de un breve periodo de tiempo (unos pocos minutos). Este temporizador limita, pero no elimina, cualquier obsolescencia a este periodo de tiempo. Esta obsolescencia sólo es un problema si se utiliza la modalidad de lectura de réplica. Si no, la obsolescencia no tiene ningún impacto en la aplicación.

Cuando el fragmento primario falla, es probable que haya unas pocas transacciones confirmadas o retrotraídas en el fragmento primario, pero el mensaje nunca llega a la réplica con estos resultados. Cuando una réplica se promociona y pasa a ser el nuevo fragmento primario, una de las primeras acciones es manejar esta condición. Cada transacción pendiente se vuelve a procesar respecto al conjunto de correlaciones del nuevo fragmento primario. Si hay un cargador, cada transacción se ofrece al cargador. Estas transacciones se aplican estrictamente en el orden primero en entrar, primero en salir (FIFO). Si una transacción falla, ésta se ignora. Si hay tres transacciones pendientes, A, B y C, A podría confirmarse, B podría retrotraerse y C podría también confirmarse Ninguna transacción tiene ningún impacto en las demás. Suponga que son independientes.

Un cargador puede que desee utilizar una lógica un poco distinta cuando está en modalidad de recuperación de migración tras error comparada con la modalidad normal. El cargador puede saber fácilmente cuando está en modalidad de recuperación de migración tras error implementando la interfaz ReplicaPreloadController. El método checkPreloadStatus sólo se invoca cuando se completa la recuperación de la migración tras error. Por lo tanto, si el método de aplicación de la interfaz del cargador se invoca antes del método checkPreloadStatus, se trata de una transacción de recuperación. Después de llamar al método checkPreloadStatus, la recuperación de migración tras error está completa.

Técnicas de sincronización de base de datos

Cuando se utiliza WebSphere eXtreme Scale como memoria caché, se deben escribir aplicaciones que admitan datos obsoletos si la base de datos puede actualizarse de forma independiente a una transacción de eXtreme Scale. Para servir como un espacio de proceso de base de datos en memoria sincronizado, eXtreme Scale proporciona distintos métodos para mantener la memoria caché actualizada.

Técnicas de sincronización de base de datos

Renovación periódica

La memoria caché se puede invalidar o actualizar de forma automática y periódica utilizando el actualizados de base de datos basado en el tiempo de JPA (Java Persistence API). El actualizador consulta periódicamente la base de datos utilizando un proveedor JPA para cualquier actualización o inserción que se haya producido desde la actualización anterior. Todos los cambios identificados se anulan o actualizan automáticamente cuando se utilizan con una memoria caché escasa. Si se utilizan con una memoria caché completa, las entradas se pueden descubrir e insertar en la memoria caché. Las entradas nunca se eliminan de la memoria caché.

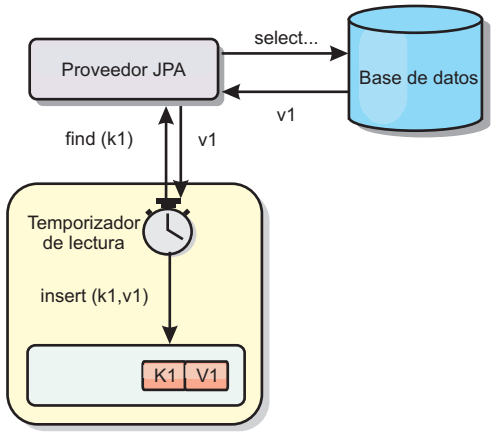


Figura 28. Renovación periódica

Desalojo

Las memorias caché escasas pueden utilizar políticas de desalojo para eliminar automáticamente datos de la memoria caché sin afectar a la base de datos. Existen tres políticas incorporadas incluidas en eXtreme Scale: tiempo de vida, menos usada recientemente y usada con menos frecuencia. Las tres políticas pueden, de forma opcional, desalojar datos de forma más agresiva a medida que la memoria pasa a estar limitada habilitando la opción de desalojo basado en memoria.

Anulación basada en sucesos

Las memorias caché escasas y completas se pueden invalidar o actualizar utilizando un generador de sucesos como, por ejemplo, JMS (Java Message Service). La anulación utilizando JMS puede unirse manualmente a cualquier proceso que actualiza el programa de fondo utilizando un desencadenante de base de datos. Se proporciona un plug-in JMS ObjectGridEventListener en eXtreme Scale que puede notificar a los clientes cuando la memoria caché del servidor tiene algún cambio. Esto puede disminuir la cantidad de tiempo que el cliente puede ver los datos obsoletos.

Anulación programática

Las API eXtreme Scale permiten la interacción manual de la memoria caché cercana y de servidor utilizando los métodos de API Session.beginNoWriteThrough(), ObjectMap.invalidate() y

EntityManager.invalidate(). Si un proceso de cliente o servidor ya no necesita una parte de los datos, los métodos de anulación se pueden utilizar para eliminar datos de la memoria caché cercana o del servidor. El método beginNoWriteThrough se aplica cualquier operación ObjectMap o EntityManager a la memoria caché local sin llamar al cargador. Si se invoca desde un cliente, la operación sólo se aplica a la memoria caché cercana (el cargador remoto no se invoca). Si se invoca en el servidor, la operación sólo se aplica a la memoria caché principal del servidor sin invocar el cargador.

Invalidación de datos de memoria caché obsoletos

Para reducir el intervalo de tiempo durante el cual los clientes pueden ver datos obsoletos, puede utilizar un mecanismo de invalidación basado en suceso o mediante programa.

Invalidación basada en sucesos

Las memorias caché escasas y completas se pueden invalidar o actualizar utilizando un generador de sucesos como, por ejemplo, JMS (Java Message Service). La anulación utilizando JMS puede unirse manualmente a cualquier proceso que actualiza el programa de fondo utilizando un desencadenante de base de datos. Se proporciona un plug-in JMS ObjectGridEventListener en eXtreme Scale que puede notificar a los clientes cuando la memoria caché del servidor cambia. Este tipo de notificación disminuye la cantidad de tiempo que el cliente puede ver los datos obsoletos.

La invalidación basada en sucesos consta normalmente de los tres componentes siguientes.

- **Cola de sucesos:** Una cola de sucesos almacena los sucesos de cambio de datos. Puede ser una cola JMS, una base de datos, una cola FIFO o cualquier clase de siempre que pueda gestionar los sucesos de cambio de datos.
- **Editor de sucesos:** Un editor de sucesos publica los sucesos de cambio de datos en la cola de sucesos. Un editor de sucesos es normalmente una aplicación que usted mismo crea o una implementación de plug-in de eXtreme Scale. El editor de sucesos sabe cuándo se cambian los datos o cambia los datos por sí mismo. Cuando se confirma una transacción, se generan los sucesos para los datos cambiados y el editor de sucesos publica estos sucesos en la cola de sucesos.
- **Consumidor de sucesos:** Un consumidor de sucesos consume sucesos de cambio de datos. El consumidor de sucesos es por lo general una aplicación para garantizar que los datos de la cuadrícula de destino se actualizan con el cambio más reciente de otras cuadrículas. Este consumidor de sucesos interactúa con la cola de sucesos para obtener los cambios de datos más recientes y aplica los cambios de datos en la cuadrícula de destino. Los consumidores de sucesos pueden utilizar las API de eXtreme Scale para invalidar datos obsoletos o actualizar la cuadrícula con los datos más recientes.

Por ejemplo, JMSObjectGridEventListener tiene una opción para un modelo cliente-servidor, en el cual la cola de sucesos es un destino de JMS designado. Todos los procesos del servidor son editores de sucesos. Cuando se confirma una transacción, el servidor obtiene los cambios de datos y los publica en la JMS de destino designada. Todos los procesos de cliente son consumidores de sucesos. Reciben los cambios de datos del destino de JMS designado y aplican los cambios en la memoria caché cercana del cliente.

Consulte el tema sobre la habilitación del mecanismo de invalidación del cliente en la *Guía de administración* si desea más información.

Anulación programática

Las API WebSphere eXtreme Scale permiten la interacción manual de la memoria caché cercana y de servidor utilizando los métodos de API Session.beginNoWriteThrough(), ObjectMap.invalidate() y EntityManager.invalidate(). Si un proceso de cliente o servidor ya no necesita una parte de los datos, los métodos de anulación se pueden utilizar para eliminar datos de la memoria caché cercana o del servidor. El método beginNoWriteThrough se aplica cualquier operación ObjectMap o EntityManager a la memoria caché local sin llamar al cargador. Si se invoca desde un cliente, la operación sólo se aplica a la memoria caché cercana (el cargador remoto no se invoca). Si se invoca en el servidor, la operación sólo se aplica a la memoria caché principal del servidor sin invocar el cargador.

Puede utilizar la anulación mediante programa con otras técnicas para determinar cuándo invalidar los datos. Por ejemplo, este método de invalidación utiliza mecanismos de invalidación basados en sucesos para recibir los sucesos de cambio de datos y luego utiliza interfaces de programación de aplicaciones para invalidar los datos obsoletos.

Índices

Utilice MapIndexPlugin para crear un índice o varios índices en una BackingMap con el acceso de datos que no son clave.

Tipos de índices y configuración

La característica de indexación está representada por MapIndexPlugin o Index para abreviar. Index es un plug-in BackingMap. Una BackingMap puede tener varios plug-ins Index configurados, siempre que cada uno de ellos siga las normas de configuración de Index.

Puede utilizar la característica de indexación para crear un índice o varios índices en una BackingMap. Un índice se crea a partir de un atributo o una lista de atributos de un objeto en la BackingMap. De esta manera, las aplicaciones pueden encontrar rápidamente determinados objetos. Con la característica de índices, las aplicaciones pueden encontrar objetos con un valor específico o dentro de un intervalo de valores de atributos indizados.

Existen dos tipos de índice: estático y dinámico. Con el índice estático, debe configurar el plug-in de índices en BackingMap antes de inicializar la instancia ObjectGrid. Puede realizar esta configuración con una configuración de XML o mediante programa de la BackingMap. Los índices estáticos empiezan a construir un índice durante la inicialización de ObjectGrid. El índice siempre está sincronizado con la BackingMap y listo para ser utilizado. Después de que se inicie el proceso de indexación estática, el mantenimiento del índice forma parte del proceso de gestión de transacciones de eXtreme Scale. Cuando las transacciones confirman cambios, estos cambios también actualizan el índice estático y los cambios de índice se retrotraen si la transacción se retrotrae.

Con el índice dinámico, puede crear un índice en una correlación BackingMap antes o después de la inicialización de la instancia ObjectGrid que contiene. Las aplicaciones tienen un control del ciclo de vida sobre el proceso de indexación dinámica, de forma que pueda eliminar un índice dinámico, cuando ya no sea necesario. Cuando una aplicación crea un índice dinámico, éste podría no estar listo para su uso inmediato debido al tiempo que tarda en completarse el proceso

de creación del índice. Puesto que la cantidad de tiempo depende del volumen datos indizados, se proporciona la interfaz DynamicIndexCallback para las aplicaciones que desean recibir notificaciones cuando se producen determinados sucesos de índices. Estos sucesos pueden incluir sucesos de error, destrucción y preparado. Las aplicaciones pueden implementar esta interfaz de devolución de llamada y registrarla con el proceso de índices dinámicos.

Si una BackingMap tiene un plug-in de índice configurado, podrá obtener el proxy de índice de aplicaciones de la ObjectMap correspondiente. Llamar al método getIndex en la ObjectMap y pasar el nombre del plug-in de índice devuelve el objeto de proxy de índice. Debe difundir el objeto de proxy de índice en una interfaz apropiada de índice de aplicaciones como, por ejemplo, MapIndex, MapRangeIndex, o una interfaz personalizada de índices. Después de obtener el objeto de proxy de índice, puede utilizar los métodos definidos en la interfaz de índices de aplicación para buscar objetos almacenados en memoria caché.

En la lista siguiente se resumen los pasos que debe seguir para utilizar los índices:

- Añada plug-ins de índices estáticos o dinámicos a BackingMap.
- Obtenga el objeto de proxy de índice de aplicación; para ello, emita el método getIndex de ObjectMap.
- Difunda el objeto de proxy de índice a una interfaz de índices de aplicación apropiada, como MapIndex, MapRangeIndex, o a una interfaz de índices personalizada.
- Utilice los métodos definidos en una interfaz de índices de aplicación para buscar los objetos almacenados en memoria caché.

La clase HashIndex es la implementación de plug-in de índice que puede soportar ambas interfaces de índice de aplicación incorporadas: MapIndex y MapRangeIndex. También puede crear sus propios índices. Puede añadir HashIndex como un índice estático o dinámico en BackingMap, obtener un objeto proxy de índice MapIndex o MapRangeIndex y utilizar el objeto proxy de índice para encontrar los objetos almacenados en memoria caché.

Para obtener información sobre cómo configurar HashIndex, consulte Configuración de HashIndex.

Si desea más información sobre cómo escribir su propio plug-in de índice, consulte la información sobre cómo escribir el plug-in de índice en *Guía de programación*..

Si desea información sobre cómo utilizar la indexación, consulte la información sobre cómo utilizar la indexación para el acceso de datos no clave en *Guía de programación* y Índice compuesto HashIndex.

Consideraciones sobre la calidad de los datos

Los resultados de los métodos de consulta de índice sólo representan una instantánea de los datos en un momento puntual. No se obtiene ningún bloqueo contra la entrada de datos después de que los resultados vuelvan a la aplicación. La aplicación tiene que ser consciente de que se pueden producir actualizaciones de datos en un conjunto de datos devuelto. Por ejemplo, la aplicación obtiene la clave de un objeto almacenado en memoria caché ejecutando el método findAll de MapIndex. Este objeto de clave devuelto se asocia a una entrada de datos de la memoria caché. La aplicación debe poder ejecutar el método get en ObjectMap para encontrar un objeto proporcionando el objeto de clave. Si otra transacción elimina el objeto de datos de la memoria caché, justo antes de que se llame al

método get, el resultado devuelto será nulo.

Consideraciones sobre el rendimiento de los índices

Uno de los principales objetivos de la característica de índices es mejorar el rendimiento global de BackingMap. Si los índices no se utilizan correctamente, podría verse afectado el rendimiento de la aplicación. Tenga en cuenta los siguientes factores antes de utilizar esta característica.

- **El número de transacciones de escritura simultáneas:** el proceso de índices se puede producir cada vez que una transacción escribe datos en una BackingMap. El rendimiento disminuye si hay muchas transacciones grabando datos en una correlación al mismo tiempo que una aplicación realiza operaciones de consulta de índices.
- **El tamaño del conjunto de resultados devuelto por una operación de consulta:** a medida que el tamaño del conjunto de resultados aumenta, el rendimiento de la consulta disminuye. El rendimiento tiene tendencia a disminuir si el tamaño del conjunto de resultados es un 15% o más de la BackingMap.
- **El número de índices creados sobre la misma BackingMap:** cada índice consume recursos del sistema. A medida que el número de índices creados sobre la BackingMap aumenta, disminuye el rendimiento.

La función de indexación puede mejorar el rendimiento de BackingMap de forma drástica. Los casos ideales se producen cuando la BackingMap tiene operaciones básicamente de lectura, el conjunto de resultados de la consulta es un pequeño porcentaje de las entradas de BackingMap, y sólo se crean unos pocos índices sobre la BackingMap.

Conceptos de almacenamiento en memoria caché de objetos Java

Básicamente, WebSphere eXtreme Scale se utiliza como una cuadrícula de datos y como memoria caché para los objetos Java. Se pueden utilizar varias API para interactuar con la cuadrícula eXtreme Scale para acceder y almacenar estos objetos.

En este tema se describen algunas de las API comunes y algunos de los conceptos que debe conocer al elegir una API y una topología de despliegue. Consulte el tema “Arquitectura de memoria caché: correlaciones, contenedores, clientes y catálogos” en la página 11 si desea una descripción de los distintos servicios y tecnologías que proporciona eXtreme Scale.

El componente central de WebSphere eXtreme Scale es ObjectGrid. ObjectGrid es el espacio de nombres que almacena datos relacionados y contiene conjuntos de correlaciones de totales de control, cada uno de ellos contiene pares de clave-valor. Estas correlaciones se pueden agrupar y particionar y tienen una gran disponibilidad y capacidad de ampliación.

Puesto que la cuadrícula contiene objetos Java por naturaleza, hay algunas consideraciones importantes cuando se diseña una aplicación, para que la cuadrícula pueda almacenar y acceder a datos de forma eficaz. Entre los factores que pueden afectar la escalabilidad, el rendimiento y el uso de se incluye lo siguiente.

Consideraciones del cargador de clases y la classpath

Puesto que eXtreme Scale almacena los objetos Java en la memoria caché de forma predeterminada, se deben definir definiciones de clase en la vía de acceso de clases siempre que se acceda a los datos.

Específicamente, los procesos de cliente y contenedor de eXtreme Scale deben incluir las clases o los JAR en la classpath al iniciar el proceso. Al diseñar una aplicación para ser utilizada con eXtreme Scale, separe las lógicas empresariales de los objetos de datos persistentes.

Consulte Carga de clases en el centro de información de WebSphere Application Server Network Deployment, si desea más información.

Para consideraciones dentro de una definición de infraestructura de Spring, consulte la sección de paquetes sobre la integración con la infraestructura de Spring en *Guía de programación*.

Para ver los valores relacionados con el uso del agente de instrumentación de WebSphere eXtreme Scale, consulte el tema del agente de instrumentación en *Guía de programación*.

Gestión de las relaciones

Los lenguajes orientados al objeto como, por ejemplo, Java, las bases de datos relacionales soportan las relaciones o asociaciones. Las relaciones reducen la cantidad de almacenamiento a través del uso de las referencias de objeto o claves foráneas.

Si se utilizan las relaciones en una cuadrícula, los datos se deben organizar en un árbol limitado. Debe haber un tipo raíz en el árbol y todos los hijos deben estar asociados sólo a una raíz. Por ejemplo: El Departamento puede tener muchos Empleados y un Empleado puede tener muchos Proyectos. Pero un Proyecto no puede tener muchos Empleados que pertenezcan a distintos departamentos. Una vez definida una raíz, todos los accesos a dicho objeto raíz y a sus descendientes se gestionan a través de la raíz. WebSphere eXtreme Scale utiliza el código hash de la clave del objeto raíz para elegir una partición.

Por ejemplo: partition = (hashCode MOD numPartitions).

Cuando todos los datos de una relación se enlazan a una única instancia de objeto, todo el árbol se puede colocar en una única partición y se puede acceder a é de forma muy eficaz mediante una transacción. Si los datos abarcan varias relaciones, se deben implicar varias particiones que conllevan llamadas remotas adicionales, que pueden llevar a cuellos de botella de rendimiento.

Datos de referencia

Algunas relaciones incluyen datos de búsqueda o de referencia como, por ejemplo: CountryName. Se trata de un caso especial donde los datos deben existir en todas las particiones. Aquí, cualquier clave raíz puede acceder a los datos y se devolverá el mismo resultado. Los datos de referencia como estos sólo deben utilizarse en los casos en los que los datos son bastante estadísticos, puesto que actualizarlos puede ser caro ya que se deben actualizar en todas las particiones. La API DataGrid es una técnica común para mantener los datos actualizados.

Costes y ventajas de la normalización

La normalización de los datos que utilizan relaciones puede ayudar a reducir la cantidad de memoria utilizada por la cuadrícula porque la duplicación de datos disminuye. Sin embargo, en general, cuantos más datos relacionales se añaden, menos se ampliarán. Si los datos se agrupan de forma conjunta, será más caro

conservar las relaciones y mantener los tamaños gestionables. Puesto que los datos de particiones de cuadrícula se basan en la clave de la raíz del árbol, el tamaño del árbol no se toma en consideración. Por lo tanto, si tiene muchas relaciones para una instancia de árbol, la cuadrícula se desequilibra, lo que provoca que una partición tenga más datos que las demás.

Cuando se deshace la normalización de los datos o se aplanan, los datos que normalmente se compartirían entre dos objetos, en lugar de esto, se duplican y cada una de las tablas de puede particionar de forma independiente, lo que proporciona una cuadrícula mucho más equilibrada. Aunque así se aumenta la cantidad de memoria utilizada, permite a la aplicación ampliarse ya que se puede acceder a una única fila de datos que contiene todos los datos necesarios. Esto es ideal para las cuadrículas que se leen con frecuencia puesto que el mantenimiento de los datos pasa a ser más caro.

Si desea más información, consulte Clasificación de sistemas XTP y ampliación.

Gestión de relaciones utilizando las API de acceso de datos

La API ObjectMap es la API de acceso de datos más rápida, más flexible y granular y proporciona un enfoque transaccional basado en sesiones para el acceso a datos de la cuadrícula de correlaciones. La API ObjectMap permite a los clientes utilizar las operaciones CRUD (crear, leer, actualizar y suprimir) comunes para gestionar los pares de clave-valor en la cuadrícula distribuida.

Cuando se utiliza la API ObjectMap, las relaciones de objetos se deben expresar mediante la incorporación del a clave foránea para todas las relaciones en el objeto padre.

A continuación se muestra un ejemplo.

```
public class Department {
 Collection<String> employeeIds;
}
```

La API EntityManager simplifica la gestión de relaciones extrayendo los datos persistentes de los objetos, incluidas las claves foráneas. Cuando el objeto se recupera más adelante de la cuadrícula, el gráfico de relaciones se vuelve a crear, como en el siguiente ejemplo.

```
@Entity
public class Department {
 Collection<String> employees;
}
```

La API EntityManager es muy similar a otras tecnologías de persistencia de objeto Java como, por ejemplo, JPA e Hibernate, porque sincroniza un gráfico de instancias de objeto Java gestionadas con el almacén persistente. En este caso, el almacén persistente está en una cuadrícula eXtreme Scale, donde cada entidad se representa como una correlación y la correlación contiene los datos de entidad, en lugar de las instancias de objeto.

Consideraciones de claves de la memoria caché

WebSphere eXtreme Scale utiliza las correlaciones de totales de control para almacenar datos en la cuadrícula, donde se utiliza un objeto Java para la clave.

Directrices

Al elegir una clave, considere los siguientes requisitos:

- Las claves no cambian nunca. Si una parte de la clave se debe modificar, la entrada de la memoria caché se debe eliminar y volver a insertar.
- Las claves deben ser pequeñas. Puesto que las claves se utilizan en todas las operaciones de acceso de datos, es una buena idea mantener la clave lo suficientemente pequeña para que se pueda serializar de forma eficaz y utilice menos memoria.
- Implemente un buen código good y un algoritmo equals. Los métodos hashCode y equals(Object o) siempre se deben alterar temporalmente para cada objeto de clave.
- Guarde en la memoria caché el hashCode de clave. Si es posible, guarde en la memoria caché el código hash en la instancia del objeto de clave para acelerar los cálculos de hashCode(). Dado que la clave es inmutable, se debe poder guardar en la memoria caché el hashCode.
- Evitar la duplicación de la clave en el valor. Cuando se utilice la API ObjectMap, es conveniente almacenar la clave dentro del objeto de valor. Cuando esto se realiza, los datos de la clave se duplican en la memoria.

Rendimiento de serialización

WebSphere eXtreme Scale utiliza varios procesos Java para alojar datos. Estos procesos serializan los datos: es decir, convierten los datos (que tienen el formato de las instancias de objeto Java) en bytes y de nuevo en objetos, según sea necesario mover los datos entre los procesos de cliente y servidor. La ordenación de los datos es la operación más costosa y el desarrollador de aplicaciones debe ocuparse de ella al designar el esquema, configurar la cuadrícula e interactuar con las API de acceso de datos.

Las rutinas predeterminadas de serialización y copia de Java son relativamente lentas y pueden consumir entre un 60 y 70 por ciento del procesador en una configuración típica. Las siguientes secciones son opciones para mejorar el rendimiento de la serialización.

Escribir un ObjectTransformer para cada BackingMap

Se puede asociar ObjectTransformer a BackingMap. La aplicación puede tener una clase que implemente la interfaz ObjectTransformer y proporcione implementaciones para las operaciones siguientes:

- Copia de valores
- Serialización e inflado de claves en corrientes o desde éstas
- Serialización e inflado de valores en corrientes o desde éstas

La aplicación no necesita copiar claves porque éstas se consideran inmutables.

Si desea más información, consulte Plug-ins para serializar y copiar los objetos almacenados en memoria caché y Procedimientos recomendados para la interfaz ObjectTransformer.

Nota: ObjectTransformer sólo se invoca cuando ObjectGrid conoce los datos que se están transformando. Por ejemplo, cuando se utilizan agentes de la API DataGrid, los agentes además de los datos de la instancia del agente o los datos devueltos del agente deben optimizarse mediante técnicas de serialización personalizadas. ObjectTransformer no se invoca para agentes de la API DataGrid.

Uso de entidades

Cuando se utiliza la API EntityManager con entidades, ObjectGrid no almacena los objetos de entidad en los objetos BackingMap. La API EntityManager convierte el objeto de entidad en objetos Tuple. Consulte Si desea más información, consulte el tema sobre cómo utilizar un cargador con correlaciones de entidad y tuples en *Guía de programación*. Las correlaciones de entidad se asocian automáticamente con un objeto ObjectTransformer altamente optimizado. Siempre que se utiliza la API ObjectMap o EntityManager para interactuar con correlaciones de entidad, se invoca a la entidad ObjectTransformer.

Serialización personalizada

Hay algunos casos en los que deben modificarse los objetos para utilizar la serialización personalizada, como la implementación de la interfaz java.io.Externalizable o al implementar los métodos writeObject y readObject para las clases que implementan la interfaz java.io.Serializable. Las técnicas de serialización personalizada deben emplearse cuando se serializan los objetos mediante mecanismos que no sean los métodos de la API ObjectGrid o la API EntityManager.

Por ejemplo, cuando los objetos o las entidades se almacenan como datos de instancia en un agente de la API DataGrid o el agente devuelve objetos o entidades, dichos objetos no se transforman mediante ObjectTransformer. El agente utilizará automáticamente ObjectTransformer al utilizar la interfaz `EntityMixin`. Si desea obtener más información, consulte el tema Agentes DataGrid y correlaciones basadas en entidades

Matrices de bytes

Cuando se utilizan las API ObjectMap o DataGrid, los objetos de clave y valor se serializan siempre que el cliente interactúa con la cuadrícula y cuando se duplican los objetos. Para impedir la sobrecarga de la serialización, utilice las matrices de bytes, en lugar de los objetos Java. Las matrices de bytes son mucho más baratas para almacenar en memoria, porque el JDK tiene menos objetos para buscar durante la recogida de basura y se pueden aumentar sólo cuando sea necesario. Las matrices de bytes sólo se deben utilizar si no es necesario acceder a los objetos utilizando consultas o índices. Puesto que los datos se almacenan como bytes, sólo se puede acceder a los datos a través de su clave.

WebSphere eXtreme Scale puede almacenar automáticamente los datos como matrices de bytes utilizando la opción de configuración de correlación CopyMode.COPY_TO_BYTES, o el cliente los puede gestionar manualmente. Esta opción almacenará los datos de forma eficaz en la memoria y también puede inflar automáticamente los objetos dentro de la matriz de bytes para ser utilizados por la consulta y los índices a petición.

Consulte los procedimientos recomendados del método CopyMode en *Guía de programación* si desea más información.

Inserción de datos para husos horarios diferentes

Al insertar datos con los atributos calendar, java.util.Date y timestamp en un ObjectGrid, debe asegurarse de que estos atributos de fecha y hora se creen basándose en el mismo huso horario, sobre todo cuando se realiza el despliegue en diversos servidores en varios husos horarios. El uso de los mismos objetos de fecha

y hora basados en el mismo huso horario puede garantizar que la aplicación sea segura por lo que respecta al huso horario y los datos se puedan consultar mediante los predicados calendar, java.util.Date y timestamp.

Si no se especifica explícitamente un huso horario al crear objetos de fecha y hora, Java utilizará el huso horario local y puede hacer que haya valores de fecha y hora incoherentes en clientes y servidores.

Considere un ejemplo en un despliegue distribuido en el cual client1 está en el huso horario [GMT-0] y client2 está en [GMT-6] y ambos quieren crear un objeto java.util.Date con el valor '1999-12-31 06:00:00'. Entonces client1 creará el objeto java.util.Date con el valor '1999-12-31 06:00:00 [GMT-0]' y client2 creará el objeto java.util.Date con el valor '1999-12-31 06:00:00 [GMT-6]'. Los dos objetos java.util.Date no son iguales porque el huso horario es diferente. Un problema similar se produce al precargar datos en particiones que residen en servidores en husos horarios diferentes si se utiliza el huso horario local para crear objetos de fecha y hora.

Para evitar el problema descrito, la aplicación puede elegir un huso horario como [GMT-0] como huso horario base para crear los objetos calendar, java.util.Date y timestamp.

Si desea más información, consulte el tema sobre la consulta de datos en varios husos horarios en la *Guía de programación*.

Capítulo 3. visión general de integración de memoria caché: JPA, sesiones y memoria caché dinámica

El elemento decisivo que proporciona a WebSphere eXtreme Scale la capacidad de ejecutarse con tal versatilidad y fiabilidad es su aplicación de conceptos de colocación en memoria caché para optimizar la persistencia y la recogida de datos en prácticamente cualquier entorno.

Cargadores JPA

Java Persistence API (JPA) es una especificación que permite la correlación de objetos Java con bases de datos relacionales. JPA contiene una especificación de correlación de objetos relacionales (ORM) completa que utiliza las anotaciones de metadatos de lenguaje Java, los descriptores XML, o ambos, para definir la correlación entre los objetos Java y una base de datos relacional. Hay diversas implementaciones de código abierto y comerciales disponibles.

Puede utilizar una implementación de un plug-in de cargador de Java Persistence API (JPA) con eXtreme Scale para interactuar con cualquier base de datos soportada por su cargador elegido.Para utilizar JPA, debe tener un proveedor JPA soportado como, por ejemplo, OpenJPA o Hibernate, archivos JAR y un archivo `META-INF/persistence.xml` en la classpath.

Los plug-ins de JPALoader com.ibm.websphere.objectgrid.jpa.JPALoader y JPAEntityLoader com.ibm.websphere.objectgrid.jpa.JPAEntityLoader son dos plug-ins del cargador JPA incorporados que se utilizan para sincronizar las correlaciones de ObjectGrid con una base de datos. Debe tener una implementación JPA como, por ejemplo, Hibernate o OpenJPA, para utilizar esta característica. La base de datos puede ser cualquier programa de fondo soportado por el proveedor JPA elegido.

Puede utilizar el plug-in JPALoader al almacenar datos utilizando la API ObjectMap. Utilice el plug-in JPAEntityLoader al almacenar los datos utilizando la API EntityManager.

Arquitectura del cargador JPA

El cargador JPA se utiliza para las correlaciones de eXtreme Scale que almacenan los objetos POJO (Plain Old Java Object).

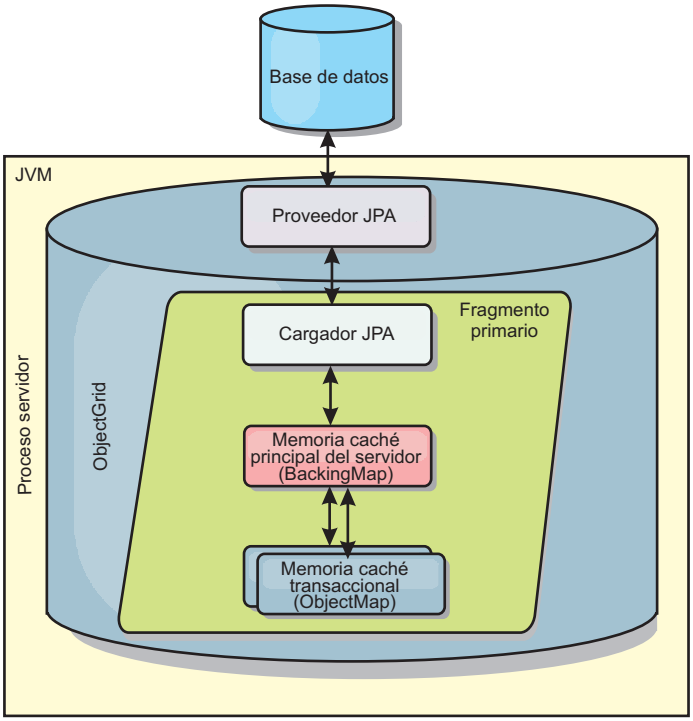


Figura 29. Arquitectura de cargador JPA

Cuando se llama un método ObjectMap.get(Object key), eXtreme Scale comprueba primero si la entrada se incluye en la capa ObjectMap. En caso negativo, el tiempo de ejecución delega la solicitud al cargador JPA. Después de solicitar la carga de la clave, JPALoader llama el método JPA EntityManager.find(Object key) para encontrar los datos de la capa JPA. Si los datos están contenidos en el gestor de entidades JPA, se devuelve; de lo contrario, el proveedor JPA interactúa con la base de datos para obtener el valor.

Cuando se produce una actualización en ObjectMap, por ejemplo, mediante el uso del método ObjectMap.update(Object key, Object value), el tiempo de ejecución de eXtreme Scale crea un LogElement para esta actualización y lo envía a JPALoader. JPALoader llama el método JPA EntityManager.merge(Object value) para actualizar el valor en la base de datos.

En JPAEntityLoader, también están implicadas las mismas cuatro capas. Sin embargo, dado que se utiliza el plug-in JPAEntityLoader para las correlaciones que almacenan las entidades de eXtreme Scale, las relaciones entre las entidades podrían complicar el escenario de uso. Se distingue una entidad eXtreme Scale de la entidad JPA. Si desea más información, consulte la información sobre el plug-in JPAEntityLoader en *Guía de programación*.

Métodos

Los cargadores proporcionan tres métodos principales:

1. get: devuelve una lista de valores que corresponden a la lista de claves que se pasan recuperando los datos que utilizan JPA. El método utiliza JPA para encontrar las entidades en la base de datos. Para el plug-in JPALoader, la lista devuelva contiene una lista de entidades JPA directamente de la operación de

búsqueda. Para el plug-in JPAEntityLoader, la lista devuelta contiene los tuples de valor de entidad eXtreme Scale que se han convertido a partir de las entidades JPA.

2. batchUpdate: graba los datos de las correlaciones ObjectGrid en la base de datos. En función de los distintos tipos de operación (insertar, actualizar o suprimir), el cargador utiliza las operaciones de persistir, fusionar y eliminar de JPA para actualizar los datos en la base de datos. En el caso de JPALoader, los objetos de la correlación se utilizan directamente como entidades JPA. En el caso de JPAEntityLoader, los tuples de entidad de la correlación se convierten en objetos que se utilizan como entidades JPA.
3. preloadMap: precarga la correlación utilizando el método de cargador de cliente ClientLoader.load. Para las correlaciones con particiones, sólo se llama al método preloadMap en una partición. La partición se especifica en la propiedad preloadPartition de la clase JPALoader o JPAEntityLoader. Si el valor preloadPartition se establece en un valor menor que cero, o mayor que value (*número_total_de_particiones* - 1), se inhabilita la precarga.

Ambos plug-ins, JPALoader y JPAEntityLoader, trabajan con la clase JPATxCallback para coordinar las transacciones eXtreme Scale y las transacciones JPA. Se debe configurar JPATxCallback en la instancia de ObjectGrid para utilizar estos dos cargadores.

Configuración y programación

Si desea más información sobre cómo configurar los cargadores JPA, consulte la información sobre los cargadores JPA en *Guía de administración*. Si desea más información sobre cómo programar cargadores JPA, consulte *Guía de programación*.

Plug-in de memoria caché JPA

WebSphere eXtreme Scale incluye los plug-ins de memoria caché de nivel (L2) para los proveedores OpenJPA e Hibernate Java Persistence API (JPA).

EL uso de eXtreme Scale como un proveedor de memoria caché de nivel 2 aumenta el rendimiento al leer y consultar datos y reduce la carga de la base de datos. WebSphere eXtreme Scale tiene ventajas sobre las implementaciones de memoria caché incorporadas porque la memoria caché se duplica automáticamente entre todos los procesos. Cuando un cliente almacena en memoria caché un valor, todos los demás clientes son capaces de utilizar el valor almacenado en memoria caché que está localmente en la memoria.

Con los plug-ins de memoria caché de ObjectGrid OpenJPA e Hibernate, podrá crear tres tipos de topología: incorporada, incorporada con particiones y remota.

Topología incorporada

Una topología incorporada crea un servidor eXtreme Scale dentro del espacio de proceso de cada aplicación. OpenJPA e Hibernate leen directamente con la copia en memoria de la memoria caché y escriben en todas las demás copias. Puede mejorar el rendimiento de la escritura utilizando la réplica asíncrona. El rendimiento de esta topología predeterminada es mejor cuando el volumen de datos en caché es lo suficientemente pequeño para caber en un solo proceso.

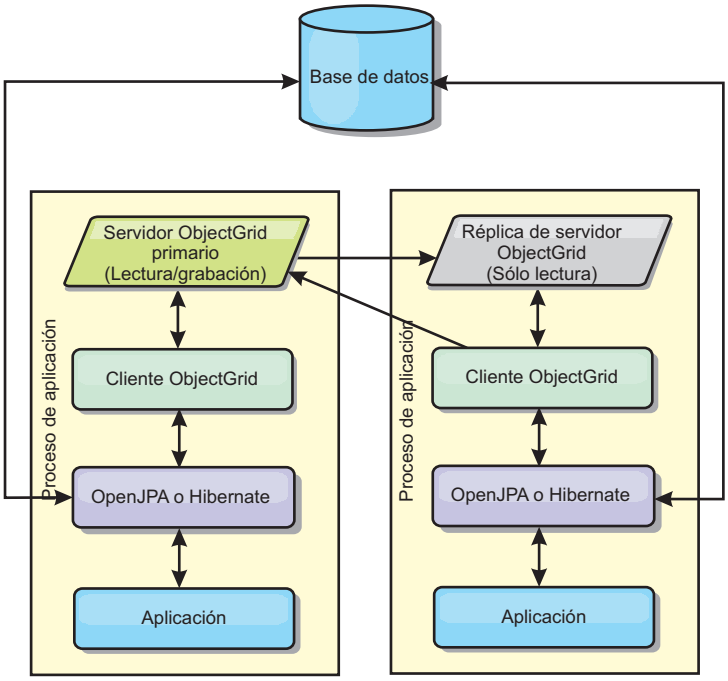


Figura 30. Topología incorporada JPA

Ventajas:

- Todas las lecturas de la memoria caché son muy rápidas, los accesos son locales.
- La configuración es sencilla.

Limitaciones:

- El volumen de los datos se limita al tamaño del proceso.
- Todas las actualizaciones de la memoria caché se envían a un proceso.

Topología incorporada con particiones

Cuando el volumen de los datos de la memoria caché es tan grande que no cabe en un único proceso, la topología incorporada con particiones utiliza las particiones de ObjectGrid para dividir los datos en varios procesos. El rendimiento no es tan alto como el de la topología incorporada porque la mayoría de las lecturas de la memoria caché son remotas. Sin embargo, puede seguir utilizando esta opción cuando la latencia de la base de datos es alta.

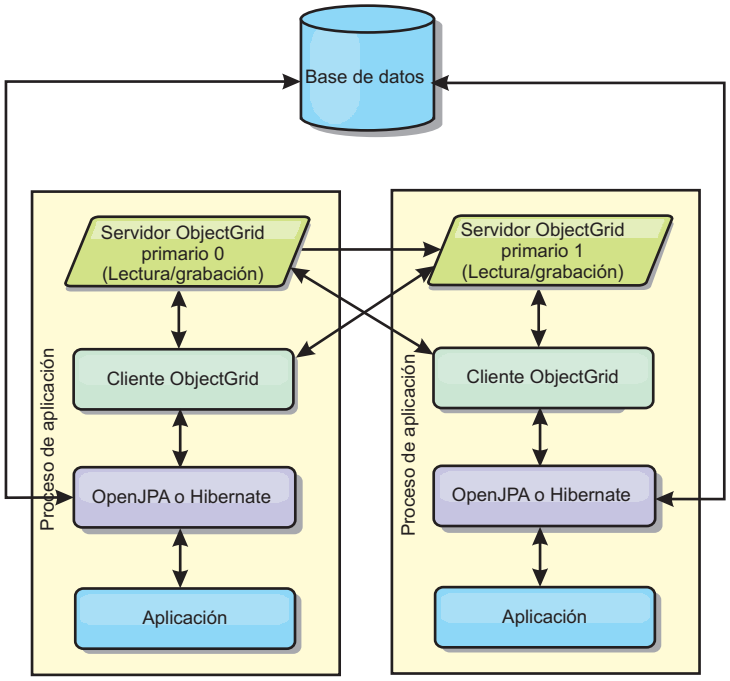


Figura 31. Topología incorporada con particiones JPA

Ventajas:

- Almacena grandes cantidades de datos.
- La configuración es sencilla.
- Las actualizaciones de la memoria caché se reparten en diversos procesos.

Limitación:

- La mayoría de las lecturas y actualizaciones de la memoria caché son remotas.

Por ejemplo, para almacenar en la memoria caché 10 GB de datos con un máximo de 1 GB por JVM, son necesarias diez Máquinas virtuales Java. Por lo tanto, el número de particiones debe establecerse en 10 o más. Lo ideal es establecer el número de particiones en un número primo, donde cada fragmento almacena una cantidad razonable de memoria. Normalmente, el valor numberOfPartitions es igual al número de Máquinas virtuales Java. Con este valor, cada JVM almacena una partición. Si habilita las réplicas, debe aumentar el número de Máquinas virtuales Java en el sistema. De lo contrario, cada JVM también almacena una partición de réplica, que consume tanta memoria como una partición primaria.

Lea la información sobre el dimensionamiento de la memoria y el cálculo del número de particiones en la *Guía de administración* para maximizar el rendimiento de su configuración elegida.

Por ejemplo, en un sistema con 4 Máquinas virtuales Java, y el valor de numberOfPartitions de 4, cada JVM aloja una partición primaria. Una operación de lectura tiene un 25 por ciento de posibilidades de captar datos desde una partición disponible localmente, que es mucho más rápido que obtener los datos de una JVM remota. Si una operación de lectura como, por ejemplo, ejecutar una consulta, debe captar una colección de datos que implican 4 particiones de manera uniforme, el 75 por ciento de las llamadas son remotas y el otro 25 por ciento son locales. Si el valor ReplicaMode se establece en `SYNC` o `ASYNC` y el valor ReplicaReadEnabled está establecido en `true`, se crean las cuatro particiones de

réplica y se expanden a lo largo de las cuatro Máquinas virtuales Java. Cada JVM aloja una partición primaria y una partición de réplica. La probabilidad de que la operación de lectura se ejecute de forma local aumenta a un 50 por ciento. La operación de lectura que capta una colección de datos que implican cuatro particiones de manera uniforme tiene un 50 por ciento de llamadas remotas y un 50 por ciento de llamadas locales. Las llamadas locales son mucho más rápidas que las memorias remotas. Siempre que se producen llamadas remotas, baja el rendimiento.

Topología remota

Una topología remota almacena todos los datos almacenados en la memoria caché en uno o más procesos separados, reduciendo el uso de la memoria de los procesos de la aplicación. Puede sacar partido de la distribución de los datos en procesos distintos desplegando una cuadrícula de eXtreme Scale particionada y replicada. A diferencia de las configuraciones incorporada y con partición incorporada descritas en las secciones anteriores, si desea gestionar la cuadrícula remota, debe hacerlo independientemente de la aplicación y del proveedor JPA. Lea la documentación sobre la supervisión de su entorno de despliegue para obtener más información sobre cómo gestionar un despliegue de cuadrícula de eXtreme Scale.

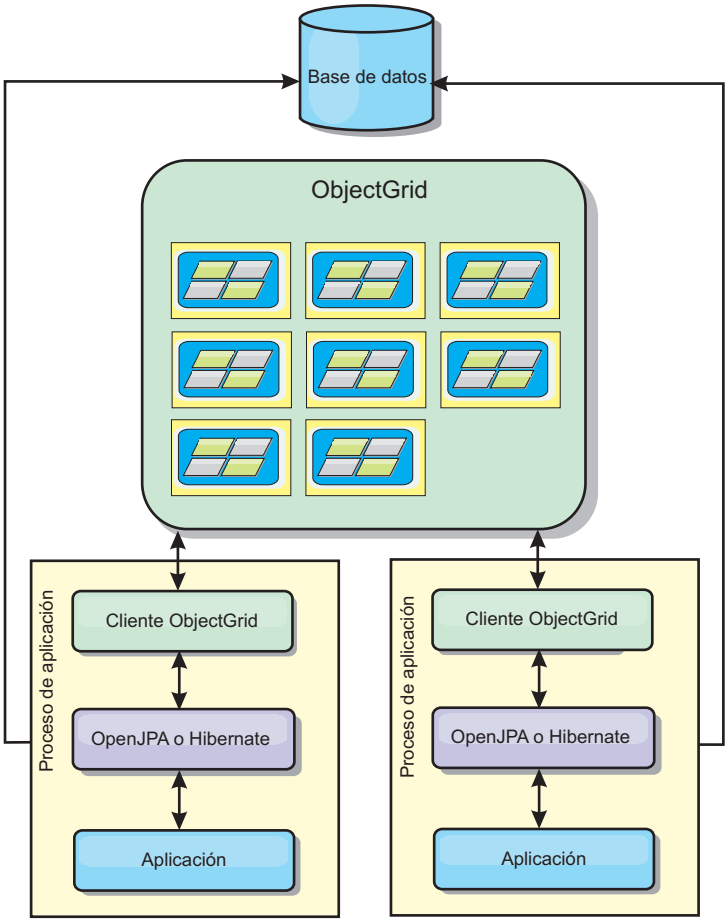


Figura 32. Topología remota JPA

Ventajas:

- Almacena grandes cantidades de datos.
- El proceso de aplicaciones está libre de los datos en memoria caché.

- Las actualizaciones de la memoria caché se reparten en diversos procesos.
- Opciones de configuración muy flexibles.

Limitación:

- Todas las lecturas y actualizaciones de la memoria caché son remotas.

Configuración

Si desea más información sobre cómo configurar los plug-ins de memoria caché JPA, consulte la sección de plug-ins en *Guía de programación*.

Gestión de sesiones HTTP

El gestor de réplica de sesión que se suministra con WebSphere eXtreme Scale funciona con el gestor de sesiones predeterminado del servidor de aplicaciones para hacer una réplica de los datos de sesión de un proceso a otro para dar soporte a la alta disponibilidad de datos de sesión del usuario.

Características

El gestor de sesiones se ha diseñado de modo que pueda ejecutarse en cualquier contenedor Java Platform, Enterprise Edition versión 1.4. Dado que el gestor de sesiones no tiene dependencias en las API de WebSphere, puede admitir varias versiones de WebSphere Application Server así como de entornos de servidor de aplicaciones de proveedores.

El gestor de sesiones HTTP proporciona funciones de réplica de sesiones para una aplicación asociada. El gestor de réplica de sesión funciona con el gestor de sesiones de contenedor web para crear sesiones HTTP y gestionar los ciclos de vida de las sesiones HTTP asociadas a la aplicación. Estas actividades de gestión de ciclo de vida incluyen: la invalidación de sesiones basadas en un tiempo de espera o una llamada a un servlet explícito o JavaServer Pages (JSP) y la invocación de escuchas de sesión asociados a la sesión o a la aplicación web. El gestor de sesiones persiste sus sesiones en una instancia de ObjectGrid. Esta instancia puede ser una instancia local, en memoria o una instancia completamente replicada, agrupada en clúster y particionada. El uso de esta última topología permite al gestor de sesiones proporcionar soporte de migración tras error de sesiones HTTP cuando los servidores de aplicaciones concluyen o se cierren de forma inesperada. El gestor de sesiones también puede funcionar en entornos que no admitan afinidad, si la afinidad no la aplica un nivel de equilibrador de carga que distribuya solicitudes al nivel de servidor de aplicaciones.

Escenarios de uso

El gestor de sesiones se puede utilizar en los siguientes escenarios:

- En entornos que utilizan servidores de aplicaciones en diferentes versiones de WebSphere Application Server, como por ejemplo en un escenario de migración clásico.
- En despliegues que utilizan servidores de aplicaciones de proveedores diferentes. Por ejemplo, una aplicación que se desarrolla en servidores de aplicaciones de código abierto y que se aloja en WebSphere Application Server. Otro ejemplo es una aplicación que se promociona de la fase intermedia a la de producción. Es posible realizar una migración sin interrupciones de estas versiones del servidor de aplicación mientras todas las sesiones HTTP están activas y dan servicio.

- En entornos que requieren que el usuario persista las sesiones con niveles superiores de calidad de servicio (QoS) y mejores garantías de disponibilidad de sesiones durante la sustitución por anomalía del servidor que los niveles Qos predeterminados de WebSphere Application Server.
- En un entorno donde la afinidad de sesiones no puede garantizarse, o en entornos en los que la afinidad se mantiene mediante un equilibrador de carga del proveedor y el mecanismo de afinidad debe personalizarse de acuerdo con dicho equilibrador de carga.
- En un entorno donde se descarga la sobrecarga de la gestión de sesiones y se almacena en un proceso Java externo.
- En varias células que permiten la sustitución por anomalía de las sesiones entre las células.
- En múltiples centros de datos o múltiples zonas.

Cómo funciona el gestor de sesiones

El gestor de réplica de sesión utiliza un escucha de sesión estándar para escuchar los cambios de datos de la sesión y persiste los datos de la sesión en una instancia de ObjectGrid local o remotamente. Los datos de la sesión se recargan en la vía de acceso de la solicitud mediante el servlet estándar desde la instancia de ObjectGrid local o remotamente. Puede añadir el escucha de sesión y el filtro de servlet en todos los módulos web de la aplicación con las herramientas que se proporcionan con WebSphere eXtreme Scale. También puede añadir estos escuchas y filtros de forma manual al descriptor de despliegue web de la aplicación.

Este gestor de réplica de sesión funciona con el gestor de sesiones de contenedor web de cada proveedor para replicar datos de sesión en máquinas virtuales Java. Cuando el servidor original queda inactivo, los usuarios pueden recuperar datos de sesión de otros servidores.

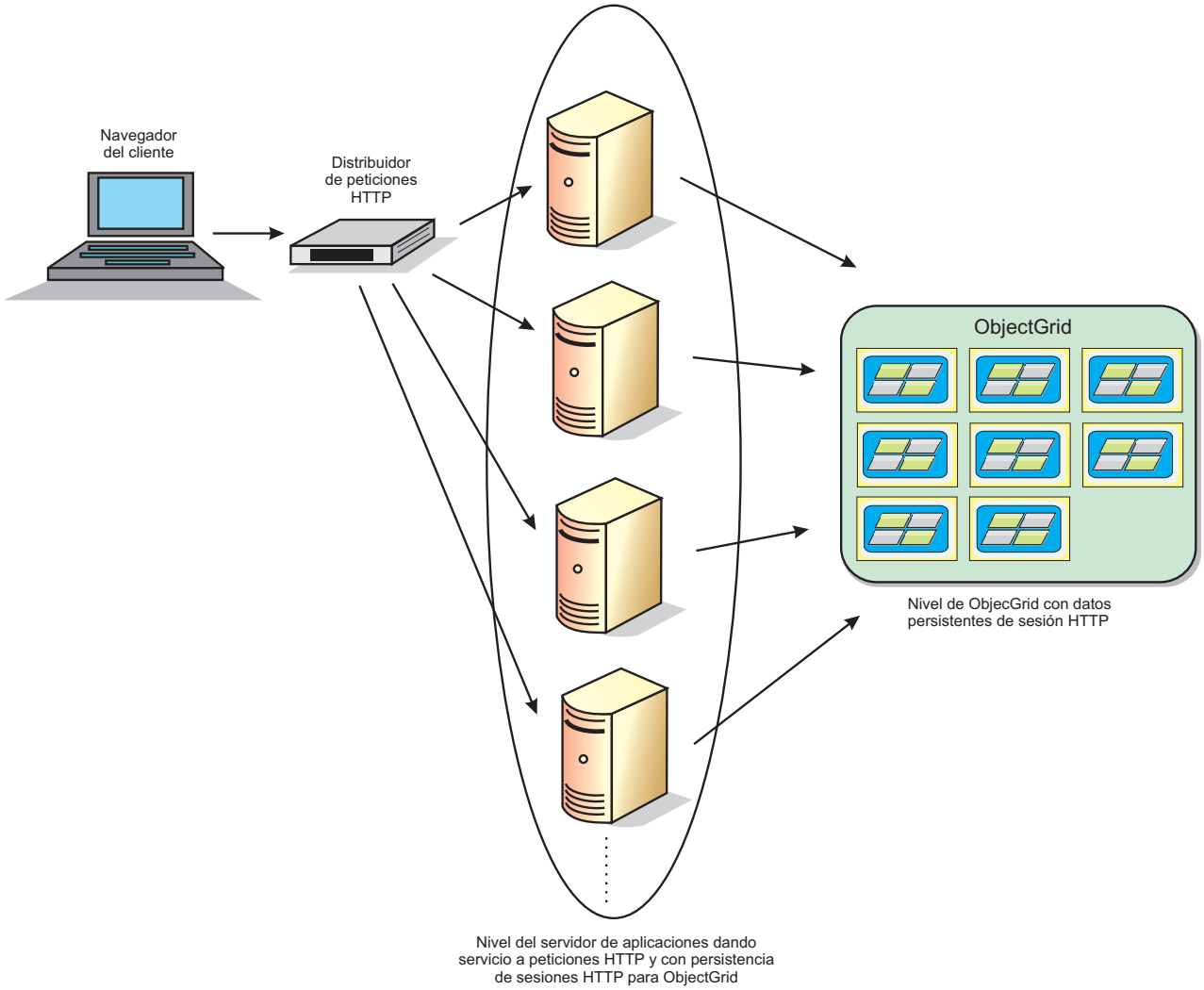


Figura 33. Topología de gestión de sesiones HTTP con una configuración de contenedor remoto

Topologías de despliegue

El gestor de sesiones puede configurarse mediante el uso de dos escenarios de despliegue dinámico diferentes:

- **Contenedores de eXtreme Scale incorporados, conectados a la red**

  En este escenario, los servidores eXtreme Scale se colocan en los mismos procesos que los servlets. El gestor de sesiones se puede comunicar directamente con la instancia de ObjectGrid local, lo que evita costosos retrasos de red. Este caso de ejemplo es preferible al realizar la ejecución con afinidad y cuando es crítico el rendimiento.
- **Contenedores de eXtreme Scale remotos, conectados a la red**

  En este escenario, los servidores eXtreme Scale ejecutan en procesos externos el proceso en el que se ejecutan los servlets. El gestor de sesiones se comunica con una cuadrícula de servidor de eXtreme Scale remoto. Este caso de ejemplo es preferible cuando el nivel del contenedor web no tiene la memoria para almacenar los datos de sesión. Los datos de sesión se descargarán en un nivel aparte, lo que provocará un uso menor de memoria en el nivel de contenedor web, pero una mayor latencia debido a la ubicación remota de los datos.

Inicio del contenedor incorporado genérico

eXtreme Scale inicia automáticamente un contenedor de ObjectGrid incorporado dentro de cualquier proceso servidor de aplicaciones cuando el contenedor web inicializa el escucha de sesión o el filtro de servlet, si la propiedad objectGridType está establecida en EMBEDDED. Consulte el apartado Parámetros de inicialización del contexto del servlet para obtener detalles.

No es necesario que empaquete un archivo ObjectGrid.xml ni un archivo objectGridDeployment.xml en el archivo WAR o EAR de su aplicación web con eXtreme Scale Versión 7.1. Los archivos ObjectGrid.xml y objectGridDeployment.xml predeterminados están empaquetados en el JAR del producto. De forma predeterminada, se crean correlaciones dinámicas para distintos contextos de aplicaciones web. Se siguen admitiendo las correlaciones estáticas de eXtreme Scale.

Este enfoque para iniciar contenedores ObjectGrid incorporados se aplica a cualquier tipo de servidor de aplicaciones. Los enfoques relacionados con un componente WebSphere Application Server o WebSphere Application Server Community Edition GBean están en desuso.

Gestor de réplica de sesión basado en escucha

El gestor de réplica de sesión de eXtreme Scale que se suministra con WebSphere eXtreme Scale puede funcionar con el gestor de sesiones predeterminado en el servidor de aplicaciones para replicar datos de sesión de un proceso a otro proceso para dar soporte a la alta disponibilidad de datos de sesión del usuario.

El gestor de sesiones se ha diseñado de modo que pueda ejecutarse en cualquier contenedor Java™ Platform, Enterprise Edition Versión 1.4. El gestor de sesiones no tiene dependencias en las API de WebSphere, por lo que puede admitir varias versiones de WebSphere Application Server y entornos de servidor de aplicaciones de proveedores.

El gestor de sesiones HTTP proporciona funciones de réplica de sesiones para una aplicación asociada. El gestor de réplica de sesión funciona con el gestor de sesiones del contenedor web para crear sesiones HTTP y gestionar los ciclos de vida de las sesiones HTTP asociadas con la aplicación. Estas actividades de gestión de ciclo de vida incluyen: la invalidación de sesiones basadas en un tiempo de espera o una llamada a un servlet explícito o JavaServer Pages (JSP) y la invocación de escuchas de sesión asociados a la sesión o a la aplicación web. El gestor de sesiones persiste sus sesiones en una instancia de ObjectGrid. Esta instancia puede ser una instancia local, en memoria o una instancia completamente replicada, agrupada en clúster y particionada. El uso de esta última topología permite al gestor de sesiones proporcionar soporte de sustitución por anomalía de sesiones HTTP cuando los servidores de aplicaciones concluyen o se cierren de forma inesperada. El gestor de sesiones también puede funcionar en entornos que no admitan afinidad, si la afinidad no la aplica un nivel de equilibrador de carga que distribuya solicitudes al nivel de servidor de aplicaciones.

Escenarios de uso

El gestor de sesiones se puede utilizar en los siguientes escenarios:

- En entornos que utilizan servidores de aplicaciones en diferentes versiones de WebSphere Application Server, como por ejemplo en un escenario de migración clásico.

- En despliegues que utilizan servidores de aplicaciones de proveedores diferentes. Por ejemplo, una aplicación que se desarrolla en servidores de aplicaciones de código abierto y que se aloja en WebSphere Application Server. Otro ejemplo es una aplicación que se promociona de la fase intermedia a la de producción. Es posible realizar una migración sin interrupciones de estas versiones del servidor de aplicación mientras todas las sesiones HTTP están activas y dan servicio.
- En entornos que requieren que el usuario persista las sesiones con niveles superiores de calidad de servicio (QoS) y mejores garantías de disponibilidad de sesiones durante la sustitución por anomalía del servidor que los niveles Qos predeterminados de WebSphere Application Server.
- En un entorno donde la afinidad de sesiones no puede garantizarse, o en entornos en los que la afinidad se mantiene mediante un equilibrador de carga del proveedor y el mecanismo de afinidad debe personalizarse de acuerdo con dicho equilibrador de carga.
- En un entorno donde se descarga la sobrecarga de la gestión de sesiones y se almacena en un proceso Java externo.
- En varias células que permiten la sustitución por anomalía de las sesiones entre las células.
- En múltiples centros de datos o múltiples zonas.

Detalles del gestor de sesiones

El gestor de réplica de sesión utiliza un escucha de sesión estándar para escuchar los cambios de datos de la sesión y persiste los datos de la sesión en una instancia de ObjectGrid local o remotamente. Los datos de la sesión se recargan en la vía de acceso de la solicitud mediante el servlet estándar desde la instancia de ObjectGrid local o remotamente. Puede añadir el escucha de sesión y el filtro de servlet en todos los módulos web de la aplicación con las herramientas que se proporcionan con WebSphere eXtreme Scale. También puede añadir estos escuchas y filtros de forma manual al descriptor de despliegue web de la aplicación.

El gestor de réplica de sesión funciona con el gestor de sesiones base de cada proveedor para replicar datos de sesión de aplicación. Tenga en cuenta estas consideraciones.

- Elija los contenedores ObjectGrid incorporados o los contenedores ObjectGrid remotos, según los requisitos de rendimiento y los tamaños de datos. El caso incorporado proporciona la configuración más sencilla y de mejor rendimiento.
- Elija el número de contenedores ObjectGrid remotos por contenedor web según los tamaños de datos de usuario.
- Elija almacenar los datos de sesión completos o almacene cada atributo por independiente, según el número y el tamaño de los datos de usuario de atributos y de las frecuencias de cambio.
- Elija el intervalo de réplica. Un intervalo de réplica menos agresivo proporciona un mayor rendimiento.
- Elija el tamaño de tabla de sesión para equilibrar el tamaño de memoria local y el rendimiento. El producto descarga los datos de usuario de la sesión cuando se alcanza el tamaño máximo de memoria caché de sesión local.
- El producto admite las sesiones HTTP en el contexto de la aplicación, según la especificación de Servlet, para impedir problemas de conflicto de denominación de atributos de uso y seguridad. Su clase de singleton puede compartir sesiones en todo el contexto de aplicación.

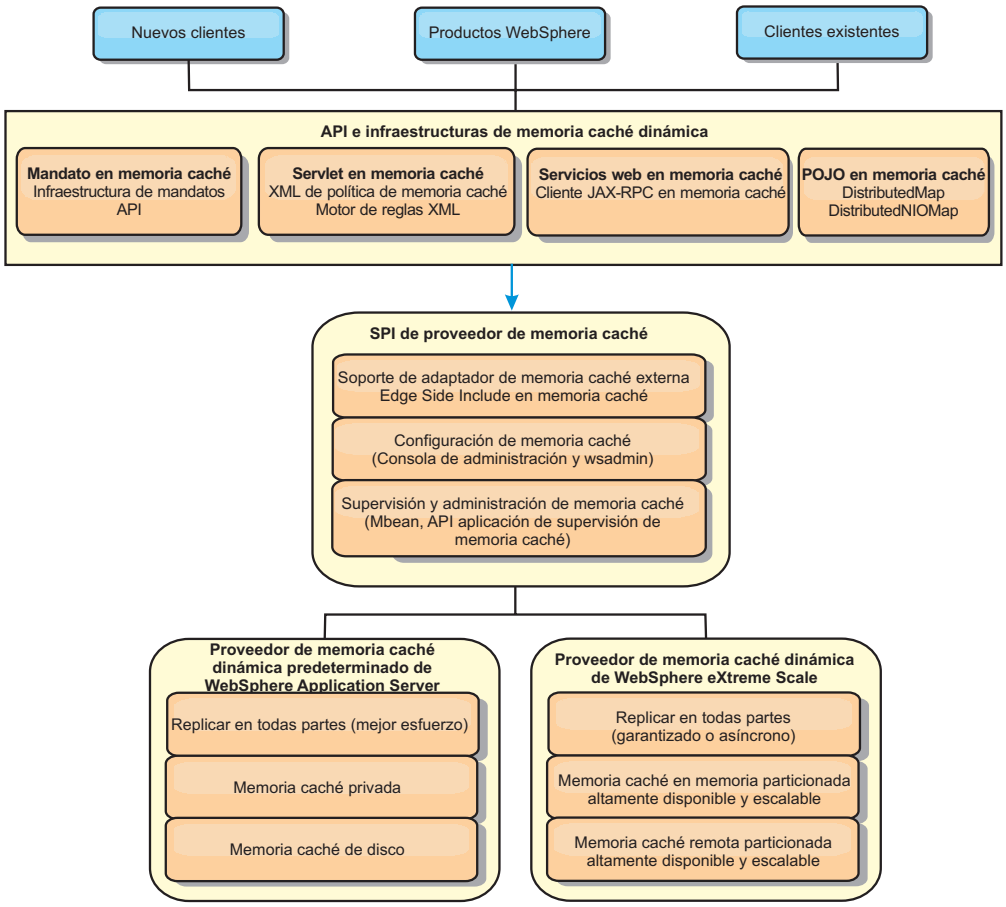
- El gestor de réplica de sesión hace una réplica de los datos de la sesión para una alta disponibilidad, estando a la escucha de los cambios de datos de la sesión y volviendo a cargar los datos de sesión almacenados bajo petición. Esto se consigue reutilizando el gestor de sesiones base del contenedor web de cada proveedor. La sesión se enlaza junta a través de varios ID de enlace nativos. Ha habido una migración tras error de los datos de sesión de un contenedor web a otro al volver a cargar los datos de sesión de la instancia de ObjectGrid. El ID de sesión y la hora de creación podrían ser distintos antes y después de la migración tras error.

Proveedor de memoria caché dinámica

La API de memoria caché dinámica está disponible para las aplicaciones Java EE desplegadas en WebSphere Application Server. Se puede sacar el máximo partido del proveedor de memoria caché dinámica para almacenar en la memoria caché los datos empresariales, el HTML generado o para sincronizar los datos de la memoria caché en la célula utilizando el servicio de duplicación de datos (DRS).

Visión general

Previamente, el único proveedor de servicios para la API de memoria caché dinámica era el motor de la memoria caché dinámica incorporada en WebSphere Application Server. Los clientes pueden utilizar la interfaz del proveedor de servicios de memoria caché dinámica en WebSphere Application Server para conectarse a la memoria caché dinámica de eXtreme Scale. Configurando esta prestación, podrá habilitar aplicaciones escritas con la API de memoria caché dinámica o aplicaciones que utilizan la memoria caché de nivel de contenedor (como, por ejemplo, servlets) para sacar el máximo partido de las características y capacidades de rendimiento de WebSphere eXtreme Scale.



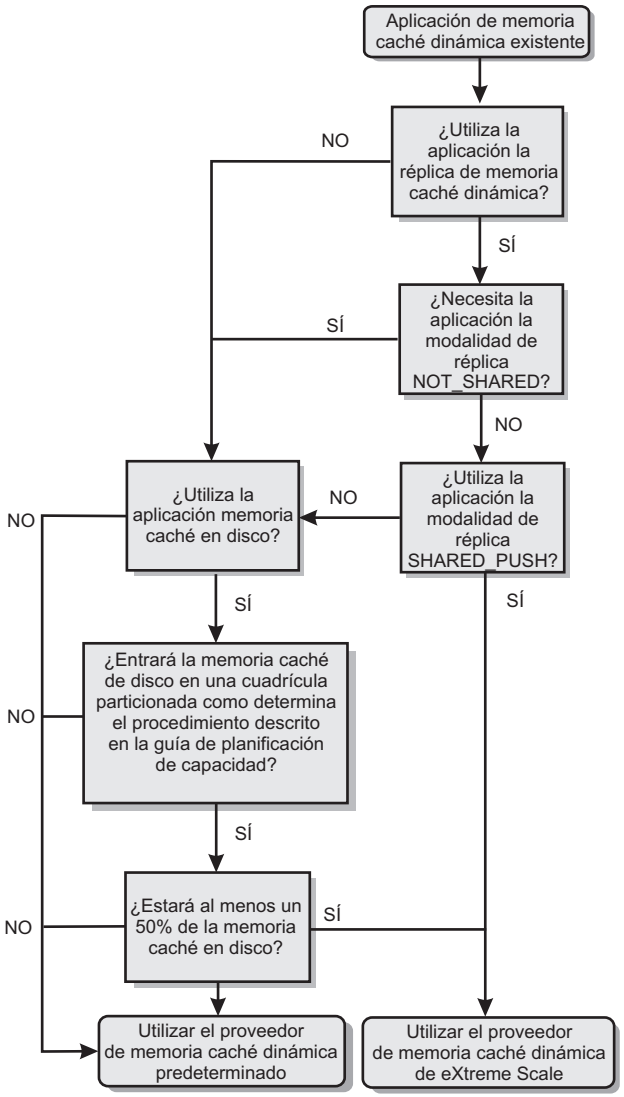
Puede instalar y configurar el proveedor de memoria caché dinámica tal como se describe en Configuración del proveedor de memoria caché dinámica para WebSphere eXtreme Scale.

Decidir cómo sacar partido de WebSphere eXtreme Scale

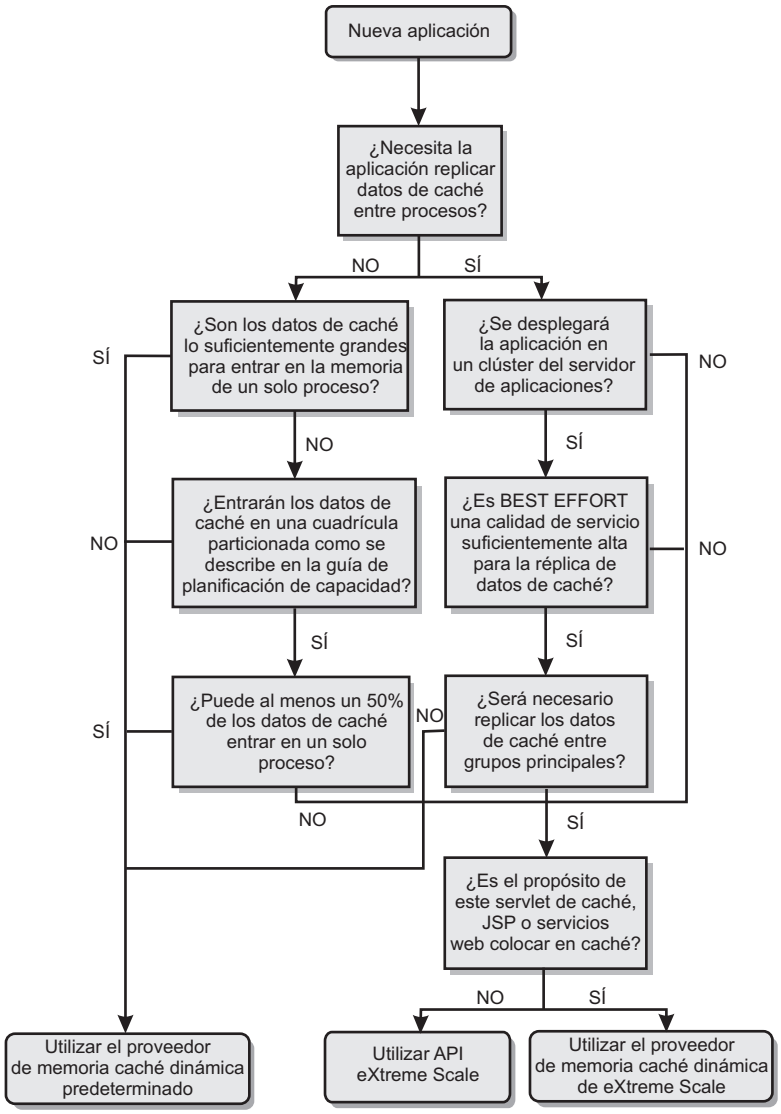
Las características disponibles en WebSphere eXtreme Scale aumentan de forma significativa las capacidades distribuidas de la API de memoria caché dinámica más allá de lo que ofrece el motor de memoria caché dinámica predeterminado y el servicio de duplicación de datos. Con eXtreme Scale, puede crear memorias caché que se distribuyen verdaderamente entre varios servidores, en lugar de sólo duplicarlas y sincronizarlas entre los servidores. Además, las memorias caché de eXtreme Scale son transaccionales y están disponibles, lo que asegura que cada servidor ve los mismos contenidos para el servicio de memoria caché dinámica. WebSphere eXtreme Scale ofrece una mayor calidad de servicio para la réplica de memoria caché que DRS.

Sin embargo, estas ventajas no implican que el proveedor de memoria caché dinámica de eXtreme Scale sea la opción correcta para cada aplicación. Utilice los árboles de decisiones y la matriz de comparaciones de característica siguiente para determinar qué tecnología se adapta mejora a la aplicación.

Árbol de decisiones para migrar las aplicaciones de la memoria caché dinámica existente



Árbol de decisiones para seleccionar un proveedor de memoria caché para las nuevas aplicaciones



Comparación de características

Tabla 7. Comparación de características

Características de memoria caché	Proveedor predeterminado	Proveedor de eXtreme Scale	API de eXtreme Scale
Memoria caché local en memoria	x	x	x
Memoria caché distribuida	Incorporado	Incorporado, particionado-incorporado y particionado-remoto	Varios
Ampliable de forma lineal		x	x
Réplica fiable (síncrona)		ORB	ORB

Tabla 7. Comparación de características (continuación)

Características de memoria caché	Proveedor predeterminado	Proveedor de eXtreme Scale	API de eXtreme Scale
Desbordamiento de disco	x		
Desalojo	LRU/TTL/basado en almacenamiento dinámico	LRU/TTL (por partición)	Varios
Invalidación	x	x	x
Relaciones	ID de dependencia, plantillas,	ID de dependencia, plantillas,	x
Búsquedas sin clave			Consulte e índice
Integración de fondo			Cargadores
Transaccional		Implícita	x
Almacenamiento basado en clave	x	x	x
Sucesos y receptores	x	x	x
Integración de WebSphere Application Server	Sólo una única célula	Varias células	Célula independiente
Soporte de Java Standard Edition		x	x
Supervisión y estadísticas	x	x	x
Seguridad	x	x	x

Tabla 8. Integración de tecnología sin fisuras

Características de memoria caché	Proveedor predeterminado	Proveedor de eXtreme Scale	API de eXtreme Scale
Colocación en memoria caché de los resultados del servlet/JSP de WebSphere Application Server	V5.1+	V6.1.0.25+	
Colocación en memoria caché del resultado de WebSphere Application Server Web Services (JAX-RPC)	V5.1+	V6.1.0.25+	
Colocación en memoria caché de la sesión HTTP			x
Proveedor de memoria caché para OpenJPA e Hibernate			x

Tabla 8. Integración de tecnología sin fisuras (continuación)

Sincronización de la base de datos utilizando OpenJPA e Hibernate			x

Tabla 9. Interfaces de programación

Características de memoria caché	Proveedor predeterminado	Proveedor de eXtreme Scale	API de eXtreme Scale
API basada en mandato	API de infraestructura de mandatos	API de infraestructura de mandatos	API de DataGrid
API basada en correlación	API DistributedMap	API DistributedMap	API ObjectMap
API EntityManager			x

Si desea una descripción más detallada sobre cómo funcionan las memorias caché distruidas de eXtreme Scale, consulte la información de configuración de despliegue en la *Guía de administración*.

Nota: Una memoria caché distribuida de eXtreme Scale sólo puede almacenar entradas en las que la clave y el valor ambos implementan la interfaz java.io.Serializable.

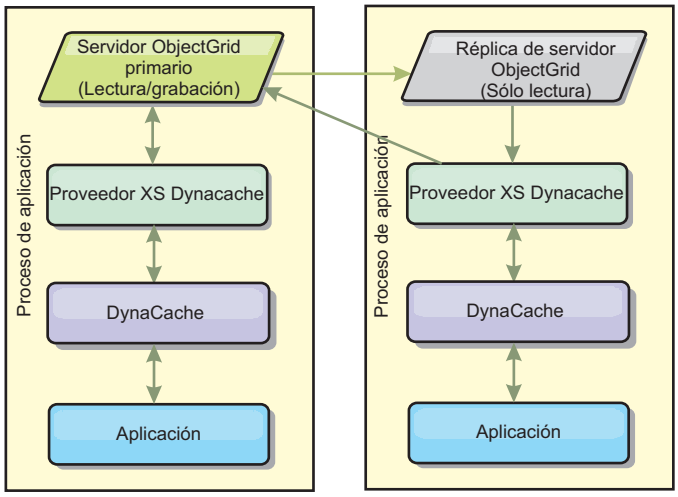
Tipos de topología

Un servicio de memoria caché dinámica creado con el proveedor eXtreme Scale se puede desplegar en cualquiera de las tres topologías disponibles, lo que le permite adaptar la memoria caché específicamente al rendimiento, los recursos y las necesidades administrativas. Estas topologías son: incorporado, particionado incorporado y remoto.

Topología incorporada

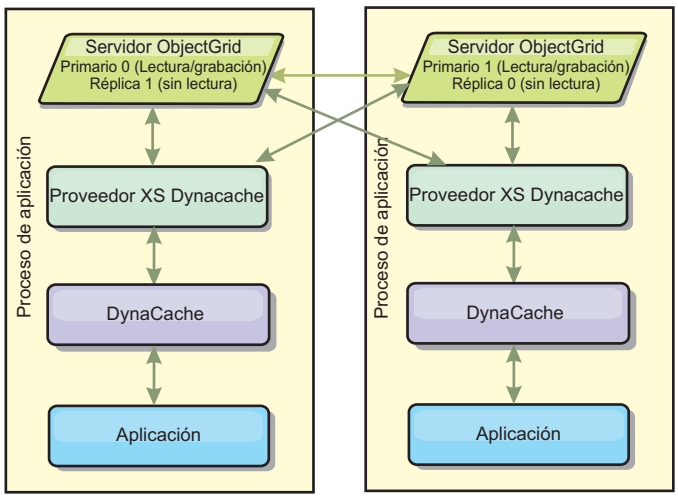
La topología incorporada es similar a la memoria caché dinámica predeterminada y al proveedor DRS. Las instancias de memoria caché distribuida creadas con la topología incorporada conservan una copia de la memoria caché dentro de cada proceso eXtreme Scale que accede al servicio de memoria caché dinámica, lo que permite que todas las operaciones de lectura se produzcan de forma local. Todas las operaciones de escritura pasan por un proceso de único servidor, en el que se gestionan los bloqueos transaccionales, antes de duplicarse el resto de los servidores. Consecuentemente, esta topología es mejor para las cargas de trabajo donde las operaciones de memoria caché-lectura superan en número a las operaciones de memoria caché-escritura.

Con la topología incorporada, las entradas de memoria caché nuevas o actualizadas no son visibles de forma inmediata en cada proceso de servidor único. Una entrada de caché no será visible, incluso para el servidor que lo ha generado, hasta que se propague a través de los servicios de duplicación asíncrona de WebSphere eXtreme Scale. Estos servicios funcionan tan rápido como lo permita el hardware, pero sigue habiendo un pequeño retardo. La topología incorporada se muestra en la siguiente imagen:



Topología incorporada con particiones

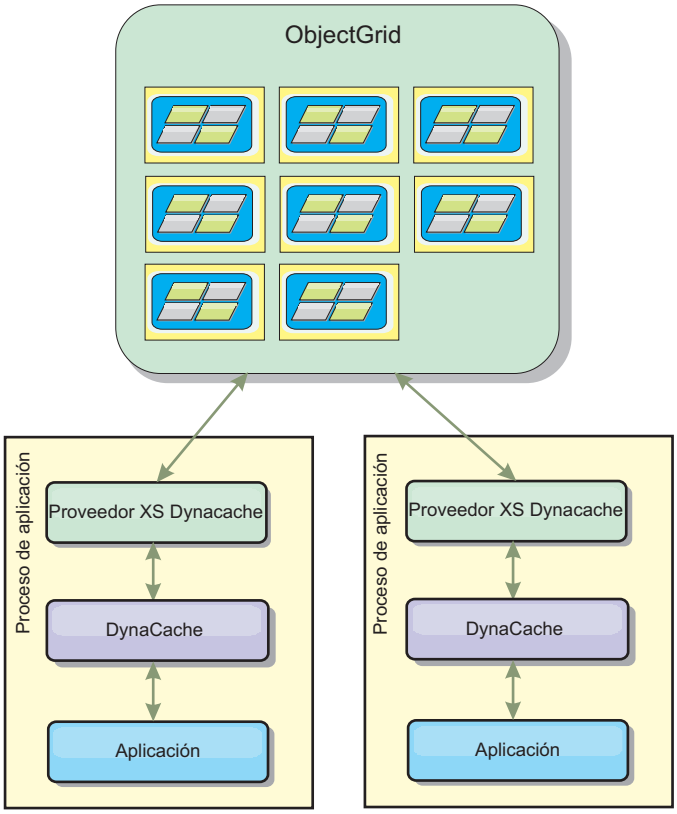
Para las cargas de trabajo donde se producen las escrituras de memoria caché tan a menudo o con más frecuencia que las lecturas, se recomiendan las topologías incorporadas con particiones o las remotas. La topología incorporada con particiones conserva todos los datos de la memoria caché dentro de los procesos WebSphere Application Server que acceden a la memoria caché. Sin embargo, cada proceso sólo almacena una parte de los datos de la memoria caché. Todas las lecturas y escrituras de los datos situados en esta "partición" pasan por el proceso, lo que significa que la mayoría de las solicitudes para la memoria caché se cumplirán con una llamada a procedimiento remoto. Esto genera una mayor latencia para las operaciones de lectura que la topología incorporada, pero la capacidad de la memoria caché distribuida para manejar las operaciones de lectura y escritura se ampliará de forma lineal con el número de procesos WebSphere Application Server que acceden a la memoria caché. Además, con esta topología, el tamaño máximo de la memoria caché no está vinculado al tamaño de un único proceso WebSphere. Puesto que cada proceso sólo alberga una parte de la memoria caché, el tamaño máximo de la memoria caché pasa a ser el tamaño agregado de todos los procesos, menos la sobrecarga del proceso. La topología incorporada con particiones se muestra en la siguiente imagen:



Por ejemplo, suponga que tiene una cuadrícula de procesos de servidor con 256 megabytes de almacenamiento dinámico libre para alojar el servicio de memoria caché dinámica. El proveedor de la memoria caché dinámica predeterminada y el proveedor eXtreme Scale que utiliza la topología incorporada se deben limitar ambos a un tamaño de memoria caché en memoria de 256 megabytes menos la sobrecarga. Consulte la sección Planificación de capacidad y Alta disponibilidad más adelante en este documento. El proveedor eXtreme Scale que utiliza la topología incorporada con particiones se limitará a un tamaño de memoria caché de un gigabyte menos la sobrecarga. De esta forma, el proveedor WebSphere eXtreme Scale posibilita tener servicios de memoria caché dinámica en memoria mayores que el tamaño de un proceso de servidor único. El proveedor de la memoria caché dinámica predeterminada se basa en el uso de una memoria caché de disco para permitir a las instancias de memoria caché crecer más allá del tamaño de un proceso único. En muchas situaciones, el proveedor WebSphere eXtreme Scale puede eliminar la necesidad de una memoria caché de disco y los caros sistemas de almacenamiento en disco que hacen que funcione.

Topología remota

La topología remota también se puede utilizar para eliminar la necesidad de una memoria caché de disco. La única diferencia entre las topologías remota e incorporada con particiones es que todos los datos de la memoria caché se almacenan fuera de los procesos WebSphere Application Server cuando se utiliza la topología remota. WebSphere eXtreme Scale soporta los procesos de contenedor autónomo para los datos de memoria caché. Estos procesos de contenedor tienen una sobrecarga menor que un proceso WebSphere Application Server y, además, no están limitados a utilizar una máquina virtual Java (JVM) determinada. Por ejemplo, un proceso WebSphere Application Server de 32 bits que está accediendo a los datos de un servicio de memoria caché dinámica podría estar situado en un proceso de contenedor eXtreme Scale que se ejecuta en una JVM de 64 bits. Esto permite a los usuarios sacar el máximo partido de la capacidad de memoria aumentada de los procesos de 64 bits para la colocación en memoria caché, sin generar la sobrecarga adicional de 64 bits para los procesos de servidor de aplicaciones. La topología remota se muestra en la siguiente imagen:



Compresión de datos

La compresión es otra característica de rendimiento ofrecida por el proveedor de memoria caché dinámica WebSphere eXtreme Scale que puede ayudar a los usuarios a gestionar la sobrecarga de la memoria caché. El proveedor de la memoria caché dinámica predeterminada no permite la compresión de los datos almacenados en la memoria caché. Con el proveedor eXtreme Scale, esto ahora es posible. La compresión de memoria caché mediante el algoritmo de deflate se puede habilitar en cualquiera de las tres topologías distribuidas. Habilitar la compresión aumentará la sobrecarga para las operaciones de lectura y escritura, pero aumentará drásticamente la densidad de la memoria caché para las aplicaciones, como la colocación en memoria caché de servlet y JSP.

Almacenamiento local de memoria caché en memoria

El proveedor de memoria caché dinámica de WebSphere eXtreme Scale se puede utilizar también para recuperar las instancias de memoria caché dinámica que tienen la **réplica inhabilitada**. Al igual que el proveedor de memoria caché dinámica predeterminado, estas memorias caché pueden almacenar datos no serializables. También pueden ofrecer un mejor rendimiento que el proveedor de memoria caché dinámica en grandes servidores de varios procesadores porque la vía de acceso del código eXtreme Scale está diseñado para maximizar la concurrencia de la memoria caché en memoria.

Diferencias funcionales del motor de memoria caché dinámica y eXtreme Scale

En el caso de las memorias de caché en memoria donde la réplica está inhabilitada, no debería haber ninguna diferencia funcional apreciable entre las memorias caché

respaldadas por el proveedor de la memoria caché dinámica predeterminada y WebSphere eXtreme Scale. Los usuarios no deberían advertir ninguna diferencia funcional entre las dos memorias caché, excepto que las memorias caché respaldadas por WebSphere eXtreme Scale no soportan las estadísticas ni la descarga de disco, ni las operaciones relacionadas con el tamaño de la memoria caché en memoria.

En el caso de las memorias caché donde está habilita la réplica, no habrá ninguna diferencia apreciable en los resultados devueltos por las mayoría de las llamadas de la API de memoria caché dinámica, independientemente de si el cliente está utilizando el proveedor de memoria caché dinámica o el proveedor de memoria caché dinámica eXtreme Scale. Para algunas operaciones, no podrá emular el comportamiento del motor de memoria caché dinámica utilizando eXtreme Scale.

Estadísticas de la memoria caché dinámica

Se informa de las estadísticas de memoria caché dinámica a través de la aplicación CacheMonitor o el MBean de memoria caché dinámica. Cuando se utiliza el proveedor de memoria caché dinámica eXtreme Scale, se informará de las estadísticas a través de estas interfaces, pero el contexto de los valores estadísticos serán diferentes.

Si se comparte una instancia de memoria caché dinámica entre tres servidores llamados A, B y C, el objeto de las estadísticas de memoria caché dinámica sólo devuelve las estadísticas para la copia de la memoria caché en el servidor que realizó la llamada. Si se recuperan las estadísticas en el servidor A, sólo reflejan la actividad en el servidor A.

Con eXtreme Scale, sólo hay una memoria caché distribuida compartida entre todos los servidores, de forma que no es posible rastrear las mayoría de las estadísticas en una base de servidor-por-servidor, como lo hace el proveedor de la memoria caché dinámica predeterminada. A continuación, aparece una lista de las estadísticas generadas por la API de estadísticas de memoria caché y lo que representan cuando se utiliza el proveedor de memoria caché dinámica WebSphere eXtreme Scale. Del mismo modo que el proveedor predeterminado, estas estadísticas no se sincronizan y, por lo tanto, pueden variar hasta el 10% para las cargas de trabajo concurrentes.

- **Coincidencias en la memoria caché** : las coincidencias de la memoria caché se rastrean por servidor. Si el tráfico en el servidor A genera 10 coincidencias de memoria caché y el tráfico en el servidor B genera 20 coincidencias de memoria caché, las estadísticas de la memoria caché informarán de 10 coincidencias de memoria caché en el servidor A y 20 coincidencias de memoria caché en el servidor B.
- **Faltas de coincidencia de la memoria caché**: las faltas de coincidencia de la memoria caché se rastrean por servidor simplemente como las coincidencias de memoria caché.
- **Entradas de la memoria caché**: esta estadística informa del número de entradas de memoria caché en la memoria caché distribuida. Todos los servidores que acceden a la memoria caché informarán del mismo valor para esta estadística y dicho valor será el número total de entradas de memoria caché en la memoria de todos los servidores.
- **Tamaño de la memoria caché en MB**: esta métrica se admite solo para memorias caché que utilizan las topologías remota, incorporada o embedded_partitioned. Notifica el número de megabytes del espacio de almacenamiento dinámico Java consumido por la memoria caché, en toda la cuadrícula. Esta estadística notifica

el uso de almacenamiento dinámico solo para las particiones primarias; debe tener en cuenta las réplicas. Dado que el valor predeterminado de las topologías remotas y embedded_partitioned en una réplica asíncrona, doble este número para obtener el consumo real de memoria de la memoria caché.

- **Supresiones de memoria caché**: esta estadística informa del número total de entradas eliminadas de la memoria caché por un método cualquiera y es un valor de agregado para toda la memoria caché distribuida. Si el tráfico en el servidor A genera 10 invalidaciones y el tráfico en el servidor B genera 20 invalidaciones, el valor en ambos servidores será 30.
- **Supresiones de la memoria caché menos utilizada recientemente (LRU)**: esta estadística es un agregado, al igual que las supresiones de memoria caché. Rastrea el número de entradas que se eliminaron para mantener la memoria caché debajo de su tamaño máximo.
- **Invalidaciones de tiempo de espera**: también se trata de una estadística agregada y rastrea el número de entradas que se eliminaron porque excedieron el tiempo de espera.
- **Invalidaciones explícitas** : también es una estadística agregada, rastrea el número de entradas que se eliminaron con la invalidación directa mediante clave, ID de dependencia o plantilla.
- **Stats ampliados** : el proveedor de memoria caché dinámica de eXtreme Scale exporta las siguientes series de clave stat ampliada.
  - **com.ibm.websphere.xs.dynacache.remote_hits**: el número total de coincidencias de memoria caché en el contenedor eXtreme Scale. Se trata de una estadística agregada y su valor en la correlación de stats ampliados es `long`.
  - **com.ibm.websphere.xs.dynacache.remote_misses**: el número total de faltas de coincidencia de la memoria caché rastreadas en el contenedor eXtreme Scale. Una estadística agregada, su valor en la correlación de stats ampliados es `long`.

Informando de las estadísticas de restablecimiento

El proveedor de memoria caché dinámica le permite restablecer las estadísticas de memoria caché. Con el proveedor predeterminado, la operación restablecer sólo borra las estadísticas en el servidor afectado. El proveedor de memoria caché dinámica eXtreme Scale rastrea la mayoría de sus datos estadísticos en los contenedores de la memoria caché remota. Estos datos no se borran ni modifican cuando se restablecen las estadísticas. En lugar de esto, el comportamiento de la memoria caché dinámica predeterminada se simula en el cliente informando de la diferencia entre el valor actual de una estadística determinada y el valor de dicha estadística la última que se llamó a la operación restablecer en dicho servidor.

Por ejemplo, si el tráfico en el servidor A genera 10 supresiones de memoria caché, las estadísticas en el servidor A y en el servidor B informarán de 10 supresiones. Ahora, si las estadísticas en el servidor B se restablecen y el tráfico en el servidor A genera 10 supresiones adicionales, las estadísticas en el servidor A informarán de 20 supresiones y los stats en el servidor B informarán de 10 supresiones.

Sucesos de la memoria caché dinámica

La API de memoria caché dinámica permite a los usuarios registrar escuchas de sucesos. Cuando se utiliza eXtreme Scale como el proveedor de memoria caché dinámica, los escuchas de sucesos funcionan como se esperaba para las memorias caché locales.

Para las memorias caché distribuidas, el comportamiento del sucesos dependerá de la topología que se utiliza. Para las memorias caché que utilizan la topología incorporada, los sucesos se generarán en el servidor que maneja las operaciones de escritura, también conocidas como el fragmento primario. Esto significa que sólo un servidor recibirá notificaciones de suceso, pero tendrá todas las notificaciones de suceso esperadas normalmente del proveedor de memoria caché dinámica. Puesto que WebSphere eXtreme Scale elige el fragmento primario en el tiempo de ejecución, no es posible garantizar que un proceso de servidor determinado reciba siempre estos sucesos.

Las memorias caché incorporadas con particiones generarán sucesos en cualquier servidor que aloje una partición de la memoria caché. Si una memoria caché tiene 11 particiones y cada servidor de una cuadrícula de 11 servidores de WebSphere Application Server Network Deployment aloja una de estas particiones, cada servidor recibirá los sucesos de la memoria caché dinámica para las entradas de memoria caché que aloja. Ningún proceso de servidor único verá toda la información de todos los sucesos, a menos que las 11 particiones estuvieran alojadas en dicho proceso de servidor. Del mismo modo que la topología incorporada, no es posible garantizar que un proceso de servidor determinado vaya a recibir un conjunto determinado de sucesos o cualquier suceso.

Las memorias caché que utilizan la topología remota no soportan los sucesos de memoria caché dinámica.

Llamadas de MBean

El proveedor de la memoria caché dinámica WebSphere eXtreme Scale no soporta la memoria caché de disco. Cualquier llamada de MBean relacionadas con la memoria caché de disco no funcionará.

Correlación de políticas de duplicación de memoria caché dinámica

El proveedor de memoria caché dinámica incorporada de WebSphere Application Server soporta varias políticas de duplicación de memoria caché. Estas políticas se pueden configurar de forma global o en cada entrada de memoria caché. Consulte la documentación de la memoria caché dinámica si desea una descripción de de estas políticas de duplicación.

El proveedor de memoria caché dinámica eXtreme Scale no permite estas políticas directamente. Las características de duplicación de una memoria caché se determinan a través del tipo de topología distribuida de eXtreme Scale configurado y se aplica a todos los valores colocados en dicha memoria caché, independientemente de la política de duplicación establecida por el servicio de memoria caché dinámica en la entrada. A continuación, aparece una lista de todas las políticas de duplicación soportadas por el servicio de memoria caché dinámica e ilustra que topología eXtreme Scale proporciona características de duplicación similares.

Tenga en cuenta que el proveedor de memoria caché dinámica eXtreme Scale ignora los valores de la política de duplicación DRS en una memoria caché o en una entrada de memoria caché. Los usuarios deben elegir la topología que sea apropiada para sus necesidades de duplicación.

- NOT_SHARED: actualmente ninguna de las topologías proporcionadas por el proveedor de memoria caché dinámica eXtreme Scale puede aproximarse a esta

política. Esto significa que todos los datos almacenados en la memoria caché deben tener claves y valores que implementen java.io.Serializable.

- SHARED_PUSH: la topología incorporada se aproxima a esta topología de duplicación. Cuando se crea una entrada de memoria caché, se duplica en todos los servidores. Los servidores sólo buscan las entradas de memoria caché localmente. Si no se encuentra una entrada de forma local, se da por supuesto que no existe y que no se consulta a otros servidores en relación con esta.
- SHARED_PULL y SHARED_PUSH_PULL: las topologías incoporada con particiones y remota se aproximan a esta política de duplicación. El estado distribuido de la memoria caché es completamente coherente entre todos los servidores.

Esta información se proporciona principalmente para que pueda garantizar que la topología cumple con sus necesidades de coherencia distribuida. Por ejemplo, si la topología incorporada es una mejor opción para sus necesidades de despliegue y rendimiento, pero necesita el nivel de coherencia de memoria caché proporcionado por SHARED_PUSH_PULL, considere utilizar la topología incorporada con particiones, aunque el rendimiento pueda disminuir ligeramente.

Seguridad

Puede proteger las instancias de memoria caché dinámica que se ejecutan en las topologías incorporada e incorporada con particiones con las funciones de seguridad incorporadas en WebSphere Application Server. Consulte la documentación en Protección de servidores de aplicaciones en el centro de información de WebSphere Application Server.

Cuando se ejecuta una memoria caché en la topología remota, es posible para un cliente autónomo de eXtreme Scale para conectarse a la memoria caché y afecta a los contenidos de la instancia de la memoria caché dinámica. El proveedor de la memoria caché dinámica eXtreme Scale tiene una característica de cifrado de sobrecarga baja que puede impedir que los clientes no WebSphere Application Server lean o modifiquen los datos de la memoria caché. Para habilitar esta característica, establezca el parámetro opcional **com.ibm.websphere.xs.dynacache.encryption_password** en el mismo valor en todas las instancias de WebSphere Application Server que accedan al proveedor de memoria caché dinámica. Así se cifrará el valor y los metadatos de usuario para la CacheEntry que utiliza el cifrado AES de 128 bits. Es muy importante que se establezca el mismo valor en todos los servidores. Los servidores no podrán leer los datos colocados en la memoria caché por los servidores con un valor diferente para este parámetro.

Si el proveedor eXtreme Scale detecta que se han establecido distintos valores para esta variable en la misma memoria caché, genera un aviso en el registro del proceso del contenedor eXtreme Scale.

Consulte la documentación de eXtreme Scale sobre WebSphere eXtreme Scaleseguridad si es necesaria la autenticación SSL o de cliente.

Información adicional

- Redbook de memoria caché dinámica
- Documentación de la memoria caché dinámica
  - WebSphere Application Server 7.0
  - WebSphere Application Server 6.1

- Documentación de DRS
  - WebSphere Application Server 7.0
  - WebSphere Application Server 6.1

Planificación de capacidad y alta disponibilidad (almacenamiento en memoria caché dinámica)

La API de memoria caché dinámica está disponible para las aplicaciones Java EE que están desplegadas en WebSphere Application Server. Se puede sacar el máximo partido de la memoria caché dinámica para almacenar en la memoria caché los datos empresariales, el HTML generado o para sincronizar los datos de la memoria caché en la célula utilizando el servicio de duplicación de datos (DRS).

Visión general

De forma predeterminada, todas las instancias de memoria caché dinámica creadas con el proveedor de memoria caché dinámica WebSphere eXtreme Scale tienen una alta disponibilidad. El coste del nivel y de la memoria de la alta disponibilidad depende de la topología utilizada.

Cuando se utiliza la topología incorporada, el tamaño de memoria caché está limitado a la cantidad de memoria libre de un único proceso de servidor y cada proceso de servidor almacena una copia completa de la memoria caché. Mientras el proceso de servidor único se sigue ejecutando, la memoria caché sobrevive. Los datos de la memoria caché sólo se perderán si todos los servidores que acceden a la memoria caché se concluyen.

Para la memoria caché que utiliza la topología particionada incoporada, el tamaño de memoria caché está limitado a un agregado del espacio libre disponible en todos los procesos del servidor. De forma predeterminada, el proveedor de memoria caché dinámica eXtreme Scale utiliza 1 réplica para cada fragmento primario, de forma que cada conjunto de datos de la memoria caché se almacena dos veces.

Utilice la siguiente fórmula A para determinar la capacidad de una memoria caché incorporada con particiones:

Fórmula A

F * C / (1 + R) = M

Donde:
- F = memoria libre por proceso de contenedor
- C = número de contenedores
- R = número de réplicas
- M = tamaño total de la memoria caché

Para una cuadrícula de WebSphere Network Deployment que tiene 256 MB de espacio disponible en cada proceso, con 4 procesos de servidor en total, una instancia de memoria caché entre todos estos servidores podría almacenar hasta 512 megabytes de datos. En esta modalidad, la memoria caché puede sobrevivir a que se cuelgue un servidor sin perder datos. Además, se podrían concluir hasta dos servidores de forma secuencial sin perder datos. Así puede, para el ejemplo anterior, la fórmula es la siguiente:

256mb * 4 contenedores/ (1 primario + 1 réplica) = 512mb.

Las memorias caché que utilizan la topología remota tienen unas características de tamaño similares a las de las memorias caché que utilizan las particiones incorporadas, pero están limitadas por la cantidad de espacio disponible en todos los procesos de contenedor de eXtreme Scale.

En las topologías remotas, es posible aumentar el número de réplicas para proporcionar un nivel superior de disponibilidad con el coste de la sobrecarga de memoria adicional. En la mayoría de las aplicaciones de memoria caché dinámica, esto no debería ser necesario, pero puede editar el archivo dynacache-remote-deployment.xml para aumentar el número de réplicas.

Utilice las siguientes fórmulas, B y C, para determinar el efecto de añadir más réplicas en la alta disponibilidad de la memoria caché.

Fórmula B

N = Minimum(T -1, R)

Donde:
- N = el número de procesos que se pueden colgar simultáneamente
- T = el número total de contenedores
- R = el número total de réplicas

Fórmula C

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

Donde:
- T = el número total de contenedores
- N = el número total de réplicas
- m = el número mínimo de contenedores necesarios para soportar los datos de la memoria caché.

Para el ajuste de rendimiento con el proveedor de memoria caché dinámica, consulte Ajuste del proveedor de la memoria caché dinámica.

Tamaño de la memoria caché

Antes de que se pueda desplegar una aplicación que utiliza el proveedor de memoria caché dinámica WebSphere eXtreme Scale, los principales generales descritos en la sección anterior se deben combinar con los datos de entorno para los sistemas de producción. La primera figura para establecer es el número total de procesos de contenedor y la cantidad de memoria disponible en cada procesa para contener datos de memoria caché. Al utilizar la topología incorporada, los contenedores de memoria caché se volverán a colocar dentro de los procesos de WebSphere Application Server, de forma que haya un contenedor para cada servidor que comparte la memoria caché. Determinar la sobrecarga de memoria de la aplicación sin la memoria caché habilitada y WebSphere Application Server es el mejor método para descubrir la cantidad de espacio disponible en el proceso. Esto se puede realizar analizando los datos de la recogida de basura verbosa. Al utilizar la topología remota, esta información se puede encontrar consultando la salida de la recogida de basura verbosa de un contenedor autónomo iniciado recientemente,

que todavía no se haya rellenado con datos de la memoria caché. El último concepto que se debe tener en cuenta al descubrir la cantidad de espacio disponible para el proceso para los datos de memoria caché es reservar algo de espacio de almacenamiento dinámico para la recogida de basura. La sobrecarga del contenedor, WebSphere Application Server o servidor autónomo, además del tamaño reservado para la memoria caché no debe representar más del 70% del almacenamiento dinámico total.

Una vez recopilada esta información, los valores de pueden utilizar en la fórmula A, descrita previamente, para determinar el tamaño máximo para la memoria caché particionada. Una vez que se conoce el tamaño máximo, el siguiente paso es determinar el número total de entradas de la memoria caché que se pueden soportar, que requiere que se determine el tamaño medio por entrada de memoria caché. El método sencillo par hacer esto es añadir un 10% al tamaño del objeto del cliente. Consulte la guía de ajusta para la memoria caché dinámica y el servicio de duplicación de datos si desea una información más detallada sobre los tamaños de las entradas de la memoria caché dinámica cuando se utiliza la memoria caché dinámica.

Cuando está habilitada la compresión, afecta al tamaño del objeto del cliente, no a la sobrecarga del sistemas de colocación en la memoria caché. Utilice la siguiente fórmula para determinar el tamaño de un objeto guardado en la memoria caché cuando utilice la compresión:

S = O * C + O * 0.10

Donde:

- S = tamaño medio del objeto almacenado en memoria caché
- O = tamaño medio de un objeto de cliente no comprimido
- C = proporción de compresión expresada como una fracción.

Así, una proporción de comprensión de 2 a 1 es 1/2 = 0,50. Para este valor es mejor valores pequeños. Si el objeto que se almacena es un POJO normal, básicamente lleno de tipos primitivos, presuponga una proporción de compresión de 0,60 a 0,70. Si el objeto almacenado en memoria caché es un Servlet, JSP, o un objeto WebServices, el método óptimo para determinar la proporción de comprensión es comprimir un ejemplo representativo con un programa de utilidad de comprensión ZIP. Si esto no es posible, una proporción de compresión de 0,2 a 0,35 es común para este tipo de datos.

A continuación, utilice esta información para determinar el número total de entradas de memoria caché que se pueden soportar. Utilice la siguiente fórmula D:

Fórmula D

T = S / A

Donde:

- T= número total de entradas de la memoria caché
- S = tamaño total disponible para los datos de la memoria caché calculados utilizando la fórmula A
- A = tamaño medio de cada entrada de la memoria caché

Finalmente, debe establecer el tamaño de memoria caché en la instancia de la memoria caché dinámica para aplicar este límite. El proveedor de la memoria

caché dinámica WebSphere eXtreme Scale difiere del proveedor de la memoria caché dinámica en este aspecto. Utilice la siguiente fórmula para determinar el valor que se debe establecer para el tamaño de memoria caché en la instancia de la memoria caché dinámica. Utilice la siguiente fórmula E:

Fórmula E

Cs = Ts / Np

Donde:

- Ts = tamaño total de la memoria caché
- Cs = valor del tamaño de memoria caché para establecer en la instancia de la memoria caché dinámica
- Np = número de particiones. El valor predeterminado es 47.

Establezca el tamaño de la instancia de memoria caché dinámica en un valor calculado por la fórmula E en cada servidor que comparta la instancia de memoria caché.

Capítulo 4. Visión general de conceptos de escalabilidad

La escalabilidad permite que los datos de un despliegue de WebSphere eXtreme Scale se distribuyan en un conjunto de servidores (contenedores), basándose en la opción de configuración.

Escalabilidad

WebSphere eXtreme Scale se puede escalar a través del uso de datos particiones y puede escalar miles de contenedores si es necesario porque cada contenedor es independiente de otros contenedores.

WebSphere eXtreme Scale divide los datos en distintas particiones que se pueden mover entre procesos o incluso entre máquinas durante la ejecución. Puede, por ejemplo, empezar con un despliegue de cuatro servidores y, a continuación, ampliar a un despliegue con diez servidores a medida que crece la demanda en la memoria caché. Simplemente, como puede añadir más máquinas físicas y unidades de proceso para la escalabilidad vertical, puede ampliar la capacidad de escalabilidad elástica de eXtreme Scale de forma horizontal con el particionamiento. Esta es otra diferencia principal con bases de datos en memoria (IMDB) respecto a eXtreme Scale (que es una cuadrícula de datos), puesto que las IMDB sólo se pueden escalar de forma vertical.

Con WebSphere eXtreme Scale, también puede utilizar un conjunto de API para obtener acceso transaccional a estos datos particionados y distribuidos de forma opcional. En términos de rendimiento, las selecciones que realice para interactuar con la memoria caché son tan importantes como las funciones para gestionar la memoria caché para la disponibilidad.

Nota: La escalabilidad no está disponible cuando los contenedores se comunican entre sí. El protocolo de la gestión de la disponibilidad, o de la agrupación principal, es un algoritmos de vista y pulsación $O(N^2)$, pero se mitiga manteniendo el número de miembros de principal a 20. Sólo existe la réplica de igual a igual entre fragmentos.

Clientes distribuidos

El protocolo del cliente de WebSphere eXtreme Scale soporta una gran cantidad de clientes. La estrategia de particionamiento ofrece asistencia dando por supuesto que todos los clientes no siempre están interesado en todas las particiones porque las conexiones pueden esparcirse por varios contenedores. Los clientes se conectan directamente a las particiones, por lo que la latencia se limita a una conexión transferida.

Cuadrículas, particiones y fragmentos

Una cuadrícula distribuida de eXtreme Scale se divide en particiones. Una partición contiene un subconjunto exclusivo de los datos. Una partición consta de uno o más fragmentos: un fragmento primario y fragmentos de réplica. No es necesario que tenga fragmentos en una partición, pero los fragmentos de réplica proporcionan alta disponibilidad. Si el despliegue es una cuadrícula de datos de

memoria independiente o un espacio de proceso de base de datos de memoria, el acceso de datos en eXtreme Scale se basa en gran medida en los conceptos de la creación de fragmentos.

Los datos de una partición se almacenan en tiempo de ejecución en un conjunto de fragmentos. Este conjunto de fragmentos incluye un fragmento primario y posiblemente uno más fragmentos réplica. Un fragmento es la unidad más pequeña que eXtreme Scale puede añadir a una Máquina virtual Java o eliminar de ésta.

Existen dos estrategias de colocación: FIXED_PARTITIONS (valor predeterminado) y PER_CONTAINER. El tema siguiente se centra en el uso de la estrategia FIXED_PARTITIONS.

Número de fragmentos

Si su entorno incluyese diez particiones que contuvieran un millón de objetos sin ninguna réplica, existirían diez fragmentos con 100.000 objetos cada uno. Si añadiera una réplica a este escenario, existiría un fragmento adicional en cada partición. En este caso, existirían 20 fragmentos, 10 fragmentos primarios y 10 fragmentos réplica. De nuevo, cada uno de estos fragmentos contendría 100.000 objetos. Cada partición se compone de un fragmento primario y uno o más (N) fragmentos réplica. La determinación del número óptimo de fragmentos es clave. Si configura un número pequeño de fragmentos, los datos no se distribuyen uniformemente entre los fragmentos, lo que provocará errores de falta de memoria y problemas de sobrecarga del procesador. Debe tener al menos 10 fragmentos para cada JVM al realizar la escala. Al desplegar inicialmente la cuadrícula, debería utilizar posiblemente un número grande de particiones.

Número de fragmentos por JVM

Escenario: número pequeño de fragmentos para cada JVM

Los datos se añaden a una JVM y se eliminan de ésta mediante unidades de fragmentos. Los fragmentos nunca se dividen en partes. Si existen 10 GB de datos y existen 20 fragmentos para alojar estos datos, cada fragmento incluye de media 500 MB de datos. Si hay 9 Mäquinas virtuales Java que alojan la cuadrícula, de media cada JVM tiene dos fragmentos. Como 20 no es divisible entre 9, unas cuantas Mäquinas virtuales Java tienen tres fragmentos, con la distribución siguiente:

- 7 Mäquinas virtuales Java con 2 fragmentos
- 2 Mäquinas virtuales Java con 3 fragmentos

Puesto que cada fragmento incluye 500 MB de datos, la distribución de datos no es uniforme. Las 7 Mäquinas virtuales Java con 2 fragmentos contienen cada uno 1 GB de datos. Las 2 Mäquinas virtuales Java con 3 fragmentos tienen el 50% más de datos, o 1,5 GB, que es una carga de memoria mucho mayor. Como estas 2 Mäquinas virtuales Java contienen 3 fragmentos, reciben el 50% más de peticiones de sus datos. Como resultado, un número pequeño de fragmentos para cada JVM causa desequilibrio. Para aumentar el rendimiento, debe incrementar el número de fragmentos para cada JVM .

Escenario: número mayor de fragmentos por JVM

Para este escenario el número de fragmentos es mucho mayor. En este escenario hay 101 fragmentos con 9 Mäquinas virtuales Java que contienen 10 GB de datos. En este caso, cada fragmento contiene 99 MB de datos. La distribución de fragmentos de las Mäquinas virtuales Java es la siguiente:

- 7 Mäquinas virtuales Java con 11 fragmentos
- 2 Mäquinas virtuales Java con 12 fragmentos

Las 2 Mäquinas virtuales Java con 12 fragmentos tienen sólo 99 MB más de datos que los otros fragmentos, que es una diferencia del 9%. Este escenario se distribuye de manera más uniforme que la diferencia del 50% del escenario con un número pequeño de fragmentos. Desde la perspectiva de uso del procesador, sólo existe el 9% más de trabajo para las 2 Mäquinas virtuales Java con los 12 fragmentos en comparación con las 7 Mäquinas virtuales Java que tienen 11 fragmentos. Al incrementar el número de fragmentos en cada JVM , los datos y el uso del procesador se distribuyen de manera más equilibrada y uniforme.

Al crear el sistema, use 10 fragmentos para cada JVM como escenario de tamaño máximo, o cuando el sistema está ejecutando su número máximo de Mäquinas virtuales Java en el horizonte de planificación.

Factores adicionales de ubicación

El número de particiones, la estrategia de ubicación y el número y tipo de réplicas se establecen en la política de despliegue. El número de fragmentos colocados dependerá de la política de despliegue definida. numInitialContainers, minSyncReplicas, developmentMode, maxSyncReplicas y maxAsyncReplicas afectarán a dónde y cuándo se pueden colocar las particiones y las réplicas. Si el inicio del servidor inicial no permite que se coloquen maxSyncReplicas y maxAsyncReplicas, debe tener réplicas adicionales colocadas si inicia más servidores más adelante. Al planificar el número de fragmentos por JVM, el número máximo de fragmentos, incluidas las réplicas, será dependiente de tener suficientes JVM iniciadas para admitir el número máximo de réplicas solicitadas. Una réplica nunca se coloca en el mismo proceso que su fragmento primario porque si el proceso se pierde, se perderían tanto la réplica como el fragmento primario. Si developmentMode es false, el fragmento primario y las réplicas no se colocarán en la misma máquina.

Particionamiento

Utilice el particionamiento para almacenar grandes cantidades de datos en la máquina virtual Java (JVM). Para particionar los datos, utilice un esquema especificado por la aplicación para dividir los datos. Con WebSphere eXtreme Scale, el particionamiento aumenta la escalabilidad y la disponibilidad.

El particionamiento es el mecanismo que utiliza WebSphere eXtreme Scale uses para escalar una aplicación. El particionamiento es la separación del estado de la aplicación en partes donde cada una contiene algún conjunto de datos completo de la instancia. El particionamiento no es como la escritura en bandas de RAID (Redundant Array of Independent Disks), que corta cada instancia a lo largo de todas las bandas. Cada partición aloja los datos completos para las entradas individuales. El particionamiento es un medio muy eficaz para escalar, pero no puede aplicarse a todas las aplicaciones. Las aplicaciones que requieren garantías transaccionales a través de grandes conjuntos de datos no se escalan y no se pueden particionar de forma eficaz, así que, actualmente, eXtreme Scale no soporta la confirmación de dos fases entre particiones.

Importante: Seleccione el número de particiones con atención. El número de particiones definidas en la política de despliegue afecta directamente al número de contenedores con los que se puede escalar una aplicación. Cada partición está compuesta de un fragmento primario y el número configurado de fragmentos de réplica. La fórmula `(Número_Particiones*(1 + Número_Réplicas))` es el número de contenedores que se pueden utilizar para escalar de forma horizontal una única aplicación.

Uso de particiones

Una cuadrícula puede tener muchas particiones, o miles, si fuera preciso. Una cuadrícula puede aumentar (escalar) el producto del número de particiones multiplicado por el número de fragmentos por partición. Por ejemplo, si tiene 16 particiones y cada partición tiene un primario y una réplica, o dos fragmentos, podrá escalarla de forma potencial a 32 Máquinas virtuales Java. En este caso, se define un fragmento para cada JVM. Debe elegir un número razonable de particiones basándose en el número esperado de Máquinas virtuales Java que probablemente vaya a utilizar. Cada fragmento aumenta el uso de procesador y memoria para el sistema. El sistema se ha diseñado para escalar de forma horizontal para manejar esta sobrecarga según el número disponible de Máquinas virtuales Java de servidor.

las aplicaciones no deben utilizar miles de particiones, si la aplicación se ejecuta en una cuadrícula de cuatro Máquinas virtuales Java de contenedor. La aplicación debe configurarse con un número razonable de fragmentos para cada JVM de contenedor. Por ejemplo, una configuración no razonable sería establecer 2000 particiones con dos fragmentos que se ejecutan en cuatro Máquinas virtuales Java de contenedor. Esta configuración genera 4000 fragmentos que se colocan en cuatro Máquinas virtuales Java de contenedor o 1000 fragmentos por JVM de contenedor.

Por el contrario, una mejor configuración sería tener menos de 10 fragmentos para cada JVM de contenedor esperado. Esta configuración todavía proporciona la posibilidad de permitir una escalada elástica que es diez veces la configuración inicial, mientras que se mantiene un número razonable de fragmentos por JVM de contenedor.

Considere este ejemplo de escalada: actualmente tiene seis sistemas con dos Máquinas virtuales Java de contenedor por sistema. Espera crecer a 20 sistemas en los próximos tres años. Con 20 sistemas, tiene 40 Máquinas virtuales Java de contenedor y elige 60 por razones pesimistas. Desea contar con 4 fragmentos por JVM de contenedor. Potencialmente tiene 60 contenedores o un total de 240 fragmentos. Si tiene un fragmento primario y una réplica por partición, tendrá 120 particiones. Este ejemplo le proporciona 240 dividido por 12 Máquinas virtuales Java de contenedor, o 20 fragmentos por JVM de contenedor para el despliegue inicial con el potencial de poder escalar de forma horizontal a 20 máquinas, más adelante.

ObjectMap y particionamiento

Con la estrategia de colocación FIXED_PARTITION predeterminada, las correlaciones se dividen entre las particiones y las claves realizan un método hash en particiones diferentes. No es necesario que el cliente sepa a qué partición pertenecen las claves. Si un mapSet tiene varias correlaciones, las correlaciones se deben confirmar en transacciones distintas.

Entidades y particionamiento

Las entidades del gestor de entidades tienen una optimización que ayuda a los clientes que trabajan con entidades en un servidor. El esquema de entidad en el servidor relativo al conjunto de correlaciones puede especificar una sola entidad raíz. El cliente debe acceder a todas las entidades a través de la entidad raíz. El gestor de entidades puede encontrar las entidades relacionadas en dicha raíz en la misma partición sin necesitar que las correlaciones relacionadas tengan una clave común. La entidad raíz establece afinidad con una única partición. Esta partición se utiliza en todas las búsquedas de entidad dentro de la transacción una vez establecida la afinidad. Esta afinidad puede ahorrar memoria porque las correlaciones relacionadas no necesitan una clave común. La entidad raíz debe especificarse con una anotación de entidad modificada, como se muestra en el ejemplo siguiente:

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

Utilice la entidad para encontrar la raíz del gráfico del objeto. El gráfico del objeto define la relación entre una o varias entidades. Cada entidad enlazada se debe resolver con la misma partición. Se presupone que todas las entidades hija están en la misma partición que la raíz. Sólo se puede acceder a las entidades hija de este gráfico del objeto desde un cliente de la entidad raíz. Las entidades raíz siempre son necesarias en entornos con particiones si se utiliza un cliente eXtreme Scale para comunicarse con el servidor. Sólo se puede definir un tipo de entidad raíz por cliente. Las entidades raíz no son necesarias cuando se utilizan objetos ObjectGrid de estilo Extreme Transaction Processing (XTP), ya que toda la comunicación con la partición se establece a través de acceso local, directo y no a través del mecanismo de cliente y servidor.

Colocación y particiones

Hay dos estrategias de colocación disponibles para WebSphere eXtreme Scale: partición fija y por contenedor. La elección de la estrategia de colocación afecta al modo en que la configuración de despliegue coloca las particiones en la cuadrícula remota.

Colocación de partición fija

Puede establecer la estrategia de colocación en el archivo XML de la política de despliegue. La estrategia de colocación predeterminada es la ubicación de partición fija, habilitada con el valor `FIXED_PARTITION`. El número de fragmentos primarios que se colocan entre los contenedores disponibles es igual al número de particiones que ha configurado con el elemento numberOfPartitions. Si ha configurado réplicas, el número mínimo total de fragmentos colocados se define a través de la siguiente fórmula: `((1 fragmento primario + fragmentos mínimos síncronos) * particiones definidas)`. El número máximo total de fragmentos colocados se define a través de la siguiente fórmula: `((1 fragmento primario + máximo de fragmentos síncronos + máximo de fragmentos asíncronos) * particiones)`. El despliegue de WebSphere eXtreme Scale se distribuye entre estos fragmentos sobre los contenedores disponibles. Las claves de cada correlación pasan por el código hash en particiones asignadas basándose en las particiones totales que ha definido. Las claves realizan el código hash en la misma partición, aunque la partición se mueva debido a una migración tras error o a cambios de servidor.

Por ejemplo, si el valor de numberPartitions es 6 y el valor de minSync es 1 para MapSet1, el número total de fragmentos para dicho conjunto de correlaciones es 12, porque cada una de las 6 particiones requiere una réplica síncrona. Si se inician

tres contenedores, WebSphere eXtreme Scale coloca cuatro fragmentos por contenedor paraMapSet1.

Colocación por contenedor

La estrategia de colocación alternativa es la colocación por contenedor, que está habilitada con el valor `PER_CONTAINER` para placementStrategy en el elemento del conjunto de correlaciones en el archivo XML de despliegue. Con esta estrategia, el número de fragmentos primarios colocados en cada contenedor nuevo es igual al número de particiones, *P*, que ha configurado. El entorno de despliegue de WebSphere eXtreme Scale coloca *P* réplicas de cada partición para cada contenedor restante. El valor numInitialContainers se ignora cuando se utiliza la colocación por contenedor. Las particiones cada vez son más grandes a medida que crecen los contenedores. Las claves para las correlaciones no son fijas para una determinada partición de esta estrategia. El cliente se direcciona a una partición y utiliza el primario aleatorio. Si un cliente desea volver a conectarse a la misma sesión que se ha utilizado para volver a encontrar la clave, debe utilizar un descriptor de contexto de sesión.

Si desea más información, consulte el tema sobre cómo utilizar un SessionHandle para el direccionamiento en la *Guía de programación*.

Para los servidores detenidos o con migración tras error, el entorno de WebSphere eXtreme Scale traslada los fragmentos primarios en la estrategia de colocación por contenedor, si todavía contienen datos. Si los fragmentos están vacíos, se descartan. En la estrategia por contenedor, los fragmentos primarios antiguos no se conservan porque se colocan nuevos fragmentos primarios para cada contenedor.

WebSphere eXtreme Scale permite la colocación por contenedor como alternativa a lo que se podría denominar la estrategia de colocación "típica", un enfoque de partición fija con la clave de una correlación con el método hash aplicado a una de dichas particiones. En un caso por contenedor (que se define mediante PER_CONTAINER), el despliegue coloca las particiones en el conjunto de servidores de contenedor en línea y las escala automáticamente tanto para ampliarlas como para reducirlas a medida que se añaden o se eliminan contenedores de la cuadrícula de servidor. Una cuadrícula con el enfoque de partición fija funciona bien para las cuadrículas basadas en claves, donde la aplicación utiliza un objeto de clave para ubicar datos en la cuadrícula. A continuación se explica la alternativa.

Ejemplo de cuadrícula por contenedor

Las cuadrículas PER_CONTAINER son diferentes. Para especificar que la cuadrícula debe utilizar PER_CONTAINER, se debe utilizar el atributo placementPolicy del archivo XML de despliegue. En lugar de configurar cuántas particiones desea tener en total en la cuadrícula, debe especificar cuántas particiones desea por cada contenedor que inicie.

Por ejemplo, si define el número de particiones por contenedor de modo que sean 5, cuando inicie posteriormente un contenedor, eXtreme Scale crea 5 primarios de partición anónimos nuevos en dicho contenedor y crea las réplicas necesarias en los otros contenedores ya desplegados.

A continuación se muestra una secuencia potencial de entorno por contenedor a medida que la cuadrícula crece.

1. Iniciar el contenedor C0 que aloja 5 primarios (P0 - P4).

- C0 aloja: P0, P1, P2, P3, P4.

2. Iniciar el contenedor C1 que aloja 5 primarios más (P5 - P9). Las réplicas se equilibran en los contenedores.
   - C0 aloja: P0, P1, P2, P3, P4, R5, R6, R7, R8, R9.
   - C1 aloja: P5, P6, P7, P8, P9, R0, R1, R2, R3, R4.
3. Iniciar el contenedor C2 que aloja 5 primarios más (P10 - P14). Las réplicas se siguen equilibrando.
   - C0 aloja: P0, P1, P2, P3, P4, R7, R8, R9, R10, R11, R12.
   - C1 aloja: P5, P6, P7, P8, P9, R2, R3, R4, R13, R14.
   - C2 aloja: P10, P11, P12, P13, P14, R5, R6, R0, R1.

El patrón continúa a medida que se van iniciando más contenedores, creando 5 particiones primarias nuevas cada vez y volviendo a equilibrar las réplicas en los contenedores disponibles en la cuadrícula.

Nota: WebSphere eXtreme Scale no mueve los primarios al utilizar la estrategia PER_CONTAINER, sólo las réplicas.

Recuerde que los números de partición son arbitrarios y no tienen nada que ver con las claves, por lo que no se puede utilizar el direccionamiento basado en claves. Si un contenedor se detiene, los ID de partición creados para dicho contenedor dejan de utilizarse, de modo que queda un vacío entre los ID de partición. En el ejemplo, ya no habría particiones P5 - P9 si el contenedor C2 fallase, quedando sólo P0 - P4 y P10 - P14, por lo que el hash basado en claves resulta imposible.

El uso de número como 5, o incluso más probablemente 10, para indicar el número de particiones por contenedor funciona mejor si se tienen en cuenta las consecuencias del fallo de un contenedor. Para distribuir la carga de alojar los fragmentos equitativamente en la cuadrícula, necesita más de una sola partición para cada contenedor. Si tiene una sola partición por contenedor, cuando un contenedor falle, un solo contenedor (el que aloja el fragmento de réplica correspondiente) deberá soportar la carga total del primario perdido. En este caso, la carga se dobla inmediatamente para el contenedor. Sin embargo, si tiene 5 particiones por contenedor, entonces 5 contenedores asumirán la carga del contenedor perdido, disminuyendo el impacto sobre cada uno en un 80%. El uso de varias particiones por contenedor suele disminuir el impacto potencial sobre cada contenedor considerablemente. De forma más directa, considere un caso en el que un contenedor aumenta su carga de trabajo de improviso; la carga de réplica de dicho contenedor se distribuye en 5 contenedores en lugar de hacerlo en sólo uno.

Uso de la política por contenedor

Varios escenarios hacen que la estrategia por contenedor sea una configuración ideal, como por ejemplo la réplica de sesiones HTTP o el estado de sesión de aplicaciones. En tal caso, un direccionador HTTP asigna una sesión a un contenedor de servlet. El contenedor de servlet tiene que crear una sesión HTTP y elige uno de los 5 primarios de partición locales para la sesión. El "ID" de la partición elegida se almacena entonces en una cookie. El contenedor de servlet tiene ahora acceso local al estado de la sesión, lo que significa un acceso de latencia cero a los datos correspondientes a esta solicitud siempre que se mantenga la afinidad de sesiones. Y eXtreme Scale replica los cambios de la partición.

En la práctica, recuerde las repercusiones de un caso en el que tenga varias particiones por contenedor (por ejemplo, 5 otra vez). Naturalmente, con cada contenedor nuevo que se inicie, tiene 5 primarios de partición más y 5 réplicas más. Con el tiempo, se deben crear más particiones y no se deben mover ni destruir. Pero los contenedores no se comportarían realmente de este modo. Cuando un contenedor se inicia, aloja 5 fragmentos primarios, que se pueden denominar ser primarios de "inicio", existentes en los contenedores respectivos que los crearon. Si el contenedor falla, las réplicas se convierten en primarios y eXtreme Scale crea 5 réplicas más para mantener la alta disponibilidad (a menos que se haya inhabilitado la reparación automática). Los nuevos nuevas primarios están en un contenedor diferente del que los creó y pueden denominarse primarios "externos". La aplicación nunca debe colocar un estado o sesiones nuevos en un primario externo. Al final, el primario externo no tiene ninguna entrada y eXtreme Scale lo suprime automáticamente y también suprime sus réplicas asociadas. El objetivo de los primarios externos es permitir que las sesiones existentes sigan estando disponibles (pero no las sesiones nuevas).

Un cliente puede seguir interactuando con una cuadrícula que no dependa de claves. El cliente sólo inicia una transacción y almacena datos en la cuadrícula independiente de cualquier clave. Solicita a la sesión un objeto SessionHandle, un descriptor de contexto serializable que permite al cliente interactuar con la misma partición cuando sea necesario. Si desea más información, consulte el tema sobre cómo utilizar un SessionHandle para el direccionamiento en la *Guía de programación*.WebSphere eXtreme Scale elige una partición para el cliente de la lista de primarios de partición iniciales. No devuelve una partición primaria externa. El SessionHandle se puede serializar en una cookie HTTP, por ejemplo, y más tarde se puede volver a convertir la cookie en un SessionHandle. Luego, las API de WebSphere eXtreme Scale pueden volver a obtener una sesión enlazada a la misma partición utilizando el SessionHandle.

Nota: No se pueden utilizar agentes para interactuar con una cuadrícula PER_CONTAINER.

Ventajas

La descripción anterior es diferente de una cuadrícula FIXED_PARTITION o de hash normal porque el cliente por contenedor almacena datos en un lugar de la cuadrícula, obtiene un descriptor de contexto a los datos y utiliza el descriptor de contexto para volver a acceder a ellos. No hay ninguna clave proporcionada por la aplicación, al contrario que en el caso de partición fija.

El despliegue no crea una partición nueva para cada sesión. Por lo tanto, en un despliegue por contenedor, las claves utilizadas para almacenar datos en la partición deben ser exclusivas en dicha partición. Por ejemplo, puede hacer que su cliente genere un SessionID exclusivo y luego lo utilice como clave para encontrar información en correlaciones en dicha partición. Luego, varias sesiones de cliente interaccionan con la misma partición, por lo que la aplicación tiene que utilizar claves exclusivas para almacenar datos de sesión en cada partición concreta.

Los ejemplos anteriores utilizaban 5 particiones, pero se puede utilizar el parámetro numberOfPartitions del archivo XML para especificar las particiones del modo necesario. En lugar de por cuadrícula, el valor es por contenedor. (El número de réplicas se especifica del mismo modo que en la política de partición fija).

La política por contenedor también se puede utilizar con varias zonas. Si es posible, eXtreme Scale devuelve un SessionHandle a una partición cuyo primario

está ubicado en la misma zona que el cliente en cuestión. El cliente puede especificar la zona como un parámetro al contenedor o utilizando una API. El ID de zona del cliente se puede definir utilizando `serverproperties` o `clientproperties`.

La estrategia PER_CONTAINER para una cuadrícula es adecuada para las aplicaciones que almacenan el estado de tipo conversacional en lugar de datos orientados a bases de datos. La clave para acceder a los datos sería un ID de conversación y no está relacionada con un registro de base de datos específico. Proporciona un rendimiento superior (porque los primarios de partición pueden compartir ubicación con los servlets por ejemplo) y facilita la configuración (no es necesario calcular particiones y contenedores).

Transacciones de partición única y de partición entre cuadrícula

La diferencia principal entre WebSphere eXtreme Scale y las soluciones de almacenamiento de datos tradicionales como las bases de datos relacionales o las bases de datos en memoria es el uso del particionamiento, que permite a la memoria caché realizar las escaladas de forma lineal. Los tipos importantes de transacciones para tener en cuenta son las transacciones de partición única y las muchas particiones (entre cuadrícula).

En general, las interacciones con la memoria caché se pueden categorizar como transacciones de partición única o transacciones entre cuadrícula, tal como se describe a continuación.

Transacciones de partición única

Las transacciones de partición única son el método preferible para interactuar con las memorias caché alojadas por WebSphere eXtreme Scale. Cuando una transacción está limitada a una única partición, de forma predeterminada, está limitada a una única Máquina virtual Java y, por lo tanto, un único sistema de servidor. Un servidor puede completar *M* número de estas transacciones por segundo y si tiene *N* sistemas, puede completar `M*N` transacciones por segundo. Si el negocio aumenta y debe doblar el rendimiento respecto a muchas de estas transacciones por segundo, puede doblar el valor *N* comprando más sistemas. Puede cumplir las demandas de capacidad sin modificar la aplicación, actualizar el hardware o, incluso, colocando la aplicación fuera de línea.

Además de permitir a la memoria caché realizar escaladas de forma significativa, las transacciones de partición única también maximizan la disponibilidad de la memoria caché. Cada transacción sólo depende de un sistema. Cualquiera de los otros `(N-1)` sistemas puede falla sin que esto afecte al éxito o al tiempo de respuesta de la transacción. Por lo tanto, si ejecuta 100 sistemas y uno de ellas falla, sólo el 1 por ciento de las transacciones en curso en el momento en que falla el servidor se retrotrae. Después de que el servidor falle, WebSphere eXtreme Scale reubica las particiones alojadas por el servidor anómalo en los otros 99 sistemas. Durante este breve periodo, antes de que se complete la operación, los otros 99 sistemas pueden seguir completando transacciones. Sólo las transacciones que podrían implicar que las particiones que se están reubicando se bloqueen. Después de que se complete el proceso de migración tras error, la memoria caché puede seguir ejecutándose, plenamente operativa a un 99 por ciento de su capacidad de rendimiento original. Después de que se sustituya el servidor anómalo y se devuelva a la cuadrícula, la memoria caché vuelve al 100 por ciento de capacidad de rendimiento.

Transacciones entre cuadrícula

En términos de rendimiento, disponibilidad y escalabilidad, las transacciones entre cuadrícula son el opuesto de las transacciones de partición única. Las transacciones entre cuadrícula acceden a cada partición y, por lo tanto, a todos los sistemas de la configuración. Se solicita a cada sistema de la cuadrícula que busque algunos datos y, a continuación, devuelva el resultado. La transacción no se puede completar hasta que hayan contestado todos los sistemas y, por lo tanto, el rendimiento de toda la cuadrícula está limitado al sistemas más lento. Añadir sistemas no hace que el sistema más lento sea más rápido y, por lo tanto, no mejora el rendimiento de la memoria caché.

Las transacciones entre cuadrícula tienen un efecto similar en la disponibilidad. Ampliando el ejemplo anterior, si ejecuta 100 servidores y uno falla, el 100 por ciento de las transacciones que están en curso en el momento en el que falló el servidor se retrotraen. Después de que falle el servidor, WebSphere eXtreme Scale empieza a reubicar las particiones alojadas por dicho servidor a los otros 99 sistemas. Durante este tiempo, antes de que se complete el proceso de migración tras error, la cuadrícula no puede procesar ninguna de estas transacciones. Después de que se complete el proceso de migración tras error, la memoria caché puede seguir ejecutándose, pero a una capacidad reducida. Si cada sistema de la cuadrícula presta servicio a 10 particiones, 10 de los 99 sistemas restantes reciben, como mínimo, una partición adicional como parte del proceso de migración tras error. Añadir una partición adicional aumentar la carga de trabajo de dicho sistema en un 10 por ciento, como mínimo. Puesto que el rendimiento de la cuadrícula está limitado al rendimiento del sistema más lento en una transacción entre cuadrícula, de promedio, el rendimiento se reduce en un 10 por ciento.

Las transacciones de partición única son preferibles que las transacciones entre cuadrícula para escalar de forma horizontal con una memoria caché de objeto distribuida y con alta disponibilidad como WebSphere eXtreme Scale. Maximizar el rendimiento de estos tipos de sistemas requiere el uso de técnicas que son diferentes a las metodologías relacionales tradicionales, pero puede convertir las transacciones entre cuadrícula en transacciones de partición única.

Procedimientos recomendados para crear modelos de datos escalables

Los procedimientos recomendados para crear aplicaciones escalables con productos como WebSphere eXtreme Scale incluyen dos categorías: los principios fundacionales y las sugerencias de implementación. Los principios fundacionales son ideas principales que se deben capturar en el diseño de los propios datos. Una aplicación que no observa estos principios probablemente no realizará bien las escaladas, incluso para sus transacciones principales. Se aplican las sugerencias de implementación para las transacciones problemáticas en una aplicación bien diseñada de otra forma que observa los principios generales para los modelos de datos escalables.

Principios fundacionales

Algunos de los métodos importantes para optimizar la escalabilidad son conceptos o principios básicos que se deben tener en cuenta.

Duplicar en lugar de normalizar

El concepto clave para recordar sobre los productos como WebSphere eXtreme Scale es que se han diseñado para distribuir los datos entre un

gran número de sistemas. Si el objetivo es completar la mayoría o todas las transacciones en una única partición, el diseño del modelo de datos debe garantizar que todos los datos que podría necesitar la transacción se encuentran en la partición. La mayoría del tiempo, la única forma de conseguir esto es duplicando los datos.

Por ejemplo, considere una aplicación como un tablón de mensajes. Dos transacciones muy importantes para un tablón de mensajes son mostrar todas las publicaciones de un usuario proporcionado y todas las publicaciones sobre un tema determinado. En primer lugar, considere cómo estas transacciones funcionarían con un modelo de datos normalizado que contiene un registro de usuarios, un registro de temas y un registro de publicaciones que contiene el texto real. Si las publicaciones se particionan con registros de usuarios, la visualización del tema pasa a ser una transacción entre cuadrícula y viceversa. Los temas y los registros no se pueden particionar juntos porque tienen una relación de muchos a muchos.

El mejor método para realizar esta escalada del tablón de mensajes es duplicar las publicaciones, almacenando una copia con el registro de temas y una copia con el registro de usuarios. A continuación, la visualización de las publicaciones de un usuario es una transacción de partición única, la visualización de las publicaciones sobre un tema es una transacción de partición única y la actualización o la supresión de una publicación es una transacción de dos particiones. Todas estas tres transacciones realizarán las escaladas de forma lineal, ya que aumenta el número de sistemas de la cuadrícula.

Escalabilidad en lugar de recursos

El mayor obstáculo para superar cuando se considera eliminar la normalización de los modelos de datos es el impacto que estos modelos tendrían en los recursos. Podría parecer que conservar dos, tres o más copias de algunos datos utiliza demasiados recursos para que sea práctico. Cuando lo confronta con este escenario, recuerde los siguientes hechos: los recursos de hardware son más baratos cada año. En segundo lugar, y más importante, WebSphere eXtreme Scale elimina los costes más ocultos asociados al despliegue de más recursos.

Medir los recursos en términos de coste, en lugar de en términos de sistema como, por ejemplo, megabytes y procesadores. Generalmente, los almacenes de datos que funcionan con datos relacionales normalizados deben estar situados en el mismo sistema. Esta ubicación necesaria significa que se debe adquirir un único gran sistema empresarial, en lugar de varios sistemas pequeños. Con el hardware de empresa, no es raro que un sistema capaz de completar un millón de transacciones por segundo cueste muchos más que el coste combinado de 10 sistemas capaces de realizar 100.000 transacciones por segundos cada uno.

También existe un coste empresarial en la adición de recursos. Una negocio creciente acaba por agotar la capacidad. Cuando se agota la capacidad, debe concluir mientras se traslada a un sistema mayor y más rápido, o bien crear un segundo entorno de producción al que se puede pasar. De cualquier modo, los costes adicionales vendrán en forma de pérdidas de negocio o en el mantenimiento de casi el doble de la capacidad necesaria durante el periodo de transacción.

Con WebSphere eXtreme Scale, no es necesario concluir la aplicación para añadir capacidad. Si su empresa planea que necesita un 10 por ciento más capacidad para el año que viene, aumente el número de sistemas de la

cuadrícula en un 10 por ciento. Puede aumentar este porcentaje sin tiempo de inactividad de la aplicación y sin adquirir excesiva capacidad.

Evitar transformaciones de datos

Cuando se utiliza WebSphere eXtreme Scale, los datos se deben almacenar en un formato que pueda consumir directamente la lógica empresarial. Desglosar los datos en un formato más primitivo es costoso. La transformación se debe realizar cuando los datos se escriben y cuando los datos se leen. Con las bases de datos relacionales, esta transformación se realiza por necesidad, porque los datos se persisten de forma última en el disco con bastante frecuencia, pero con WebSphere eXtreme Scale, no es necesario que realice estas transformaciones. Para la mayoría de las partes, los datos se almacenan en la memoria y, por lo tanto, se almacenan en el formato exacto que necesita la aplicación.

Observar esta regla simple le ayuda a eliminar la normalización de los datos de acuerdo con el primer principio. El tipo más común de transformación para los datos empresariales es las operaciones JOIN que son necesarias para convertir los datos normalizados en un conjunto de resultados que se ajuste a las necesidades de la aplicación. Almacenar los datos en el formato correcto impide de forma implícita realizar estas operaciones JOIN y genera un modelo de datos no normalizados.

Eliminar consultas no enlazadas

Independientemente de cómo se estructuren los datos, las consultas no enlazadas no se escalan bien. Por ejemplo, no se tiene una transacción que solicite una lista de todos los elementos ordenados por un valor. Esta transacción podría funcionar a la primera, si el número total de elementos es 1000, pero si el número total de elementos llega a 10 millones, la transacción devuelve todos los 10 millones de elementos. Si ejecuta esta transacción, los dos resultados más probables son que la transacción agote el tiempo o que el cliente encuentre un error de memoria agotada.

La mejor opción es alterar la lógica empresarial de forma que sólo se puedan devolver los 10 o 20 primeros elementos. La alteración de la lógica mantiene el tamaño de la transacción gestionable, independientemente de cuántos elementos contenga la memoria caché.

Definir esquema

La principal ventaja de normalizar los datos es que el sistema de la base de datos puede ocuparse de la coherencia de los datos de forma interna. Cuando se elimina la normalización de los datos para la escalabilidad, deja de existir esta gestión automática de la coherencia de los datos. Debe implementar un modelo de datos que pueda funcionar en la capa de la aplicación o como un plug-in en la cuadrícula distribuida para garantizar la coherencia de los datos.

Considere el ejemplo del tablón de mensajes. Si una transacción elimina una publicación de un tema, se debe eliminar la publicación duplicada del registro de usuarios. Sin un modelo de datos, es posible que un desarrollador escriba el código de la aplicación para eliminar la publicación del tema y olvide eliminar la publicación del registro de usuarios. Sin embargo, si el desarrollador estuviera utilizando un modelo de datos, en lugar de interactuando directamente con la memoria caché, el método removePost en el modelo de datos podría extraer el ID de usuario de la publicación, buscar el registro de usuarios y eliminar la publicación duplicada de forma interna.

De forma alternativa, puede implementar un receptor que se ejecuta en la partición real que detecta el cambio en el tema y ajusta automáticamente el registro de usuarios. Un receptor podría ser beneficioso porque el ajuste del registro de usuarios se podría realizar de forma local si la partición parece tener el registro de usuarios, o incluso si el registro de usuarios está en una partición distinta, la transacción se produce entre los servidores, en lugar de entre el cliente y el servidor. Probablemente, la conexión de red entre los servidores es más rápida que la conexión de red entre el cliente y el servidor.

Impedir la competencia

Se impiden escenarios como tener un contador global. La cuadrícula no se escalará si un único registro utiliza un número desproporcionado de tiempos en comparación con el resto de los registros. El rendimiento de la cuadrícula se limitará por el rendimiento del sistema que aloja el registro determinado.

En estas situaciones, intente dividir el registro para que sea gestionado por partición. Por ejemplo, considere una transacción que devuelve el número total de entradas en la memoria caché distribuida. En lugar de tener cada acceso de la operación insert y remove en un único registro que aumenta, tener un receptor en cada partición rastrea las operaciones insert y remove. Con este rastreo del receptor, las operaciones insert y remove se pueden convertir en operaciones de partición única.

La lectura del contador se convertirá en una operación entre cuadrícula, pero para la mayor parte, ya era tan ineficaz como una operación entre cuadrícula, porque su rendimiento estaba unido al rendimiento del sistema que incluye el registro.

Sugerencias de implementación

También puede considerar las siguientes sugerencias para conseguir la mejor escalabilidad.

Utilizar los índices de búsqueda inversa

Considere un modelo de datos que ha eliminado la normalización correctamente donde los registros de clientes se particionan basándose en el número del ID de cliente. Este método de particionamiento es la opción lógica porque casi todas las operaciones empresariales realizadas con el registro de clientes utilizan el número del ID del cliente. Sin embargo, una transacción importante que no utiliza el número del ID del cliente es la transacción de inicio de sesión. Es más común tener nombres de usuario o direcciones de correo electrónico para el inicio de sesión, en lugar de números de ID de cliente.

El enfoque sencillo del escenario de inicio de sesión es utilizar una transacción entre cuadrícula para encontrar el registro de clientes. Tal como se ha explicado previamente, este enfoque no realiza escaladas.

La siguiente opción podría ser la partición en el nombre del usuario o el correo electrónico. Esta opción no es práctica porque todos las operaciones basadas en el ID de cliente se convierten en transacciones entre cuadrícula. Asimismo, los clientes del sitio podrían desear cambiar su nombre de usuario o dirección de correo electrónico. Los productos como WebSphere eXtreme Scale necesitan el valor que se utiliza para la partición de datos en constantes de permanencia.

La solución correcta es utilizar un índice de búsqueda inversa. Con WebSphere eXtreme Scale, se puede crear una memoria caché en la misma cuadrícula distribuida que la memoria caché que aloja todos los registros de usuarios. Esta memoria caché es altamente disponible, está particionada y es escalable. Se puede utilizar esta memoria caché para correlacionar un nombre de usuario o dirección de correo electrónico con un ID de cliente. Esta memoria caché convierte el inicio de sesión en una operación de dos particiones, en lugar de una operación entre cuadrícula. Este escenario no es tan bueno como una transacción de partición única, pero el rendimiento se sigue escalando de forma lineal a medida que el número de sistemas aumenta.

Calcular en el momento de la escritura

Los valores calculados comúnmente como promedios o totales pueden resultar caros para generarse, porque normalmente estas operaciones requieren leer un gran número de entradas. Puesto que leer es más comunes que escribir en la mayoría de las aplicaciones, es eficaz calcular estos valores en el momento de escribir y, a continuación, almacenar el resultado en la memoria caché. Esta práctica hace que las operaciones de lectura sean más rápidas y más escalables.

Campos opcionales

Considere un registro de usuarios que incluya una empresa, un lugar y un número de teléfono. Un usuario puede tener todos estos números definidos, o ninguno o alguna combinación de éstos. Si los datos se normalizaron, podría existir una tabla de usuarios y una tabla de números de teléfono. Los números de teléfono para un usuario determinado se podrían encontrar utilizando una operación JOIN entre las dos tablas.

Eliminar la normalización de este registro no requiera la duplicación de datos, porque la mayoría de los usuarios no comparten los números de teléfono. En lugar de esto, debe estar permitido vaciar las ranuras del registro de usuarios. En lugar de tener una tabla de números de teléfono, añada tres atributos a cada registro de usuarios, uno para cada tipo de número de teléfono. Esta adición de atributos elimina la operación JOIN y realiza una búsqueda de número de teléfono para un usuario y una operación de partición única.

Colocación de relaciones de muchos a muchos

Considere una aplicación que rastrea los productos y las tiendas en las que se venden los productos. Un único producto se vende en muchas tiendas y una sola tienda vende muchos productos. Suponga que esta aplicación rastrea 50 tiendas grandes. Cada producto se vende en un máximo de 50 tiendas, con cada tienda que vende miles de productos.

Conservar una lista de tiendas dentro de la entidad de producto (disposición A), en lugar de conservar una lista de productos dentro de cada entidad de tienda (disposición B). Consultando algunas de las transacciones, esta aplicación podría realizar ilustraciones que expliquen por qué la disposición A es más escalable.

En primer lugar, consulte las actualizaciones. Con la disposición A, eliminar un producto del inventario de una tienda bloquea la entidad del producto. Si la cuadrícula contiene 10.000 productos, sólo el 1/10000 de la cuadrícula se debe bloquear para realizar la actualización. Con la disposición B, la cuadrícula sólo contiene 50 tiendas, así que el 1/50 de la cuadrícula debe estar bloqueada para completar la actualización. Así pues,

aunque ambas disposiciones se podrían considerar operaciones de partición única, la disposición A se escala de forma horizontal de forma más eficaz.

Ahora, si se consideran las lecturas con la disposición A, buscar las tiendas en las que se vende un producto es una transacción de partición única que se escala y es rápida porque la transacción sólo transmite una pequeña cantidad de datos. Con la disposición B, esta transacción pasa a ser una transacción entre cuadrícula porque se debe acceder a cada entidad de tienda para ver si el producto se vende en dicha tienda, que implica una gran ventaja de rendimiento para la disposición A.

Escalar con datos normalizados

Un uso legítimo de las transacciones entre cuadrícula es escalar el proceso de datos. Si una cuadrícula tiene 5 sistemas y se envía una transacción entre cuadrícula que clasifica unos 100.000 registros en cada sistema, dicha transacción clasifica unos 500.000 registros. Si el sistema más lento de la cuadrícula puede realizar 10 de estas transacciones por segundo, la cuadrícula es capaz de realizar clasificaciones entre 5.000.000 de registros por segundo. Si los datos de la cuadrícula se doblan, cada sistema realiza una clasificación entre 200.000 registros y cada transacción realiza una clasificación entre 1.000.000 de registros. Estos datos reducen el rendimiento del sistema más lento a 5 transacciones por segundo, por lo tanto, reduce el rendimiento de la cuadrícula a 5 transacciones por segundo. La cuadrícula realiza la clasificación entre 5.000.000 de registros por segundo.

En este escenario, doblar el número de sistemas permite a cada sistema volver a su carga previa de clasificación entre 100.000 registros, lo que permite al sistema más lento procesar 10 de estas transacciones por segundo. El rendimiento de la cuadrícula permanece igual a 10 peticiones por segundo, pero ahora cada transacción procesa 1.000.000 de registros, de forma que la cuadrícula ha doblado su capacidad en términos de proceso de registros a 10.000.000 por segundo.

Las aplicaciones como, por ejemplo, un motor de búsqueda que se debe escalar en términos de proceso de datos para adaptar el tamaño en crecimiento de Internet y el rendimiento para adaptar el crecimiento en el número de usuarios, debe crear varias cuadrículas con un turno circular de las peticiones entre las cuadrículas. Si debe escalar de forma vertical el rendimiento, añada sistemas y añada otra cuadrícula a las solicitudes de servicio. Si el proceso de datos se debe escalar de forma vertical, añada más sistemas y mantenga constante el número de cuadrículas.

Escalado en unidades o contenedores

Este tema trata sobre la escalabilidad, pero no del modo habitual, como por ejemplo cómo escalar la cuadrícula a un número X de JVM. La perspectiva que se ofrece aquí trata la escalabilidad en términos de operaciones, planificación y gestión de riesgos. Aunque estos factores suelen ser más importantes las consideraciones tradicionales sobre la escalabilidad del producto, lamentablemente suelen pasarse por alto. Para crear sistemas de alta disponibilidad se requieren los dos tipos de consideraciones sobre la escalabilidad para desplegar eXtreme Scale en un proceso de despliegue fiable.

Despliegue de una sola cuadrícula de gran tamaño

Las pruebas han verificado que eXtreme Scale puede escalar hasta más de 1.000 JVM. Estas pruebas animan a crear aplicaciones para desplegar cuadrículas individuales en grandes cantidades de máquinas. Aunque sea posible hacerlo, no se recomienda, por varios motivos que se indican a continuación.

1. **Cuestiones presupuestarias:** Su entorno no puede probar de forma realista una cuadrícula de 1.000 servidores. No obstante, puede probar una cuadrícula mucho más pequeña considerando los motivos de presupuesto, de modo que no tenga que comprar el doble de hardware, sobre todo para un número de servidores tan elevado.
2. **Diferentes versiones de aplicaciones:** Necesitar un número elevado de máquinas para cada hebra de pruebas no es práctico. El riesgo consiste en que no se prueban los mismos factores que se probarían en un entorno de producción.
3. **Pérdida de datos:** La ejecución de una base de datos en un solo disco duro no resulta fiable. Cualquier problema con el disco duro provocará una pérdida de datos. La ejecución de una aplicación creciente en una sola cuadrícula es similar. Probablemente experimentará errores en el entorno y en las aplicaciones. Por lo tanto, la colocación de todos los datos en un solo sistema de gran tamaño a menudo provocará una pérdida de grandes cantidades de datos.

División de la cuadrícula

La división de la cuadrícula de aplicación en contenedores (unidades) es más fiable. Un contenedor es un grupo de servidores que ejecutan una pila de aplicaciones homogénea. Los contenedores pueden ser de cualquier tamaño, pero lo ideal es que consten de unas 20 máquinas. En lugar de tener 500 máquinas en una sola cuadrícula, puede tener 25 contenedores de 20 máquinas. Una sola versión de una pila de aplicaciones se debe ejecutar en un determinado contenedor, pero distintos contenedores pueden tener sus propias versiones de una pila de aplicaciones.

Generalmente, una pila de aplicaciones tiene en cuenta niveles de los componentes siguientes.

- Sistema operativo
- Hardware
- JVM
- Versión de eXtreme Scale
- Aplicación
- Otros componentes necesarios

Un contenedor es una unidad de despliegue de tamaño apropiado para las pruebas. En lugar de tener cientos de servidores para las pruebas, es más práctico tener 20 servidores. En este caso, se sigue probando la misma configuración que tendría en el entorno de producción. En el entorno de producción se utilizan cuadrículas con un tamaño máximo de 20 servidores, que constituyen un contenedor. Puede someter a pruebas de tensión un solo contenedor y determinar su capacidad, número de usuarios, cantidad de datos y rendimiento de las transacciones. Esto facilita la planificación y se ajusta al estándar de disponer de un escalamiento previsible por un coste previsible.

Configuración de un entorno basado en contenedor

En diferentes casos, el contenedor no tiene que tener necesariamente 20 servidores. El objetivo del tamaño del contenedor es ajustarse a las pruebas prácticas. El tamaño de un contenedor debe ser suficientemente pequeño, de modo que si un contenedor encuentra problemas en producción la fracción de transacciones afectadas resulte tolerable.

Lo ideal es que cualquier error posible sólo afecte a un único contenedor. En el ejemplo anterior, un error sólo afectaría al 4% de las transacciones de aplicación en lugar de afectar al 100%. Además, las actualizaciones resultan más fáciles porque se pueden desplegar en un contenedor a la vez. Esto simplifica el escenario, de modo que si una actualización de un contenedor causa problemas el usuario pueda devolver dicho contenedor al nivel anterior. Entre las actualizaciones figuran los cambios a la aplicación, la pila de aplicaciones o las actualizaciones del sistema. En la mayor medida posible, las actualizaciones sólo deben cambiar un único elemento de la pila a la vez para conseguir que el diagnóstico de problemas resulte más preciso.

Para implementar un entorno con contenedores, necesita una capa de direccionamiento sobre los contenedores que sea compatible con versiones anteriores y posteriores si se aplican actualizaciones de software a los contenedores. Además, debe crear un directorio que incluya la información acerca de qué contenedor incluye qué datos. (Puede utilizar otra cuadrícula de eXtreme Scale con este fin con una base de datos tras ella, preferiblemente utilizando el escenario de grabación diferida). Esto ofrece una solución de dos niveles. El nivel 1 es el directorio y se utiliza para ubicar qué contenedor maneja una determinada transacción. El nivel 2 consta de los propios contenedores. Cuando el nivel 1 identifica un contenedor, la configuración direcciona cada transacción al servidor correcto del contenedor, que es por lo general el servidor que contiene la partición correspondiente a los datos utilizados por la transacción. Opcionalmente, también puede utilizar una memoria caché cercana en el nivel 1 para reducir el impacto asociado con la búsqueda del contenedor correcto.

La utilización de contenedores es un poco más compleja que disponer de una sola cuadrícula, pero las pruebas operativas y las mejoras de fiabilidad hacen que sea una parte crucial de las pruebas de escalabilidad.

Capítulo 5. Visión general de disponibilidad

Alta disponibilidad

Con una alta disponibilidad, WebSphere eXtreme Scale proporciona datos fiables, redundancia y detección de anomalías.

WebSphere eXtreme Scale organiza automáticamente las cuadrículas de Mäquinas virtuales Java en un árbol federado de manera ligera, con el servicio de catálogo en la raíz y los grupos principales que contienen contenedores en las hojas del árbol. Si desea más información, consulte "Arquitectura de memoria caché: correlaciones, contenedores, clientes y catálogos" en la página 11.

El servicio de catálogo crea cada grupo principal automáticamente en grupos de unos 20 servidores. Los miembros del grupo principal proporcionan la supervisión de estado de los otros miembros del grupo. Además, cada grupo principal elige un miembro para que sea el líder para comunicar la información del grupo al servicio de catálogo. Limitar el tamaño del grupo principal permite una supervisión de buena salud y un entorno con una gran capacidad de ampliación.

Nota: en un entorno de WebSphere Application Server, en el que se puede modificar el tamaño del grupo principal, eXtreme Scale no admite más de 50 miembros por grupo principal.

Anomalías

Puede producirse una anomalía en un proceso por diversas causas. La anomalía puede ser debida a que se ha alcanzado un límite de recursos, como el tamaño máximo de almacenamiento dinámico, o que alguna lógica de control de proceso terminara un proceso. El sistema operativo podría fallar, lo que implicaría que se perdieran todos los procesos que se estuvieran ejecutando en el sistema. El hardware puede fallar, aunque es menos frecuente, como por ejemplo la tarjeta de interfaz de red (NIC), lo que provocaría que el sistema operativo se desconectase de la red. Pueden producirse más puntos de anomalías, que dejaría el proceso como no disponible. En este contexto, todas estas anomalías pueden clasificarse en uno de estos dos tipos: anomalías de proceso y pérdida de conectividad.

Anomalías de proceso

WebSphere eXtreme Scale reacciona a las anomalías del proceso muy rápidamente. Cuando se produce una anomalía en un proceso, el sistema operativo es el responsable de limpiar los recursos sobrantes que utilizada el proceso. Esta limpieza incluye la asignación de puertos y conectividad. Cuando un proceso falla, se envía una señal a las conexiones que el proceso utilizaba para cerrar cada conexión. Gracias a estas señales, otro proceso conectado con el proceso que ha fallado puede detectar inmediatamente una anomalía en el proceso.

Pérdida de conectividad

Se produce pérdida de conectividad cuando se desconecta el sistema operativo. Como resultado, el sistema operativo no puede enviar señales a otros procesos. Las razones de la pérdida de conectividad son diversas, pero se pueden dividir en dos categorías: anomalía de host y aislamiento.

Anomalía de host

Si la máquina se desconecta de la corriente, se apaga inmediatamente.

Aislamiento

Este escenario presenta la condición de anomalía más complicada para que el software pueda gestionarlo correctamente porque el proceso aparenta no estar disponible, aunque lo esté. Básicamente, el sistema cree que un servidor u otro proceso ha fallado, mientras que realmente se está ejecutando correctamente.

Anomalía en el contenedor de eXtreme Scale

Las anomalías de contenedor generalmente las descubren los contenedores de igual a través del mecanismo de grupo principal. Cuando se produce una anomalía en un contenedor o grupo de contenedores, el servicio de catálogo migra los fragmentos alojados en dichos contenedores. El servicio de catálogo busca primero una réplica síncrona antes de migrar a una réplica asíncrona. Después de que los fragmentos primarios se migren a contenedores de host nuevos, el servicio de catálogo busca los contenedores host nuevos de las réplicas que faltan.

Nota: Aislamiento de contenedor: el servicio de catálogo migra los fragmentos de los contenedores, cuando se descubre que el contenedor no está disponible. Si dichos contenedores pasan a estar disponibles, el servicio de catálogo considera los contenedores adecuados para colocación como si fuera un flujo de arranque normal.

Latencia de detección de sustitución por anomalía del contenedor

Las anomalías se pueden dividir en anomalías de poca importancia y anomalías graves. Las anomalías de poca importancia se suelen producir por un fallo en el proceso. Este tipo de anomalías las detecta el sistema operativo, que es capaz de recuperar rápidamente los recursos utilizados, como los sockets de red. La detección de este tipo de anomalía se realiza en menos de un segundo. Las anomalías graves pueden tardar hasta 200 segundos en detectarse mediante el ajuste de pulsación predeterminado. Este tipo de anomalías incluye lo siguiente: fallos físicos de la máquina, desconexiones del cable de red o anomalías del sistema operativo. Por lo tanto, eXtreme Scale debe confiar en el mecanismo de pulsación para detectar anomalías graves que pueden configurarse. Consulte el apartado “Tipos de detección de migración tras error” en la página 117 para obtener detalles sobre cómo disminuir el tiempo que se tarda en detectar una anomalía grave.

Anomalía del servicio de catálogo

Puesto que la cuadrícula del servicio de catálogo es una cuadrícula de eXtreme Scale, también utiliza el mecanismo de agrupación principal del mismo modo que el proceso de anomalía del contenedor. La diferencia principal es que el dominio de servicio de catálogo utiliza un proceso de elección de igual para definir el fragmento primario, en lugar del algoritmo del servicio de catálogo que se utiliza para los contenedores.

Tenga en cuenta que el servicio de colocación y el servicio de agrupamiento principal son uno de N servicios. Uno de N servicios se ejecuta en un miembro del grupo de alta disponibilidad. El servicio de ubicación y la administración se ejecuan en todos los miembros del grupo de alta disponibilidad. El servicio de

colocación y el servicio de agrupamiento principal son objetos singleton porque son responsables de la presentación del sistema. El servicio de ubicación y administración son servicios de solo lectura y existen en cualquier punto para proporcionar escalabilidad.

El servicio de catálogo utiliza la réplica para convertirse en tolerante a errores. Si se produce una anomalía en un proceso de servicio de catálogo, el servicio debe reiniciarse para restaurar el sistema al nivel deseado de disponibilidad. Si fallan todos los procesos que alojan el servicio de catálogo, eXtreme Scale sufre una pérdida de datos críticos. Esta anomalía provoca un reinicio obligatorio de todos los contenedores. Como el servicio de catálogo puede ejecutarse en numerosos procesos, esta anomalía es poco probable. No obstante, si ejecuta todos los procesos en una única máquina, dentro de un armazón blade sencillo, o en un conmutador de red, es más probable que se produzca una anomalía. Elimine las modalidades de anomalías comunes de las máquinas que alojan el servicio de catálogo para reducir la posibilidad de anomalía.

Anomalías de varios contenedores

Una réplica nunca se coloca en el mismo proceso que su fragmento primario porque si el proceso se pierde, se perderían tanto la réplica como el fragmento primario. La política de despliegue define un atributo booleano de modalidad de desarrollo que utiliza el servicio de catálogo para determinar si se puede colocar una duplicación en la misma máquina que un primario. En un entorno de despliegue en una única máquina, puede tener dos contenedores y realizar réplicas entre ellos. En producción, sin embargo, es suficiente el uso de una única máquina porque la pérdida de dicho host resultaría en la pérdida de los dos contenedores. Para cambiar de modalidad de desarrollo en una única máquina a una modalidad de producción con varias máquinas y viceversa, inhabilite la modalidad de desarrollo en el archivo de configuración de la política de despliegue.

Réplica para la disponibilidad

La duplicación proporciona tolerancia a anomalías y aumenta el rendimiento para una topología de eXtreme Scale distribuida.

La réplica se habilita asociando BackingMaps con un MapSet.

Un MapSet es una colección de correlaciones que se categorizan por clave de partición. Esta clave de partición se obtiene de la clave del correlación individual efectuando una operación módulo entre hash y el número de particiones. Por lo tanto, si un grupo de correlaciones dentro de MapSet tiene la clave de partición X, estas correlaciones se almacenarán en la correspondiente partición X en la cuadrícula; si otro grupo tiene la clave de partición Y, todas las correlaciones se almacenarán en la partición Y, etc. Además, los datos de las correlaciones se replican en función de la política definida en MapSet, que sólo se utiliza para topologías de eXtreme Scale distribuidas (no es necesario para instancias locales).

Si desea más información, consulte “Particionamiento” en la página 93.

A los MapSets se les asigna el número de particiones que tendrán y una política de réplica. La configuración de réplica de MapSet simplemente identifica el número de fragmentos de réplicas síncronas y asíncronas que un MapSet debe tener además del fragmento primario. Por ejemplo, si debe haber una réplica síncrona y una asíncrona, en cada una de las BackingMaps asignadas al MapSet se distribuirá automáticamente un fragmento de réplica dentro del conjunto de contenedores

disponibles para eXtreme Scale. La configuración de réplica también puede permitir que los clientes lean datos de servidores duplicados de forma síncrona. Esto puede esparcir la carga de las solicitudes de lectura entre servidores adicionales en eXtreme Scale. La réplica sólo tiene un impacto de modelo de programación cuando se realiza la precarga de BackingMaps.

Para obtener detalles sobre las distintas opciones de configuración, consulte más abajo:

Precarga de correlaciones

Las correlaciones se pueden asociar a cargadores. Un cargador se utiliza para captar objetos cuando no se pueden encontrar en la correlación (una falta de coincidencia) así como para grabar los cambios en un programa de fondo cuando se confirma una transacción. Los cargadores también se pueden utilizar para cargar previamente datos en una correlación. El método preloadMap de la interfaz del cargador se invoca en cada correlación cuando su correspondiente partición de MapSet pasa a ser un fragmento primario. El método preloadMap no se invoca en réplicas. Intenta cargar todos los datos referenciados previstos del programa de fondo en la correlación utilizando la sesión proporcionada. La correlación relevante la identifica el argumento BackingMap que se pasa al método preloadMap.

```
void preloadMap(Session session, BackingMap backingMap) throws LoaderException;
```

Precarga en el MapSet particionado

Las correlaciones puede particionarse en N particiones. Por lo tanto, las correlaciones pueden extenderse por varios servidores, con cada entrada identificada por una clave que sólo se almacena en uno de esos servidores. Las correlaciones muy grandes pueden mantenerse en un eXtreme Scale porque la aplicación ya no está limitada por el tamaño del almacenamiento dinámico de una sola JVM para mantener todas las entradas de una correlación. Las aplicaciones que desea cargar previamente con el método preloadMap de la interfaz del cargador deben identificar el subconjunto de datos que carga previamente. Siempre existe un número fijo de particiones. Puede determinar este número utilizando el siguiente ejemplo de código:

```
int numPartitions = backingMap.getPartitionManager().getNumOfPartitions();
int myPartition = backingMap.getPartitionId();
```

Este ejemplo de código muestra como una aplicación puede identificar un subconjunto de datos que se debe cargar previamente de la base de datos. Las aplicaciones siempre deben utilizar estos métodos incluso cuando la correlación no está particionada inicialmente. Estos métodos permiten una cierta flexibilidad: si posteriormente los administradores particionan la correlación, el cargador sigue funcionando correctamente.

La aplicación debe emitir consultas para recuperar el subconjunto *myPartition* del programa de fondo. Si se utiliza una base de datos, puede ser más fácil tener una columna con un identificador de partición para un registro dado salvo que haya alguna consulta natural que permita a los datos de la tabla particionarse fácilmente.

Consulte los detalles sobre cómo escribir un cargador con un controlador de precarga de réplica en *Guía de programación* para ver un ejemplo sobre cómo implementar un cargador para un eXtreme Scale duplicado.

Rendimiento

La implementación de la precarga copia datos del programa de fondo en la correlación almacenando varios objetos en la correlación de una única transacción. El número óptimo de registros para almacenar por transacción depende de varios factores, incluidos la complejidad y el tamaño. Por ejemplo, después de que la transacción incluya bloques de más de 100 entradas, se reduce la ventaja del rendimiento a medida que aumenta el número de entradas. Para determinar el número óptimo, empiece con 100 entradas y, a continuación, aumente el número hasta que no se detecte más aumento en el rendimiento. Las transacciones de mayor tamaño dan como resultado un mayor rendimiento de duplicación. Recuerde que sólo el fragmento primario ejecuta el código de precarga. Los datos cargados previamente se duplican desde el fragmento primario hasta todas las réplicas que están en línea.

Precarga de MapSets

Si la aplicación utiliza un MapSet con varias correlaciones, cada correlación tendrá su propio cargador. Cada cargador tiene un método de carga previa. eXtreme Scale carga cada correlación en serie. Será más eficaz precargar todas las correlaciones designando una única correlación como la correlación de precarga. Este proceso es un convenio de aplicación. Por ejemplo, dos correlaciones, departamento y empleado, podrían utilizar el cargador de departamento para cargar previamente las correlaciones de departamento y de empleado. Este procedimiento asegura que, transaccionalmente, si una aplicación desea un departamento los empleados de dicho departamento están en la memoria caché. Cuando el cargador de departamento precarga un departamento desde el programa de fondo, también capta los empleados de dicho departamento. El objeto de departamento y sus objetos de empleados asociados se añadirán a la correlación utilizando una sola transacción.

Precarga recuperable

Algunos clientes tienen conjuntos de datos de gran tamaño que necesitan almacenarse en la memoria caché. La precarga de estos datos puede requerir mucho tiempo. A veces, la precarga debe finalizar para que la aplicación pueda ir en línea. Puede sacar provecho de que la precarga sea recuperable. Suponga que hay un millón de registros que se deben precargar. El fragmento primario los precarga y falla al llegar al registro número 800.000. Normalmente, la réplica elegida como el nuevo fragmento primario borra los estados duplicados y empieza desde el principio. eXtreme Scale puede emplear una interfaz ReplicaPreloadController. El cargador de la aplicación también necesitará implementar la interfaz ReplicaPreloadController. Este ejemplo añade un solo método al cargador: `Status checkPreloadStatus(Session session, BackingMap bmap);`. Este método lo invoca el tiempo de ejecución de eXtreme Scale antes de que se llame al método de carga previa de la interfaz del cargador. eXtreme Scale comprueba el resultado de este método (estado) para determinar su comportamiento siempre que una réplica pasa a ser un fragmento primario.

Tabla 10. Valor de estado y respuesta

Valor de estado devuelto	Respuesta de eXtreme Scale
Status.PRELOADED_ALREADY	eXtreme Scale no llama al método de precarga porque su valor de estado indica que la correlación se ha precargado completamente.
Status.FULL_PRELOAD_NEEDED	eXtreme Scale borra la correlación y llama de forma normal al método de precarga.
Status.PARTIAL_PRELOAD_NEEDED	eXtreme Scale deja la correlación tal cual y llama a la precarga. Esta estrategia permite al cargador de aplicación seguir realizando la precarga a partir de ese momento.

Evidentemente, cuando un fragmento primario está cargando la correlación, debe dejar algún estado en una correlación del MapSet que se está duplicando para que la réplica determine qué estado debe devolver. Puede utilizar una correlación adicional llamada, por ejemplo, RecoveryMap. Esta RecoveryMap debe formar parte del mismo MapSet que se está precargando para asegurarse de que la correlación se duplica coherentemente con los datos que se están precargando. A continuación se muestra una implementación sugerida.

Cuando la precarga confirma cada bloque de registros, el proceso también actualiza un contador o valor en la RecoveryMap como parte de esa transacción. Los datos precargados y los datos de RecoveryMap se duplican de forma atómica en las réplicas. Cuando la réplica se promociona a fragmento primario, puede comprobar la RecoveryMap para ver qué ha pasado.

RecoveryMap puede mantener una sola entrada con la clave de estado. Si no existe ningún objeto para esta clave, es necesario una precarga completa (checkPreloadStatus devuelve FULL_PRELOAD_NEEDED). Si existe un objeto para esta clave de estado y el valor es COMPLETE, la precarga se completa y el método checkPreloadStatus devuelve PRELOADED_ALREADY. Si no, el objeto de valor indica donde se reinicia la precarga y el método checkPreloadStatus devuelve PARTIAL_PRELOAD_NEEDED. El cargador puede almacenar el punto de recuperación en una variable de instancia para el cargador de forma que, cuando se invoque la precarga, el cargador sepa el punto de partida. RecoveryMap también puede mantener una entrada por correlación si cada correlación se precarga independientemente.

Manejo de la recuperación en modalidad de duplicación síncrona con un cargador

El tiempo de ejecución de eXtreme Scale se ha diseñado para que no pierda datos confirmados cuando el fragmento primario falla. En la siguiente sección se muestran los algoritmos utilizados. Estos algoritmos sólo se aplican cuando un grupo de réplicas utiliza la réplica síncrona. Un cargador es opcional.

El tiempo de ejecución de eXtreme Scale puede configurarse de modo que duplique de forma síncrona todos los cambios de un fragmento primario en las réplicas. Cuando se coloca una réplica síncrona, recibe una copia de los datos existentes en el fragmento primario. Durante este tiempo, el primario continúa recibiendo transacciones y las copia en la réplica de forma asíncrona. La réplica no se considera en línea en este momento.

Después de que la réplica capte el primario, la réplica entra en la modalidad de igual y se inicia la réplica síncrona. Cada transacción confirmada en el primario se envía a las réplicas síncronas y el primario espera una respuesta de cada réplica. Una secuencia de confirmación síncrona con un cargador en el primario se parece al siguiente conjunto de pasos:

Tabla 11. Secuencia de confirmación del fragmento primario

Paso con cargador	**Paso sin cargador**
Obtener bloqueos para entradas	igual
Desechar cambios para el cargador	no operativo
Guardar cambios en la memoria caché	igual
Enviar cambios a réplicas y esperar el reconocimiento	igual

Tabla 11. Secuencia de confirmación del fragmento primario (continuación)

Paso con cargador	Paso sin cargador
Confirmar en el cargador a través del plug-in TransactionCallback	Se invoca el plug-in para enviar, pero no sucede nada
Liberar bloqueos para entradas	igual

Tenga en cuenta que los cambios se envían a la réplica antes de que se confirmen en el cargador. Para determinar cuando se confirman los cambios en la réplica, revise esta sentencia: en el momento de la inicialización, inicializar las listas tx en el fragmento primario tal como se indica a continuación.

```
CommitedTx = {}, RolledBackTx = {}
```

Durante el proceso de confirmaciónsíncrono, utilice la siguiente secuencia:

Tabla 12. Proceso de confirmación síncrona

Paso con cargador	Paso sin cargador
Obtener bloqueos para entradas	igual
Desechar cambios para el cargador	no operativo
Guardar cambios en la memoria caché	igual
Enviar cambios con una transacción confirmada, retrotraer transacción a la réplica y esperar al reconocimiento	igual
Borrar lista de transacciones confirmadas y transacciones retrotraídas	igual
Confirmar en el cargador a través del plug-in TransactionCallback	Se sigue llamando a la confirmación del plug-in TransactionCallBack, pero normalmente no sucede nada
Si la confirmación es satisfactoria, añada la transacción a las transacciones confirmadas, de lo contrario, añádala a las transacciones retrotraídas	no operativo
Liberar bloqueos para entradas	igual

Para el proceso de réplicas, utilice la siguiente secuencia:

1. Recibir cambios
2. Confirmar todas las transacciones recibidas en la lista de transacciones confirmadas
3. Retrotraer todas las transacciones recibidas en la lista de transacciones retrotraídas
4. Iniciar una transacción o sesión
5. Aplicar cambios en la transacción o sesión
6. Guardar la transacción o sesión en la lista de pendientes
7. Devolver respuesta

Tenga en cuenta que en la réplica, no se produce ninguna interacción de cargador mientras la réplica está en modalidad de réplica. El fragmentoprimario debe pasar todos los cambios a través del cargador. La réplica no realiza ningún cambio. Un efecto secundario de este algoritmo es que la réplica siempre tiene las transacciones, pero éstas no se confirman hasta que la siguiente transacción

primaria envía el estado de confirmado de estas transacciones. A continuación, las transacciones se confirman o retrotraen en la réplica. Hasta entonces, las transacciones no están confirmadas. Puede añadir un temporizador en el fragmento primario que envía el resultado de la transacción después de un breve periodo de tiempo (unos pocos minutos). Este temporizador limita, pero no elimina, cualquier obsolescencia a este periodo de tiempo. Esta obsolescencia sólo es un problema si se utiliza la modalidad de lectura de réplica. Si no, la obsolescencia no tiene ningún impacto en la aplicación.

Cuando el fragmento primario falla, es probable que haya unas pocas transacciones confirmadas o retrotraídas en el fragmento primario, pero el mensaje nunca llega a la réplica con estos resultados. Cuando una réplica se promociona y pasa a ser el nuevo fragmento primario, una de las primeras acciones es manejar esta condición. Cada transacción pendiente se vuelve a procesar respecto al conjunto de correlaciones del nuevo fragmento primario. Si hay un cargador, cada transacción se ofrece al cargador. Estas transacciones se aplican estrictamente en el orden primero en entrar, primero en salir (FIFO). Si una transacción falla, ésta se ignora. Si hay tres transacciones pendientes, A, B y C, A podría confirmarse, B podría retrotraerse y C podría también confirmarse Ninguna transacción tiene ningún impacto en las demás. Suponga que son independientes.

Un cargador puede que desee utilizar una lógica un poco distinta cuando está en modalidad de recuperación de migración tras error comparada con la modalidad normal. El cargador puede saber fácilmente cuando está en modalidad de recuperación de migración tras error implementando la interfaz ReplicaPreloadController. El método checkPreloadStatus sólo se invoca cuando se completa la recuperación de la migración tras error. Por lo tanto, si el método de aplicación de la interfaz del cargador se invoca antes del método checkPreloadStatus, se trata de una transacción de recuperación. Después de llamar al método checkPreloadStatus, la recuperación de migración tras error está completa.

Equilibrio de carga entre réplicas

eXtreme Scale, salvo que se configure de otra manera, envía todas las solicitudes de lectura y grabación al servidor primario para un grupo de réplicas determinado. El fragmento primario debe atender todas las solicitudes de los clientes. Es posible que desee permitir que las solicitudes de lectura se envíen a las réplicas del fragmento primario. Si envía solicitudes de lectura a las réplicas la carga de las solicitudes de envío se podrá compartir entre varias JVM (Java Virtual Machines). No obstante, el uso de réplicas para las solicitudes de lectura puede generar repuestas incoherentes.

El equilibrio de carga entre réplicas normalmente se utiliza sólo cuando los clientes almacenan en la memoria caché datos que almacenan siempre o cuando los clientes utilizan el bloqueo pesimista.

Si lo datos se almacenan en la memoria caché constantemente y luego se invalidan en la memoria caché cercana del cliente, como resultado el fragmento primario debe detectar un índice de solicitudes get relativamente alto de los clientes. Asimismo, en modalidad de bloqueo pesimista, no existe memoria caché local, y por ello todas las solicitudes se envían al fragmento primario.

Si los datos son relativamente estáticos o si no se utiliza la modalidad pesimista, el envío de solicitudes de lectura a la réplica no tendrá un gran impacto en el rendimiento. La frecuencia de las solicitudes get de los clientes con memoria caché que están llenas de datos no es alta.

Cuando un cliente se inicia, su memoria caché cercana está vacía. Las solicitudes de memoria caché para la memoria caché vacía se remiten al fragmento primario. La memoria caché del cliente obtendrá datos con el tiempo, lo que provocará que la carga de solicitudes baje. Si una gran cantidad de clientes se inicia simultáneamente, la carga puede ser significativa y la lectura de réplicas puede ser una opción de rendimiento apropiada.

Réplica del lado del cliente

Con eXtreme Scale, puede duplicar una correlación de servidor con uno o más clientes utilizando la réplica asíncrona. Un cliente puede solicitar una copia de sólo lectura local de una correlación en el servidor utilizando el método ClientReplicableMap.enableClientReplication.

```
void enableClientReplication(Mode mode, int[] partitions, ReplicationMapListener
listener) throws  ObjectGridException;
```

El primer parámetro es la modalidad de réplica. Esta modalidad puede ser una réplica continua o una réplica de instantánea. El segundo parámetro es una matriz de ID de particiones que representan las particiones desde las que duplicar los datos. Si el valor es nulo o una matriz vacía, los datos se duplican desde todas las particiones. El último parámetro es un escucha para recibir los sucesos de réplica de cliente. Para obtener detalles, consulte ClientReplicableMap y ReplicationMapListener en la documentación de la API.

Después de habilitar la réplica, el servidor empieza a duplicar la correlación con el cliente. Con el tiempo, el cliente sólo estará a unas pocas transacciones por detrás del servidor en cualquier momento dado.

Tipos de detección de migración tras error

WebSphere eXtreme Scale puede detectar anomalías de forma fiable.

Pulsaciones

1. Los sockets se mantienen abiertos entre Mäquinas virtuales Java, y si un socket se cierra inesperadamente, este cierre imprevisto se detecta como una anomalía de la Máquina virtual Java de igual. Esta detección capta los casos de anomalías como, por ejemplo, la Máquina virtual Java que termina muy rápidamente. Dicha detección también permite la recuperación de estos tipos de anomalías, normalmente, en menos de un segundo.
2. Otros tipos de anomalías incluyen: una situación muy grave del sistema operativo, una anomalía del servidor físico o una anomalía en la red. Estas anomalías se descubren mediante pulsaciones.

Las pulsaciones se envían de forma periódica entre pares de procesos: cuando se pierde un número fijo de pulsaciones, se supone que hay una anomalía. Este enfoque detecta las anomalías en N*M segundos, donde N es el número de pulsaciones que se han perdido y M es el intervalo en el que se deben establecer las pulsaciones. No se da soporte a la especificación directa de M y N; en cambio, se utiliza un mecanismo de graduador para que se puedan utilizar un rango de combinaciones M y N probadas.

Anomalías

Puede producirse una anomalía en un proceso por diversas causas. La anomalía puede ser debida a que se ha alcanzado un límite de recursos, como el tamaño máximo de almacenamiento dinámico, o que alguna lógica de control de proceso terminara un proceso. El sistema operativo podría fallar, lo que implicaría que se perdieran todos los procesos que se estuvieran ejecutando en el sistema. El hardware puede fallar, aunque es menos frecuente, como por ejemplo la tarjeta de interfaz de red (NIC), lo que provocaría que el sistema operativo se desconectase de la red. En este contexto, todas estas anomalías pueden clasificarse en uno de estos dos tipos: anomalías de proceso y pérdida de conectividad.

Anomalías de proceso

WebSphere eXtreme Scale reacciona para procesar las anomalías muy rápidamente. Cuando se produce una anomalía en un proceso, el sistema operativo es el responsable de limpiar los recursos sobrantes que utilizada el proceso. Esta limpieza incluye la asignación de puertos y conectividad. Cuando un proceso falla, se envía inmediatamente una señal a las conexiones que el proceso utilizaba para cerrar cada conexión.

Pérdida de conectividad

Se produce pérdida de conectividad cuando se desconecta el sistema operativo. Como resultado, el sistema operativo no puede enviar señales a otros procesos. Las razones de la pérdida de conectividad son diversas, pero se pueden dividir en dos categorías: anomalía de host y aislamiento.

Anomalía de host

Si se interrumpe la alimentación eléctrica en una máquina host, ésta deja de estar disponible al instante.

Aislamiento

Este escenario presenta la condición de anomalía más complicada para que el software pueda gestionarlo correctamente porque el proceso aparenta no estar disponible, aunque lo esté.

Anomalía de contenedor

Las anomalías de contenedor generalmente las descubren los contenedores de igual a través del mecanismo de grupo principal. Cuando se produce una anomalía en un contenedor o grupo de contenedores, el servicio de catálogo migra los fragmentos alojados en dichos contenedores. El servicio de catálogo busca primero una réplica síncrona antes de migrar a una réplica asíncrona. Después de que los fragmentos primarios se migren a contenedores de host nuevos, el servicio de catálogo busca los contenedores host nuevos de las réplicas que faltan.

Nota: Aislamiento de contenedor: el servicio de catálogo migra los fragmentos de los contenedores, cuando se descubre que el contenedor no está disponible. Si dichos contenedores pasan a estar disponibles, el servicio de catálogo considera los contenedores adecuados para colocación como si fuera un flujo de arranque normal.

Latencia de detección de sustitución por anomalía del contenedor

Las anomalías se pueden dividir en anomalías de poca importancia y anomalías graves. Las anomalías de poca importancia se suelen producir por un fallo en el proceso. Este tipo de anomalías las detecta el sistema operativo, que es capaz de recuperar rápidamente los recursos utilizados, como los sockets de red. La detección de este tipo de anomalía se realiza en menos de un segundo. Las anomalías graves pueden tardar hasta 200 segundos en detectarse mediante el ajuste de pulsación predeterminado. Este tipo de anomalías incluye lo siguiente: fallos físicos de la máquina, desconexiones del cable de red o anomalías del sistema operativo. Por lo tanto, eXtreme Scale debe confiar en el mecanismo de pulsación para detectar anomalías graves que pueden configurarse.

Anomalías de varios contenedores

Una réplica nunca se coloca en el mismo proceso que su fragmento primario porque si el proceso se pierde, se perderían tanto la réplica como el fragmento primario. La política de despliegue define un atributo que utiliza el servicio de catálogo para determinar si una réplica puede colocarse en la misma máquina que un fragmento primario. En un entorno de despliegue en una única máquina, puede tener dos contenedores y realizar réplicas entre ellos. En producción, sin embargo, es suficiente el uso de una única máquina porque la pérdida de dicho host resultaría en la pérdida de los dos contenedores. Para cambiar de modalidad de desarrollo en una única máquina a una modalidad de producción con varias máquinas y viceversa, inhabilite la modalidad de desarrollo en el archivo de configuración de la política de despliegue.

Anomalía del servicio de catálogo

Puesto que la cuadrícula del servicio de catálogo es una cuadrícula de eXtreme Scale, también utiliza el mecanismo de agrupación principal del mismo modo que el proceso de anomalía del contenedor. La diferencia principal es que el dominio de servicio de catálogo utiliza un proceso de elección de igual para definir el fragmento primario, en lugar del algoritmo del servicio de catálogo que se utiliza para los contenedores.

Tenga en cuenta que el servicio de colocación y el servicio de agrupamiento principal son uno de N servicios. Uno de N servicios se ejecuta en un miembro del grupo de alta disponibilidad. El servicio de ubicación y la administración se ejecuan en todos los miembros del grupo de alta disponibilidad. El servicio de colocación y el servicio de agrupamiento principal son objetos singleton porque son responsables de la presentación del sistema. El servicio de ubicación y administración son servicios de solo lectura y existen en cualquier punto para proporcionar escalabilidad.

El servicio de catálogo utiliza la réplica para convertirse en tolerante a errores. Si se produce una anomalía en un proceso de servicio de catálogo, el servicio debe reiniciarse para restaurar el sistema al nivel deseado de disponibilidad. Si fallan todos los procesos que alojan el servicio de catálogo, eXtreme Scale sufre una pérdida de datos críticos. Esta anomalía provoca un reinicio obligatorio de todos los contenedores. Como el servicio de catálogo puede ejecutarse en numerosos procesos, esta anomalía es poco probable. No obstante, si ejecuta todos los procesos en una única máquina, dentro de un armazón blade sencillo, o en un conmutador de red, es más probable que se produzca una anomalía. Elimine las modalidades de anomalías comunes de las máquinas que alojan el servicio de catálogo para reducir la posibilidad de anomalía.

Tabla 13. Resumen del descubrimiento de anomalías y la recuperación

Tipo de pérdida	Mecanismo de descubrimiento (detección)	Método de recuperación
Pérdida de proceso	E/S	Reiniciar
Pérdida del servidor	Pulsación	Reiniciar
Parada de la red	Pulsación	Restablecer la red y la conexión
Bloqueo del lado del servidor	Pulsación	Detener y reiniciar el servidor
Servidor ocupado	Pulsación	Esperar hasta que el servidor esté disponible

Servicio de catálogo de alta disponibilidad

Un dominio de servicio de catálogo es la cuadrícula de los servidores de catálogo que utiliza, que conservan la información de topología para todos los contenedores en el entorno de eXtreme Scale. El servicio de catálogo controla el equilibrio de carga y el direccionamiento para todos los clientes. Para desplegar eXtreme Scale como un espacio de proceso de base de datos en memoria, debe agrupar en clúster el servicio de catálogo en un dominio de servicio de catálogo para alta disponibilidad.

Componentes del dominio de servicio de catálogo

Cuando se inician varios servidores, uno de ellos se elige como el servidor de catálogo maestro que acepta las pulsaciones IIOP (Internet Inter-ORB Protocol) y maneja los cambios de datos del sistema en respuesta a cualquier cambio de servicio de catálogo o contenedor.

Cuando los clientes se ponen en contacto con uno de los servidores de catálogo, la tabla de direccionamiento del dominio de servicio de catálogo se propaga en los clientes a través del contexto del servicio CORBA (Common Object Request Broker Architecture).

Configure, como mínimo, tres servidores de catálogo. Si la configuración tiene zonas, puede configurar un servidor de catálogo por zona.

Cuando un servidor o contenedor eXtreme Scale se pone en contacto con uno de los servidores de catálogo, la tabla de direccionamiento del dominio de servicio de catálogo también se propaga en el servidor y contenedor eXtreme Scale a través del contexto de servicio CORBA. Además, si el servidor de catálogo contactado no es actualmente el servidor de catálogo maestro, la solicitud se redirecciona automáticamente al servidor de catálogo maestro actual y la tabla de direccionamiento del servidor de catálogo se actualiza.

Nota: Una cuadrícula de servidor de catálogo y una cuadrícula de servidor de contenedor son muy diferentes. El dominio de servicio de catálogo es para la alta disponibilidad de los datos del sistema. La cuadrícula del contenedor es para la alta disponibilidad de datos, la escalabilidad y la gestión de carga de trabajo. Por lo tanto, existen dos tablas de direccionamiento diferentes: la tabla de direccionamiento para el dominio de servicio de catálogo y la tabla de direccionamiento para los fragmentos de la cuadrícula de servidor.

Las responsabilidades del catálogo se dividen en una serie de servicios. El gestor de grupos principales realiza el agrupamiento de igual para la supervisor de estado, el servicio de colocación realiza la asignación, el servicio de administración proporciona acceso a la administración y el servicio de ubicación gestiona la localidad.

Despliegue del dominio de servicio de catálogo

Gestor de grupos principales

El servicio de catálogo utiliza el High Availability Manager (Gestor HA) para agrupar los procesos para la supervisión de la disponibilidad. Cada grupo de procesos es un grupo principal. Con eXtreme Scale, el gestor de grupos principales agrupa dinámicamente los procesos juntos. Estos procesos son pequeños para favorecer la escalabilidad. Cada grupo principal elige un líder que tiene la responsabilidad añadida de enviar el estado al gestor de grupos principales cuando se produce una anomalía en los miembros individuales. Se utiliza el mismo mecanismo de estado para descubrir cuando fallan todos los miembros de un grupo, que provoca que falle la comunicación con el líder.

El gestor de grupos principales es un servicio totalmente automático responsable de organizar los contenedores en pequeños grupos de servidores que se federen automáticamente de manera ligera para conformar un ObjectGrid. Cuando un contenedor se pone en contacto por primera vez con el servicio de catálogo, espera a ser asignado a un grupo nuevo o existente. Un despliegue de eXtreme Scale se compone de varios de estos grupos, y este agrupamiento es un habilitador de escalabilidad clave. Cada grupo es un grupo de Mäquinas virtuales Java que utiliza la pulsación para supervisar la disponibilidad de los otros grupos. Uno de los miembros de estos grupos es elegido líder y tiene la responsabilidad añadida de transmitir información de disponibilidad al servicio de catálogo para permitir reaccionar ante anomalías mediante la reasignación y reenvío de rutas.

Servicio de colocación

El servicio de catálogo gestiona la colocación de fragmentos en el conjunto de contenedores disponibles. El servicio de colocación es responsable de mantener un equilibrio de recursos en los recursos físicos. El servicio de colocación es responsable de asignar fragmentos individuales al contenedor host. El servicio de colocación se ejecuta como uno de los N servicios elegidos en la cuadrícula de datos de modo que siempre hay exactamente una instancia del servicio en ejecución. Si la instancia falla, se elige otro proceso, que tomará el control. El estado del servicio de catálogo se replica en todos los servidores que alojan el servicio de catálogo para favorecer la redundancia.

Administración

El servicio de catálogo es también el punto de entrada lógico para la administración del sistema. EL servicio de catálogo aloja un bean gestionado (MBean) y proporciona los URL de JMX (Java Management Extensions) para cualquiera de los servidores que gestiona el servicio.

Servicio de ubicación

El servicio de ubicación actúa como el punto de contacto para clientes que buscan los contenedores que alojan la aplicación que buscan, y también para los

contenedores que registran las aplicaciones alojadas con el servicio de colocación. El servicio de ubicación se ejecuta en todos los miembros de la cuadrícula para ampliar esta función.

El servicio de catálogo contiene lógica que está inactiva durante los estados fijos. Como resultado, el servicio de catálogo influye mínimamente en la escalabilidad. El servicio se crea para dar servicio a cientos de contenedores que pasan a estar disponibles al mismo tiempo. Para la disponibilidad, configure el servicio de catálogo en una cuadrícula.

Planificación

Después de iniciar un dominio de servicio de catálogo, los miembros de la cuadrícula se enlazan juntos. Planifique con atención la topología del dominio de servicio de catálogo, porque no podrá modificar la configuración del dominio de servicio de catálogo durante la ejecución. Extienda la cuadrícula tanto como sea posible para evitar errores.

Inicio de un dominio de servicio de catálogo

Si desea más información sobre cómo crear un dominio de servicio de catálogo, consulte la información sobre cómo iniciar un dominio de servicio de catálogo en la *Guía de administración*.

Conexión de contenedores eXtreme Scale incorporados en WebSphere Application Server con un dominio de servicio de catálogo autónomo

Puede configurar contenedores eXtreme Scale que están incorporados en un entorno WebSphere Application Server para conectarse a un dominio de servicio de catálogo autónomo.

- (En desuso) En releases anteriores, ha conectado los servicios de catálogo en un dominio de servicio de catálogo creando una propiedad personalizada. Esta propiedad se puede utilizar todavía, pero está en desuso. Para obtener más información sobre esta propiedad personalizada, consulte la información sobre cómo iniciar el proceso de servicio de catálogo en WebSphere Application Server en la *Guía de administración*..

  Nota: Colisión de nombre de servidor: puesto que esta propiedad se utiliza para iniciar el servidor de catálogo eXtreme Scale así como para conectarse, los servidores de catálogo no deben tener el mismo nombre que ningún servidor WebSphere Application Server.

Consulte “Quórums del servidor de catálogos” si desea más información.

Quórums del servidor de catálogos

El quórum es el número mínimo de servidores de catálogos necesario para realizar operaciones de colocación para la cuadrícula de datos. El número mínimo es el conjunto completo de servidores de catálogos en los que ha modificado el valor de quórum.

Términos importantes

Lo que aparece a continuación es una lista de términos relacionados con las consideraciones de quórum para WebSphere eXtreme Scale.

- **Bajada de tensión**: una bajada de tensión es la pérdida temporal de conectividad entre uno o más servidores.
- **Apagón**: un apagón es la pérdida permanente de conectividad entre uno o más servidores.
- **Centro de datos**: un centro de datos es un grupo localizado geográficamente de servidores conectados de forma general a una red de área local (LAN).
- **Zona**: una zona es una opción de configuración utilizada para agrupar los servidores que comparten algunas características físicas. A continuación se ofrecen algunos ejemplos de zonas para un grupo de servidores: un centro de datos tal como se describe en el punto anterior, una red de área, un edificio, un piso de un edificio, etc.
- **Pulsación**: la pulsación es un mecanismo utilizado para determinar si una determinada máquina virtual Java (JVM) se está ejecutando o no.

Topología

Esta sección explica cómo funciona WebSphere eXtreme Scale entre una red que incluye componentes no fiables. Los ejemplos de dicha red incluirían una red que abarca varios centros de datos.

Espacio de direcciones IP

WebSphere eXtreme Scale requiere una red donde los elementos direccionables en la red se pueden conectar a cualquier otro elemento direccionable en la red sin dificultades. Esto significa que WebSphere eXtreme Scale requiere un espacio de nombres de dirección IP sin formato y requiere que todos los cortafuegos permitan que todo el tráfico fluya entre las direcciones IP y los puertos utilizados por las máquinas virtuales Java (JVM) que incluyen elementos de WebSphere eXtreme Scale.

LAN conectadas

Cada LAN se asigna a un identificador de zona para los requisitos de WebSphere eXtreme Scale. WebSphere eXtreme Scale realiza de forma agresiva las pulsaciones en las JVM de una sola zona. Si se pierde una pulsación se generará un suceso de migración tras error si el servicio de catálogo tiene quórum.

Dominio de servicio de catálogo y servidores de contenedor

Un dominio de servicio de catálogo es una colección de JVM similares. Un dominio de servicio de catálogo es una cuadrícula formada por servidores de catálogos y con un tamaño fijo. Sin embargo, el número de servidores de contenedor es dinámico. Los servidores de contenedor se pueden añadir y eliminar según la demanda. En una configuración de tres centros de datos, WebSphere eXtreme Scale requiere una JVM de servicio de catálogo por centro de datos.

El dominio de servicio de catálogo utiliza un mecanismo de quórum completo. Dado este mecanismo de quórum completo, todos los miembros de la cuadrícula de datos deben acordar cualquier acción.

Las JVM del servidor de contenedor se marcan con un identificador de zona. La cuadrícula de la JVM del contenedor se divide automáticamente en pequeños grupos principales de JVM. Un grupo principal sólo incluye las JVM de la misma zona. Las JVM de distintas zonas nunca están en el mismo grupo principal.

Un grupo principal intenta detectar de forma agresiva la anomalía de sus JVM miembro. Las JVM de contenedor de un grupo principal nunca deben abarcar varias LAN conectadas con enlaces como en una red de área amplia. Esto significa que un grupo principal no puede tener contenedores en la misma zona ejecutándose en distintos centros de datos.

Ciclo de vida del servidor

Inicio del servidor de catálogo

Los servidores de catálogo se inician utilizando el mandato startOgServer. El mecanismo de quórum está inhabilitado de forma predeterminada. Para habilitar el quórum, pase el distintivo habilitado `-quorum` en el mandato startOgServer, o añada la propiedad `enableQuorum=true` al archivo de propiedades. Todos los servidores de catálogo deben tener el mismo valor de quórum.

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

Archivo objectGridServer.properties

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

Inicio del servidor de contenedor

Los servidores de contenedor se inician utilizando el mandato startOgServer. Cuando se ejecuta una cuadrícula de datos entre centros de datos, los servidores deben utilizar el distintivo de zona para identificar el centro de datos en el que residen. La definición de la zona en los servidores de cuadrícula permite a WebSphere eXtreme Scale supervisar el estado de los servidores con un ámbito de centro de datos, minimizando el tráfico entre el centro de datos.

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

Archivo objectGridServer.properties

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

Conclusión del servidor de cuadrícula

Los servidores de cuadrícula se detienen mediante el mandato stopOgServer. Cuando se concluye un centro de datos completo para el mantenimiento, pase la lista de todos los servidores que pertenecen a dicha zona. Esto permitirá una transición limpia de estado de la zona en eliminación a la zona o zonas supervivientes.

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

Detección de errores

WebSphere eXtreme Scale detecta la muerte de procesos a través de sucesos de cierre de socket anormal. Se informará inmediatamente al servicio de catálogo cuando termina un proceso. Se ha detectado un apagón a través de pulsaciones perdidas. WebSphere eXtreme Scale se protege a sí mismo contra las condiciones

de bajada de tensión entre centros de datos mediante el uso de una implementación de quórum.

Implementación de pulsaciones

Esta sección describe cómo se implementa la comprobación de integridad y concurrencia en WebSphere eXtreme Scale.

Pulsaciones del miembro del grupo principal

El servicio del catálogo coloca las JVM del contenedor en grupos principales de un tamaño limitado. Un grupo principal intentará detectar la anomalía de sus miembros utilizando dos métodos. Si se cierra un socket de la JVM, dicha JVM se considera como muerta. Cada miembro también realiza una pulsación sobre estos sockets a una velocidad determinada por la configuración. Si una JVM no responde a estas pulsaciones dentro de un periodo máximo de tiempo configurado, la JVM se considera como muerta.

Un único miembro de un grupo principal siempre se elige para ser el líder. El líder del grupo principal (CGL) es responsable de indicar periódicamente al servicio de catálogo que el grupo principal está activo e informa de cualquier cambio de pertenencia realizado en el servicio de catálogo. Un cambio de pertenencia puede ser una JVM que falla o una JVM recién añadida que se une al grupo principal.

Si el líder del grupo principal no puede contactar con ningún miembro del dominio del servicio de catálogo, seguirá intentándolo.

Pulsaciones del dominio del servicio de catálogo

El dominio del servicio de catálogo tiene el aspecto de un grupo principal privado con una pertenencia estática y un mecanismo de quórum. Detecta las anomalías del mismo modo que un grupo principal normal. Sin embargo, el comportamiento se modifica para incluir la lógica del quórum. El servicio de catálogo también utiliza una configuración de pulsaciones menos agresiva.

Pulsaciones de grupo principal

El servicio de catálogo debe saber cuándo fallan los servidores de contenedor. Cada grupo principal es responsable de determinar la anomalía de la JVM del contenedor y de informar a este servicio de catálogo a través del líder del grupo principal. La anomalía completa de todos los miembros de un grupo principal también es una posibilidad. Si ha fallado todo el grupo principal, es responsabilidad del servicio de catálogo detectar esta pérdida.

Si el servicio de catálogo marca una JVM de contenedor como anómala y se informa más adelante de que el contenedor está activo, se indicará a la JVM del contenedor que concluya los servidores de contenedor de WebSphere eXtreme Scale. Una JVM en este estado no es visible en las consultas del mandato xsadmin. Los mensajes en los registros de la JVM de contenedor indican que esta última ha producido un error. Debe reiniciar manualmente estas JVM.

Si se ha producido un suceso de pérdida de quórum, se suspende la pulsación hasta que se restablece el quórum.

Comportamiento del quórum del servicio de catálogo

Normalmente, los miembros del servicio de catálogo tienen conectividad completa. El dominio del servicio de catálogo es un conjunto estático de JVM. WebSphere eXtreme Scale espera que todos los miembros del servicio de catálogo siempre estén en línea. El servicio de catálogo sólo responde a los sucesos de contenedor mientras que el servicio de catálogo tenga quórum.

Si el servicio de catálogo pierde quórum, espera a que se restablezca el quórum. Durante el periodo de tiempo en que el servicio de catálogo no tiene quórum, ignora los sucesos de los servidores de contenedor. Los servidores de contenedor reintentan cualquier solicitud rechazada por el servicio de catálogo durante este tiempo ya que WebSphere eXtreme Scale espera a que se restablezca el quórum.

El siguiente mensaje indica que se ha perdido el quórum. Consulte este mensaje en los registros del servicio de catálogo.

```
CWOBJ1254W: El servicio de catálogo está esperando el quórum.
```

WebSphere eXtreme Scale espera perder el quórum por los siguientes motivos:

- El miembro de la JVM del servicio de catálogo falla
- Caída de red
- Pérdida del centro de datos

Detener una instancia del servidor de catálogo utilizando `stopOgServer` no provoca ninguna pérdida de quórum porque el sistema sabe que la instancia del servidor se ha detenido, que es diferente de una anomalía de JVM o de un apagón.

Pérdida de quórum debida a una anomalía de JVM

Un servidor de catálogo que falla provocará que se pierda quórum. En este caso, el quórum se debe alterar temporalmente tan rápidamente como sea posible. El servicio de catálogo anómalo no se pueden volver a unir a la cuadrícula hasta que se haya alterado temporalmente el quórum.

Pérdida de quórum debida a una caída de red

WebSphere eXtreme Scale se ha diseñado para esperar la posibilidad de apagones o bajadas de tensión. Una bajada de tensión es cuando se produce una pérdida temporal de conectividad entre centros de datos. Normalmente, tiene una naturaleza temporal y las caídas de tensión se suelen solucionar en cuestión de segundos o minutos. Mientras que WebSphere eXtreme Scale intenta mantener la operación normal durante el periodo de la bajada de tensión, este periodo se considera como un único suceso de anomalía. Se espera que se arregle la anomalía y que se reanude la operación normal sin ninguna acción necesaria de WebSphere eXtreme Scale.

Una bajada de tensión de larga duración se puede clasificar como un apagón sólo a través de la intervención del usuario. Es necesario alterar temporalmente el quórum en un lado de la bajada de tensión para que el suceso se clasifique como un apagón.

Ciclos de la JVM del servicio de catálogo

Si se detiene el servidor de catálogo mediante el uso de stopOgServer, el quórum pierde un servidor. Esto significa que los servidores restantes sigan teniendo quórum. Reiniciar el servidor de catálogo devuelve al quórum el número anterior.

Consecuencias de la pérdida de quórum

Si una JVM del contenedor falló mientras se perdió el quórum, la recuperación no se producirá hasta que se recupera la bajada de tensión, o en el caso de un apagón, el cliente realiza un mandato de alteración temporal de quórum. WebSphere eXtreme Scale considera un suceso de pérdida de quórum y una anomalía de contenedor como una anomalía doble, que es un suceso raro. Esto significa que las aplicaciones podrían perder el acceso de escritura a los datos que se almacenaron en la JVM anómala hasta que se restaure el quórum en el que tendrá lugar la recuperación normal de tiempo.

De forma similar, si intenta iniciar un contenedor durante un suceso de pérdida de quórum, el contenedor no se iniciará.

La conectividad total de cliente está autorizada durante la pérdida de quórum. Si no se produce ninguna anomalía de contenedor, ni ningún problema de conectividad durante el suceso de pérdida de quórum, los clientes pueden seguir interactuando de forma completa con los servidores de contenedor.

Si se produce una bajada de tensión, algunos clientes podrían no tener acceso a primarios o a copias de réplica de los datos hasta que se solucione la bajada.

Se pueden iniciar nuevos clientes, ya que debe haber una JVM de servicio de catálogo en cada centro de datos de forma que, como mínimo, un cliente pueda alcanzar una JVM del servicio de catálogo durante un suceso de bajada de tensión.

Recuperación del quórum

Si el quórum se pierde por algún motivo, cuando se restablece, se ejecuta un protocolo de recuperación. Cuando se produce un suceso de pérdida de quórum, se suspenden todas las comprobaciones de integridad y concurrencia para los grupos principales y también se ignoran los informes de anomalías. Una vez recuperado el quórum, el servicio de catálogo realiza una comprobación de integridad y concurrencia de todos los grupos principales para determinar inmediatamente su pertenencia. Cualquier fragmento alojado anteriormente o JVM de contenedor indicada como anómala se recuperará en este momento. Si se perdieron los fragmentos primarios se ascenderán las réplicas supervivientes. Si se perdieron los fragmentos de réplica, se crearán réplicas adicionales en los supervivientes.

Alteración temporal del quórum

Esto sólo se debe utilizar si se ha producido una anomalía del centro de datos. La pérdida de quórum debida a una anomalía de JVM de servicio de catálogo o a una bajada de tensión de la red se deberá recuperar automáticamente una vez que se reinicie la JVM de servicio de catálogo o que se solucione la bajada de tensión de la red.

Los administradores son los únicos que conocen la anomalía de un centro de datos. WebSphere eXtreme Scale trata una bajada de tensión y un apagón, de forma similar. Debe informar al entorno de eXtreme Scale de dichas anomalías utilizando el mandato xsadmin para alterar temporalmente el quórum. Esto indicará al

servicio de catálogo que presuponga que dicho quórum se ha conseguido con la pertenencia actual y se llevará a cabo una recuperación completa. Cuando se emite un mandato de alteración temporal de quórum, está garantizando que las JVM del centro de datos anómalo han fallado verdaderamente y no se recuperarán.

La siguiente lista considera algunos escenarios para alterar temporalmente el quórum. Suponga que tiene tres servidores de catálogo: A, B y C.

- Caída de tensión: suponga que ha sufrido una caída de tensión en la que C se ha aislado temporalmente. El servicio de catálogo perderá quórum y esperará a que se solucione la caída de tensión en el punto en el que C se vuelve a unir en el dominio del servicio de catálogo y se restablece el quórum. La aplicación no verá ningún problema durante este momento.
- Anomalía temporal: aquí C falla y el servicio de catálogo pierde quórum, de forma que debe alterar temporalmente el quórum. Una vez que se restablece el quórum, C se puede reiniciar. C se volverá a unir al dominio del servicio de catálogo cuando se reinicie. La aplicación no verá ningún problema durante este periodo de tiempo.
- Anomalía del centro de datos: verifique que el centro de datos ha fallado realmente y que se ha aislado en la red. Emita el mandato xsadmin de alteración temporal de quórum. Los dos centros de datos supervivientes realizarán una recuperación completa sustituyendo los fragmentos que estaban alojados en el centro de datos anómalo. Ahora, el servicio de catálogo se ejecuta con un quórum completo de A y B. La aplicación podría ver retardos o excepciones durante el intervalo entre el inicio del apagón y cuando se altera temporalmente el quórum. Una vez que se ha alterado temporalmente el quórum, la cuadrícula se recupera y se reanuda la operación normal.
- Recuperación de centro de datos: los centros de datos supervivientes ya se están ejecutando con el quórum alterado temporalmente. Cuando se reinicia el centro de datos que contiene C, todas las JVM del centro de datos se deben reiniciar. C se volverá a unir al dominio del servicio de catálogo existente y el quórum volverá a la situación normal sin ninguna intervención de usuario.
- Anomalía de centro de datos y caída de la tensión: el centro de datos que contiene C falla. El quórum se ha alterado temporalmente y se ha recuperado en los centros de datos restantes. Si se produce una caída de tensión entre A y B, se aplican las reglas de recuperación normal de caída de tensión. Una vez que se soluciona la caída de tensión, el quórum se restablece y se produce la recuperación necesaria de la pérdida de quórum.

Comportamiento de contenedor

Esta sección describe cómo se comportan las JVM de servidor de contenedor mientras se ha perdido y recuperado el quórum.

Los contenedores alojan uno o más fragmentos. Los fragmentos son primarios o réplicas para una partición específica. El servicio de catálogo asigna fragmentos a un contenedor y el contenedor otorgará dicha asignación hasta que se reciban nuevas instrucciones del servicio de catálogo. Esto significa que si un fragmento primario de un contenedor no se puede comunicar con un fragmento de réplica debido a una caída de tensión, seguirá intentándolo hasta que reciba nuevas instrucciones del servicio de catálogo.

Si se produce una caída de red y un fragmento primario pierde la comunicación con la réplica, volverá a intentar la conexión hasta que el servicio de catálogo proporcione nuevas instrucciones.

Comportamiento de réplica síncrona

Mientras la conexión está rota, el primario puede aceptar nuevas transacciones siempre que haya, como mínimo, tantas réplicas en línea como haya definido la propiedad minsync para el conjunto de correlaciones. Si se procesa alguna transacción nueva en el primario mientras que el enlace con la réplica síncrona está roto, la réplica se borrará y se volverá a sincronizar con el estado actual del primario cuando se restablezca el enlace.

La réplica síncrona está totalmente desaconsejada entre los centros de datos o en un enlace de estilo WAN.

Comportamiento de réplica asíncrona

Mientras la conexión está rota, el primario puede aceptar nuevas transacciones. El primario guardará en el almacenamiento intermedio los cambios hasta un límite. Si la conexión con la réplica se restablece antes de que se alcance el límite, la réplica se actualiza con los cambios del almacenamiento intermedio. Si se ha alcanzado el límite, el primario destruye la lista del almacenamiento intermedio y cuando se vuelve a conectar la réplica, se borra y se vuelve a sincronizar.

Comportamiento del cliente

Los clientes siempre se pueden conectar al servidor de catálogo para realizar el programa de arranque de la cuadrícula, independientemente de que el dominio del servicio de catálogo tenga o no quórum. El cliente intentará conectarse a cualquier instancia de servidor de catálogo para obtener una tabla de direccionamiento e interactuar con la cuadrícula. La conectividad de red puede impedir al cliente interactuar con algunas particiones debido a la configuración de la red. El cliente se puede conectar a las réplicas locales para los datos remotos si se ha configurado para esto. Los clientes no podrán actualizar los datos, si la partición primaria para dichos datos no está disponible.

Mandatos de quórum con xsadmin

Esta sección describe los mandatos xsadmin prácticos para las situaciones de quórum.

Consulta de estado de quórum

El estado del quórum de una instancia de servidor de catálogo se puede interrogar a través del mandato xsadmin.

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

Existen cinco resultados posibles.

- El quórum está inhabilitado: los servidores de catálogo se ejecutan en una modalidad de quórum inhabilitado. Se trata de una modalidad de desarrollo o de un solo centro de datos único. No se recomienda para las configuraciones de varios centros de datos.
- El quórum está habilitado y el servidor de catálogo tiene quórum: el quórum está habilitado y el sistema funciona normalmente.
- El quórum está habilitado pero el servidor de catálogo está esperando al quórum: el quórum está habilitado y se ha perdido.

- El quórum está habilitado y se ha alterado temporalmente: el quórum está habilitado y se ha alterado temporalmente.
- El estado del quórum no está autorizado: cuando se produce una caída de tensión, dividiendo el servicio de catálogo en dos particiones, A y B. El servidor de catálogo A ha alterado temporalmente el quórum. La partición de la red se resuelve y el servidor de la partición B no está autorizado, lo que requiere un reinicio de la JVM. También se produce si la JVM del catálogo en B se reinicia durante la caída de tensión y ésta se soluciona.

Alteración temporal del quórum

El mandato xsadmin se puede utilizar para alterar temporalmente el quórum. Se puede utilizar cualquier instancia de servidor de catálogo superviviente. Se notificará a todos los supervivientes que alteren temporalmente el quórum. La sintaxis de esto es la siguiente.

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

Mandatos de diagnóstico

- Estado de quórum: tal como se ha descrito en la sección anterior.
- Lista de grupos principales: ésta visualiza una lista de todos los grupos principales. Se visualizan los miembros y líderes de los grupos principales.

  ```
  xsadmin –ch cathost –p 1099 –coregroups
  ```
- Eliminación de servidores: este mandato elimina un servidor manualmente de la cuadrícula. Normalmente esto no es necesario puesto que los servidores se eliminan automáticamente cuando se ha detectado que han fallado, pero el mandato se proporciona para ser utilizado bajo la ayuda del soporte de IBM.

  ```
  xsadmin –ch cathost –p 1099 –g Grid –teardown server1,server2,server3
  ```
- Visualizar la tabla de direccionamiento: este mandato muestra la tabla de direccionamiento actual simulando una nueva conexión de cliente con la cuadrícula. También valida la tabla de direccionamiento confirmando que todos los servidores de contenedor reconocen su rol en la tabla de direccionamiento como, por ejemplo, qué tipo de fragmento para qué partición.

  ```
  xsadmin –ch cathost –p 1099 –g myGrid -routetable
  ```
- visualizar fragmentos no asignados: si algunos fragmentos no se pueden colocar en la cuadrícula, esto se puede utilizar para listarlos. Esto sólo sucede cuando el servicio de colocación tiene una limitación que impide la colocación. Por ejemplo, si inicia las JVM en una sola máquina física, mientras está en la modalidad de producción, sólo se pueden colocar los fragmentos primarios. Las réplicas estarán sin asignar hasta que las JVM se inicien en una segunda máquina. El servicio de colocación sólo coloca réplicas en las JVM con distintas direcciones IP que las JVM que alojan los fragmentos primarios. No tener ninguna JVM en una zona también puede provocar que los fragmentos se queden sin asignar.

  ```
  xsadmin –ch cathost –p 1099 –g myGrid –unassigned
  ```
- Establecer valores de rastreo: este mandato establece los valores de rastreo para todas las JVM que coinciden con el filtro especificado para el mandato xsadmin. Este valor sólo cambia los valores de rastreo hasta que se utiliza otro mandato o hasta que fallan o se detienen las JVM modificadas.

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

  Esto habilita el rastreo para todas las JVM de la máquina con el nombre de sistema principal especificado, en este caso host1.

- Comprobación de tamaños de correlación: el mandato de los tamaños de correlación es útil para verificar que la distribución de claves es uniforme en los fragmentos de la clave. Si algunos contenedores tienen más claves de forma significativa que los otros, es probable que la función hash en los objetos clave tenga una pobre distribución.

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

Consideraciones sobre la seguridad del transporte

Puesto que los centros de datos suelen desplegarse en ubicaciones geográficas diferentes, es posible que los usuarios deseen habilitar la seguridad del transporte entre los centros de datos por motivos de seguridad.

Lea la información sobre la seguridad de la capa de transporte en la *Guía de administración*.

Réplicas y fragmentos

Con eXtreme Scale, una base de datos o fragmento en memoria puede duplicarse desde una Máquina virtual Java (JVM) en otra. Un fragmento representa una partición que se coloca en un contenedor. En un contenedor pueden existir varios fragmentos que representan distintas particiones. Cada partición tiene una instancia que es un fragmento primario y un número configurable de fragmentos de réplica. Los fragmentos de réplica son síncronos o asíncronos. Los tipos y la ubicación de los fragmentos de réplica los determina eXtreme Scale mediante una política de despliegue, que especifica el número mínimo y máximo de los fragmentos síncronos y asíncronos.

Tipos de fragmento

La réplica utiliza tres tipos de fragmentos:

- Fragmento primario
- Réplica síncrona
- Réplica asíncrona

El fragmento primario recibe todas las operaciones de inserción, actualización y eliminación. El fragmento primario añade y elimina réplicas, duplica datos en las réplicas y gestiona confirmaciones y retrotracciones de transacciones.

Las réplicas síncronas mantienen el mismo estado que el fragmento primario. Cuando un fragmento primario duplica datos en una réplica síncrona, la transacción no se confirma hasta que se confirma en la réplica síncrona.

Las réplicas asíncronas pueden o no estar en el mismo estado que el fragmento primario. Cuando un fragmento primario duplica datos en una réplica asíncrona, el fragmento primario no espera a que la réplica asíncrona se confirma.

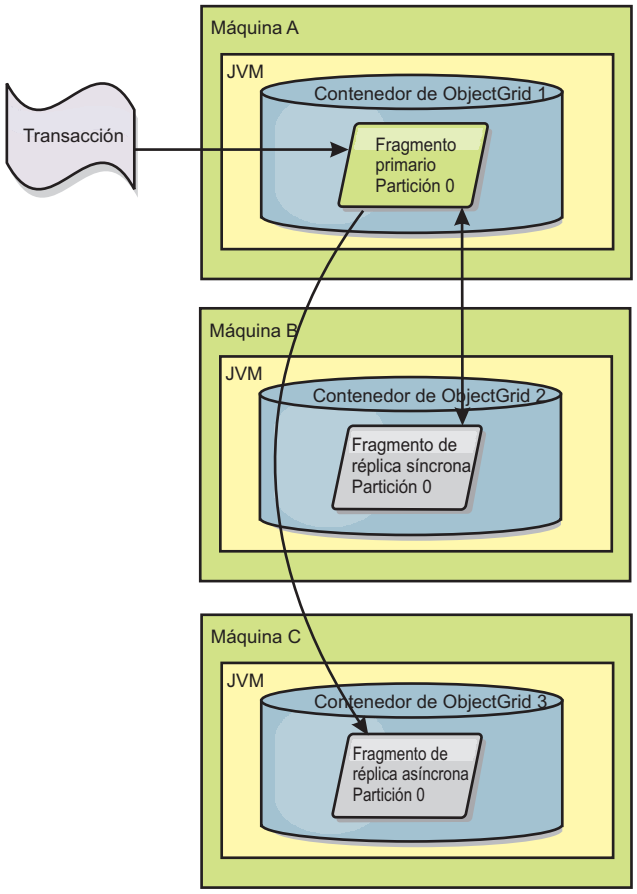


Figura 34. Vía de acceso de comunicación entre un fragmento primario y fragmentos de réplica

Fragmentos mínimos de réplicas síncronas

Cuando un fragmento primariose prepara para confirmar datos, comprueba cuántos fragmentos de réplicas síncronas han votado por confirmar la transacción. Si la transacción normalmente se procesa en la réplica, vota por confirmarse. Si se ha producido algún error en la réplica síncrona, se vota por no confirmarse. Antes de que un fragmento primario se confirme, el número de fragmentos de réplicas síncronas que votan por confirmarse deben satisfacer el valor minSyncReplica en la política de despliegue. Cuando el número de fragmentos de réplicas síncronas que votan por confirmarse es demasiado bajo, el fragmento primario no confirma la transacción y resulta en un error. Esta acción garantiza que la cantidad necesaria de réplicas síncronas esté disponible con los datos correctos. Las réplicas síncronas que han encontrado errores se han vuelto a registrar para corregir su estado. Si desea más información sobre cómo volver a realizar el registro, consulte Recuperación del fragmento de réplica.

El fragmento primario genera un error ReplicationVotedToRollbackTransactionException si muy pocas réplicas síncronas han votado por confirmarse.

Réplica y cargadores

Normalmente, un fragmento primario graba de forma síncrona en una base de datos los cambios a través del cargador. El cargador y la base de datos siempre

están sincronizados. Cuando el fragmento primario realiza una migración tras error en un fragmento de réplica, es posible que la base de datos y el cargador no estén sincronizados. Por ejemplo:

- El fragmento primario puede enviar la transacción a la réplica y luego sufrir una anomalía antes de confirmarse en la base de datos.
- El fragmento primario puede confirmarse en la base de datos y luego sufrir una anomalía antes de enviar la réplica.

Los dos enfoques llevan a que la réplica esté una transacción por delante o por detrás de la base de datos. Esta situación no es aceptable. eXtreme Scale utiliza un protocolo especial y un contrato con la implementación de cargador para resolver esta cuestión sin la confirmación de dos fases. El protocolo es el siguiente:

Lado del fragmento primario

- Enviar la transacción junto con resultados de transacciones anteriores.
- Grabar en la base de datos e intentar confirmar la transacción.
- Si la base de datos se confirma, confirmar en eXtreme Scale. Si la base de datos no está confirmada, retrotraer la transacción.
- Grabar el resultado.

Lado de la réplica

- Recibir una transacción y almacenarla en el almacenamiento intermedio.
- Para todos los resultados, enviar con la transacción, confirmar todas las transacciones almacenadas en el almacenamiento intermedio y descartar todas las transacciones retrotraídas.

Lado de réplica en la migración tras error

- Para todas las transacciones almacenadas en el almacenamiento intermedio, proporcionar las transacciones para el cargador y éste intenta confirmar las transacciones.
- Es necesario grabar el cargador para que cada transacción sea idempotente.
- Si la transacción ya está en la base de datos, el cargador no realizar ninguna operación.
- Si la transacción no está en la base de datos, el cargador aplica la transacción.
- Una vez que se han procesado todas las transacciones, el nuevo fragmento primario puede empezar a atender a solicitudes.

Este protocolo garantiza que al base de datos está en el mismo nivel que el estado del nuevo fragmento primario.

Asignación de fragmentos: primarios y réplicas

El servicio de catálogo se ocupa de colocar los fragmentos. Cada ObjectGrid tiene varias particiones, cada una de las cuales tiene un fragmento primario y un conjunto opcional de fragmentos réplica. El servicio de catálogo no coloca fragmentos primarios y réplica de la misma partición en el mismo contenedor. Tampoco coloca fragmentos primarios y réplica en contenedores con la misma dirección IP (a no ser que la configuración esté en modalidad de desarrollo). El servicio de catálogo asigna los fragmentos equilibrándolos de forma que se distribuyan uniformemente en los contenedores disponibles.

Si se inicia un nuevo contenedor, eXtreme Scale recupera los fragmentos de los contenedores relativamente sobrecargados para colocarlos en el nuevo contenedor

vacío. Con este comportamiento, eXtreme Scale establece y mantiene su elasticidad básica. La elasticidad se manifiesta en su potente capacidad por realizar escaladas de forma horizontal, tanto para la escalada horizontal como la vertical.

Escalar hacia fuera

Escalar hacia fuera significa que cuando se añaden Máquina virtual Java o contenedores adicionales a una cuadrícula de eXtreme Scale, eXtreme Scale intenta mover los fragmentos existentes, primario o réplicas, del antiguo conjunto de las JVM al nuevo conjunto. Este movimiento permite que la cuadrícula se amplíe y se aprovechan así el procesador, la red y la memoria de las JVM recién añadidas. El movimiento también equilibra el fragmento e intenta garantizar que cada JVM de la cuadrícula contiene la misma cantidad de datos. Cuando la cuadrícula se amplía, cada servidor contiene un subconjunto menor de la cuadrícula total. eXtreme Scale presupone que los datos se distribuyen uniformemente entre las particiones. Esta ampliación favorece la operación de escalar hacia fuera.

Escalar hacia dentro

Escalar hacia dentro significa que si falla una JVM , eXtreme Scale intenta reparar el daño. Si la JVM donde se ha producido la anomalía tenía una réplica, eXtreme Scale sustituye la réplica perdida mediante la creación de una nueva réplica en una JVM superviviente. Si la JVM donde se ha producido la anomalía tenía un fragmento primario, eXtreme Scale busca la mejor réplica entre los supervivientes y promociona la réplica para que sea el nuevo fragmento primario. eXtreme Scale sustituye la réplica promocionada por una nueva réplica creada en los servidores restantes. Para mantener la escalabilidad, eXtreme Scale conserva el recuento de réplicas para las particiones, a medida que se producen anomalías en los servidores.

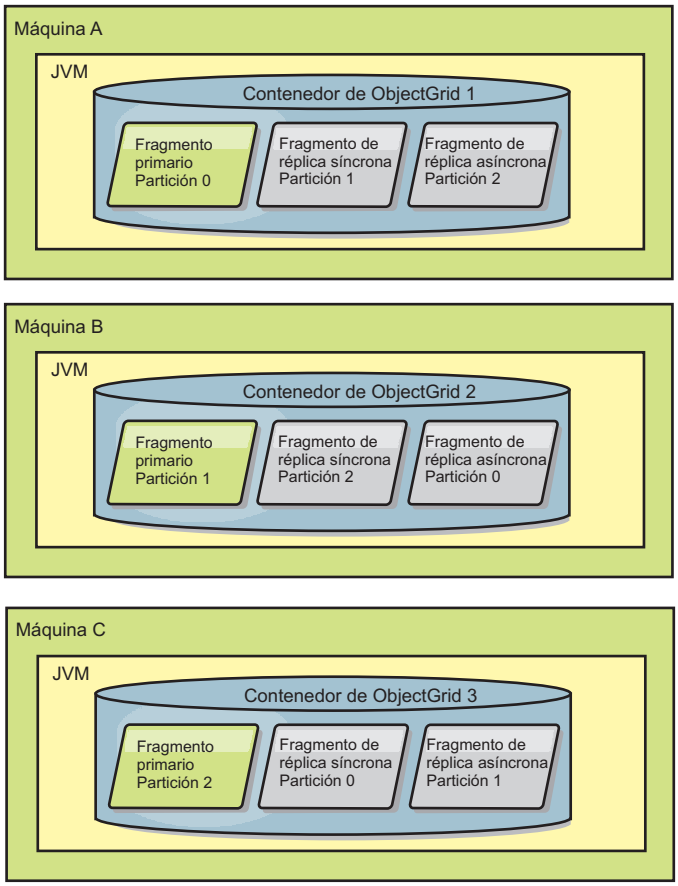


Figura 35. La colocación de un conjunto de correlaciones de ObjectGrid con una política de despliegue de 3 particiones con un valor minSyncReplicas de 1, un valor maxSyncReplicas de 1 y un valor maxAsyncReplicas de 1

Lectura de réplicas

Puede configurar conjuntos de correlaciones como, por ejemplo, que un cliente está autorizado para leer una réplica, en lugar de estar limitado sólo a los fragmentos primarios.

A menudo, puede resultar provechoso permitir a las réplicas actuar como más que unos primarios simplemente potentes en caso de anomalías. Por ejemplo, los conjuntos de correlaciones se pueden configurar para permitir que se direccionen operaciones de lectura a réplicas estableciendo la opción replicaReadEnabled de MapSet en true. El valor predeterminado es false.

Si desea más información sobre el elemento MapSet, consulte el tema sobre el archivo XML del descriptor de política de despliegue en *Guía de administración*.

La habilitación de lectura de réplicas puede mejorar el rendimiento distribuyendo las peticiones a más máquinas virtuales Java™. Si la opción no está habilitada, todas las peticiones de lectura como los métodos ObjectMap.get o Query.getResultIterator se direccionan al primario. Cuando replicaReadEnabled está definido en true, algunas peticiones get podrían devolver datos obsoletos, así pues una aplicación que utiliza esta opción debe poder tolerar esta posibilidad. Sin embargo, no se producirá ningún fallo de la memoria caché. Si los datos no están en la réplica, la petición get se direcciona al primario y se vuelve a intentar.

La opción replicaReadEnabled se puede utilizar tanto con la réplica síncrona como con la asíncrona.

Equilibrio de carga entre réplicas

El equilibrio de carga entre réplicas normalmente se utiliza sólo cuando los clientes almacenan en la memoria caché datos que almacenan siempre o cuando los clientes utilizan el bloqueo pesimista.

eXtreme Scale, salvo que se configure de otra manera, envía todas las solicitudes de lectura y grabación al servidor primario para un grupo de réplicas determinado. El fragmento primario debe atender todas las solicitudes de los clientes. Es posible que desee permitir que las solicitudes de lectura se envíen a las réplicas del fragmento primario. Si envía solicitudes de lectura a las réplicas la carga de las solicitudes de envío se podrá compartir entre varias JVM (Java Virtual Machines). No obstante, el uso de réplicas para las solicitudes de lectura puede generar repuestas incoherentes.

El equilibrio de carga entre réplicas normalmente se utiliza sólo cuando los clientes almacenan en la memoria caché datos que almacenan siempre o cuando los clientes utilizan el bloqueo pesimista.

Si lo datos se almacenan en la memoria caché constantemente y luego se invalidan en la memoria caché cercana del cliente, como resultado el fragmento primario debe detectar un índice de solicitudes get relativamente alto de los clientes. Asimismo, en modalidad de bloqueo pesimista, no existe memoria caché local, y por ello todas las solicitudes se envían al fragmento primario.

Si los datos son relativamente estáticos o si no se utiliza la modalidad pesimista, el envío de solicitudes de lectura a la réplica no tendrá un gran impacto en el rendimiento. La frecuencia de las solicitudes get de los clientes con memoria caché que están llenas de datos no es alta.

Cuando un cliente se inicia, su memoria caché cercana está vacía. Las solicitudes de memoria caché para la memoria caché vacía se remiten al fragmento primario. La memoria caché del cliente obtendrá datos con el tiempo, lo que provocará que la carga de solicitudes baje. Si una gran cantidad de clientes se inicia simultáneamente, la carga puede ser significativa y la lectura de réplicas puede ser una opción de rendimiento apropiada.

Sucesos de ciclo de vida, recuperación y anomalía

Los fragmentos pasan por estados y sucesos diferentes para poder admitir operaciones de réplica. El ciclo de vida de un fragmento incluye el paso a estar en línea, el tiempo de ejecución, la conclusión, la migración tras error y el manejo errores. Los fragmentos se pueden trasladar de un fragmento de réplica a un fragmento primario para manejar los cambios del estado del servidor.

Sucesos de ciclo de vida

Cuando se colocan y se inician los fragmentos réplica, pasan por una serie de sucesos hasta llegar a estar en línea y en modalidad de escucha.

Fragmento primario

El servicio de catálogo coloca un fragmento primario para una partición. El servicio de catálogo también equilibra las ubicaciones de los fragmentos primarios y e inicia la sustitución por anomalía para fragmentos primarios.

Cuando un fragmento se convierte en un fragmento primario, recibe una lista de réplicas del servicio de catálogo. El fragmento primario nuevo crea un grupo de réplicas y las registra.

Cuando el fragmento primario está listo, se muestra un mensaje de listo para operaciones empresariales en el archivo `SystemOut.log` del contenedor donde se esté ejecutando. El mensaje abierto o el mensaje CWOBJ1511I, lista el nombre de la correlación, el nombre del conjunto de correlaciones y el número de partición del fragmento primario que se ha iniciado.

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

Consulte el apartado "Asignación de fragmentos: primarios y réplicas" en la página 133 para obtener más información sobre cómo coloca fragmentos el servicio de catálogo.

Fragmento réplica

Los fragmentos réplica los controla principalmente el fragmento primario a no ser que la réplica detecte un problema. Durante un ciclo de vida normal, el fragmento primario coloca, registra y elimina el registro de un fragmento de réplica.

Cuando el fragmento primario inicializa un fragmento réplica, un mensaje muestra el archivo de anotaciones cronológicas que describe dónde se ejecuta la réplica para indicar que el fragmento réplica está disponible. El mensaje abierto, o el mensaje CWOBJ1511I, lista el nombre de correlación, el nombre del conjunto de correlaciones y el número de partición del fragmento de réplica. Este mensaje es el siguiente:

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

o bien

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

Fragmento réplica asíncrono: un fragmento réplica asíncrono sondea el fragmento primario para obtener los datos. La réplica ajustará automáticamente la temporización del sondeo si no recibe los datos del fragmento primario, que indica que se ha puesto al día con el fragmento primario. Se ajustará también si recibe un error que podría indicar que se ha producido un error en el fragmento primario o si hay un problema de red.

Cuando la réplica asíncrona comienza la réplica, imprime el mensaje siguiente en el archivo SystemOut.log para la réplica. Este mensaje podría imprimirse más de una vez por mensaje CWOBJ1511. Se imprimirá de nuevo si la réplica se conecta a un fragmento primario distinto o si se han añadido correlaciones de plantilla.

```
CWOBJ1543I: Se ha iniciado la réplica asíncrona objectGridName:mapsetName:partitionNumber o
ha seguido realizando la réplica desde el fragmento principal. Réplica de correlaciones: [mapName]
```

Fragmento réplica síncrono: cuando se inicia por primera vez el fragmento réplica asíncrono, no está todavía en modalidad de igual. Cuando un fragmento réplica está en modalidad de igual, recibe datos del fragmento primario a medida que los datos van entrando en el fragmento primario. Antes de pasar a la modalidad de igual, el fragmento réplica necesita una copia de todos los datos existentes en el fragmento primario.

La réplica síncrona copia los datos del fargmento primario similar a una réplica asíncrona sondeando los datos. Cuando copia los datos existentes del fragmento primario, cambia a modalidad de igual y comienza a recibir los datos a medida que el fragmento primario recibe los datos.

Cuando un fragmento réplica alcanza la modalidad de igual, imprime un mensaje en el archivo SystemOut.log de la réplica. El tiempo se refiere a la cantidad de tiempo que tardó el fragmento réplica en obtener todos los datos iniciales del fragmento primario. El valor de tiempo puede mostrarse como cero o muy bajo si el fragmento primario no tiene ningún dato que deba replicar.Puede que este mensaje se imprima más de una vez por mensaje CWOBJ1511. Se imprimirá de nuevo si la réplica se conecta a un fragmento primario distinto o si se han añadido correlaciones de plantilla.

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

Cuando el fragmento de réplica síncrono está en modalidad de igual, el fragmento primario debe hacer una réplica de las transacciones a todas las réplicas síncronas de modalidad de igual. Los datos del fragmento réplica síncrono permanecen al mismo nivel que los del fragmento primario. Si se establece un número mínimo de réplicas síncronas o minSync en la política de despliegue, ese número de réplicas síncronas debe votar confirmarse antes de que la transacción pueda confirmarse satisfactoriamente en el fragmento principal.

Sucesos de recuperación

Las operaciones de réplica se han diseñado para realizar recuperaciones a partir de sucesos de anomalías y errores. Si se produce una anomalía en un fragmento primario, otra réplica pasa a tener el control. Si las anomalías se producen en los fragmentos réplica, éstos intentarán recuperarse. El servicio de catálogo controla la colocación y las transacciones de fragmentos primarios nuevos o fragmentos réplica nuevos.

Un fragmento réplica se convierte en un fragmento primario

Un fragmento réplica se convierte en un fragmento primario por dos razones: el fragmento primario se ha detenido o ha fallado, o se ha realizado una decisión de equilibrio para mover el fragmento primario anterior a una nueva ubicación.

El servicio de catálogo selecciona un nuevo fragmento primario de los fragmentos réplica síncronos existentes. Si debe tener lugar un movimiento de fragmento primario y no hay réplicas, se colocará una réplica temporal para completar la transición. El fragmento primario nuevo registra todas las réplicas existentes y acepta transacciones como el nuevo fragmento primario. Si los fragmentos réplica existentes tienen el nivel correcto de datos, los datos actuales se conservan a medida que los fragmentos réplica se registran con el nuevo fragmento primario. Se sondearán las réplicas asíncronas con el nuevo fragmento primario.

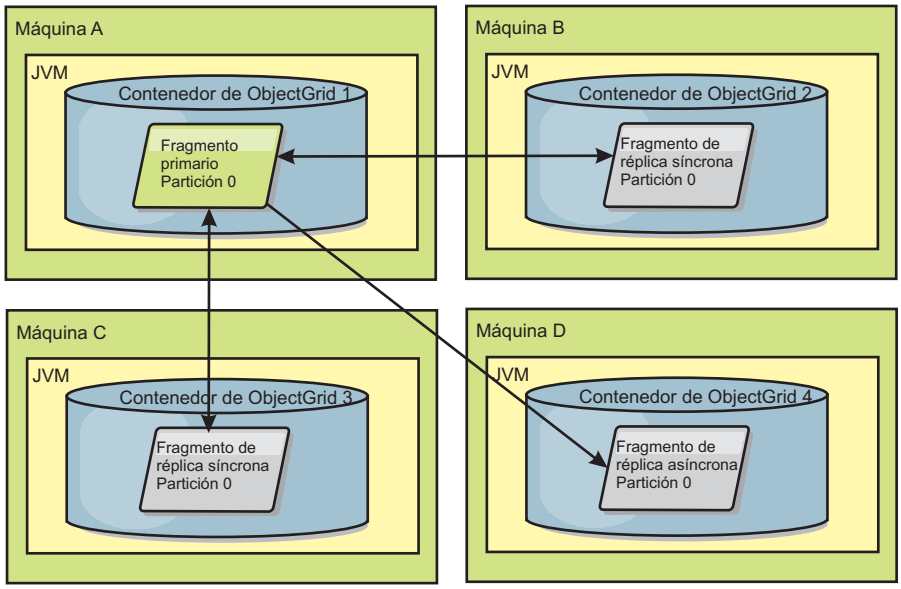


Figura 36. Colocación de ejemplo de un conjunto de correlaciones ObjectGrid para la partición partition0. La política del despliegue tiene un valor minSyncReplicas de 1, un valor maxSyncReplicas de 2 y un valor maxAsyncReplicas de 1.

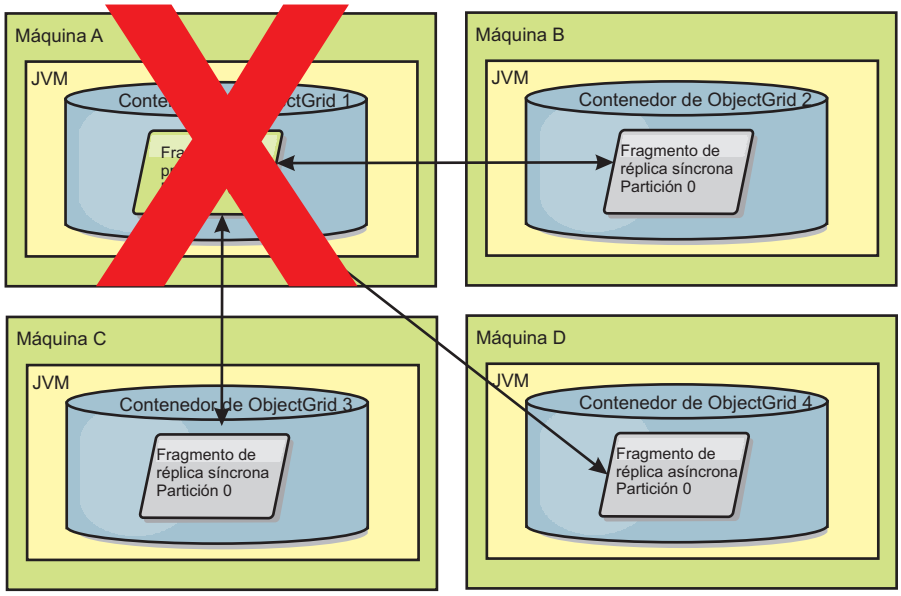


Figura 37. El contenedor del fragmento primario falla.

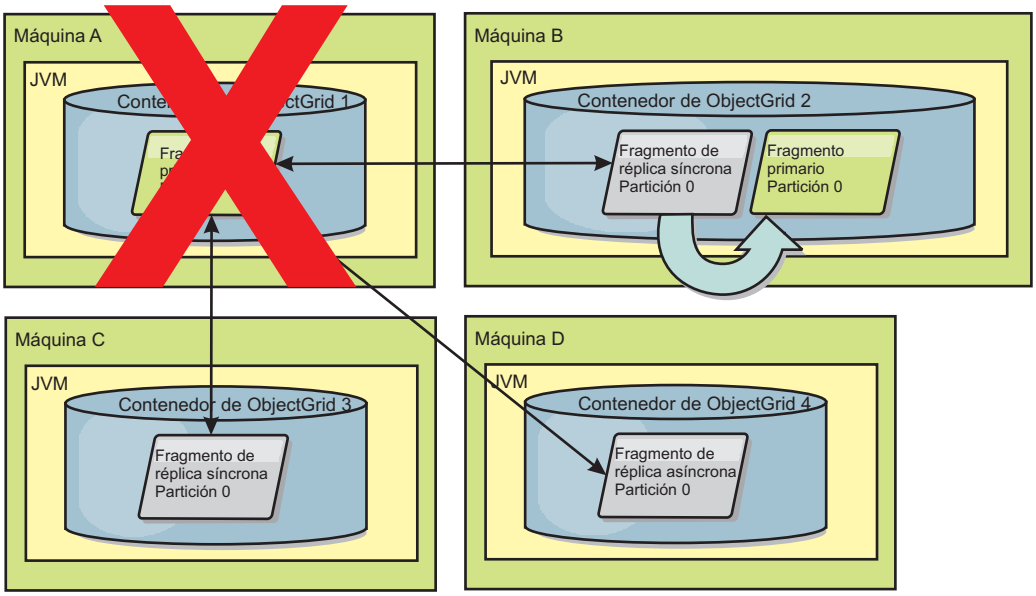


Figura 38. El fragmento de réplica síncrona en el contenedor 2 de ObjectGrid pasa a ser el fragmento primario.

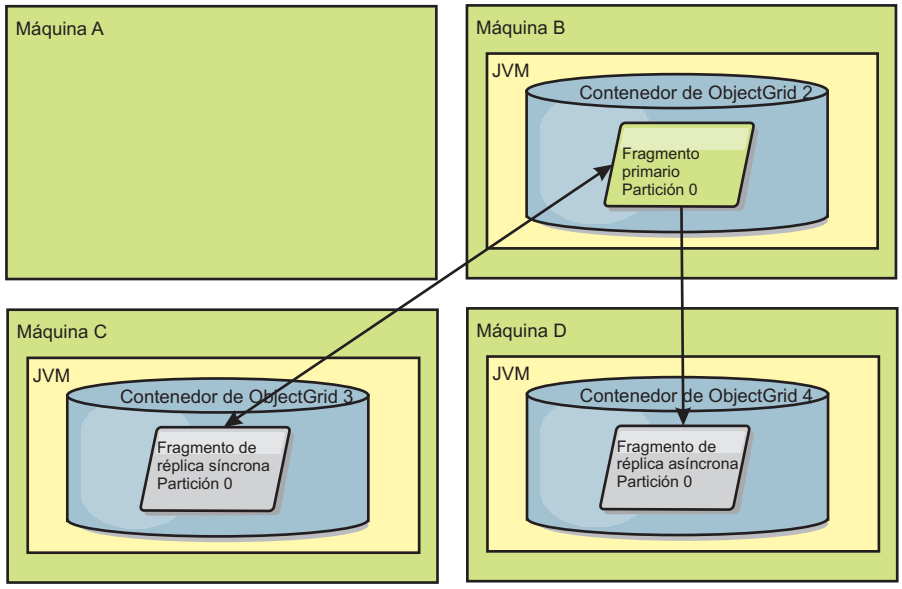


Figura 39. La máquina B contiene el fragmento primario. En función de cómo se establece la modalidad de reparación automática y la disponibilidad de los contenedores, un nuevo fragmento de réplica síncrona se podría colocar o no en una máquina.

Recuperación de un fragmento réplica

El fragmento primario controla al fragmento réplica síncrono. No obstante, si un fragmento réplica detecta un problema, puede desencadenar un suceso de volver a registrar para corregir el estado de los datos. La réplica borra los datos actuales y obtiene una nueva copia del fragmento primario.

Cuando un fragmento réplica inicia un suceso de volver a registrar, la réplica imprime un mensaje de anotaciones cronológicas.

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

Si una transacción provoca un error en un fragmento réplica durante el proceso, el fragmento réplica está en un estado desconocido. La transacción se ha procesado correctamente en el fragmento primario, pero se ha producido una anomalía en la réplica. Para corregir esta situación, la réplica inicia un suceso de volver a registrar. Con una nueva copia de los datos del fragmento primario, el fragmento réplica puede continuar. Si se vuelve a repetir el problema, el fragmento réplica no vuelve a registrarse de forma continuada. Si desea más información, consulte “Sucesos de errores”.

Sucesos de errores

Una réplica puede detener la réplica de datos si encuentra situaciones de error para las que no se puede recuperar la réplica.

Demasiados intentos de registro

Si una réplica desencadena un suceso de volver a registrar varias veces, pero no se confirman los datos correctamente, la réplica se detiene. Al detenerse, se evita que la réplica entre en un bucle infinito de sucesos de volver a registrarse. De manera predeterminada, un fragmento réplica intenta volver a registrarse tres veces seguidas antes de detenerse.

Si un fragmento réplica vuelve a registrarse demasiadas veces, imprimirá el siguiente mensaje en el archivo de anotaciones cronológicas.

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

Si la réplica no se recupera mediante operaciones de volver a registrarse, podría existir un problema generalizado con las transacciones relativas al fragmento réplica. Un problema posible podría ser que faltan recursos en la variable CLASSPATH si se produce un error al inflar las claves o valores de la transacción.

Anomalía al entrar en modalidad de igual

Si una réplica intenta entrar en modalidad de igual y se produce un error al procesar los datos en bloque del fragmento primario (datos del punto de control), la réplica concluye. Esto impide que la réplica se inicie con datos iniciales incorrectos. Puesto que recibe los mismos datos del primario si se vuelve a registrar, no se vuelve a intentar la réplica.

Si un fragmento réplica no puede entrar en modalidad de igual, imprimirá el siguiente mensaje en el archivo de anotaciones cronológicas:

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

En el archivo de anotaciones cronológicas se muestra otro mensaje que explica por qué la réplica no ha podido entrar en modalidad de igual.

Recuperación después de una anomalía al volver a registrarse o de la modalidad de igual

Si una réplica falla al volver a registrarse o al entrar en la modalidad de igual, la réplica está en un estado inactivo hasta que se produce un nuevo suceso de colocación. Un nuevo suceso de colocación podría ser un nuevo servidor que se

inicia o detiene. También puede iniciar un suceso de colocación utilizando el método triggerPlacement en el MBean PlacementServiceMBean.

Direccionamiento a zonas según preferencias

Con el direccionamiento a zonas según preferencia, eXtreme Scale puede direccionar las transacciones a las zonas basándose en las especificaciones indicadas.

WebSphere eXtreme Scale le permite ejercer un control significativo sobre dónde se colocan los fragmentos de ObjectGrid.

El direccionamiento a zonas según preferencias permite a los clientes de eXtreme Scale especificar una tendencia para una zona o un conjunto de zonas determinado. Así pues, eXtreme Scale intentará direccionar las transacciones de cliente a las zonas preferidas antes de intentar direccionarlas a cualquier otra zona.

Requisitos para el direccionamiento a zonas según preferencias

Hay varios factores que se deben tener en cuenta antes de intentar el direccionamiento a zonas según preferencias. Asegúrese de que la aplicación puede satisfacer los requisitos de su escenario.

Es necesaria la ubicación de particiones por contenedor para poder sacar partido del direccionamiento a zonas según preferencias. Esta estrategia de colocación es muy adecuada para las aplicaciones que almacenan datos de sesión en ObjectGrid. La estrategia de colocación de particiones predeterminada de WebSphere eXtreme Scale es la partición fija. Las claves utilizan el código hash en el momento de confirmar una transacción para determinar qué partición alberga el par de clave-valor de la correlación cuando se utiliza la ubicación de partición fija.

La colocación por contenedor asigna los datos a una partición aleatoria en el momento de confirmar una transacción a través de SessionHandle. Debe poder reconstruir SessionHandle para recuperar los datos de ObjectGrid.

Puesto que puede utilizar zonas para tener más control sobre dónde residirán los primarios y su duplicaciones en el dominio, un despliegue de varias zonas es útil si los datos residen en varias ubicaciones físicas. Separar geográficamente los primarios y sus duplicaciones es una forma de asegurar que la pérdida catastrófica de 1 centro de datos no tendrá repercusiones en la disponibilidad de los datos.

Si los datos se distribuyen en una topología de varias zonas, es probable que los clientes también se extiendan a lo largo de la topología. El direccionamiento de clientes a su zona o centro de datos local tiene la ventana obvia de rendimiento de tener una latencia de red reducida. Aproveche este escenario, cuando sea posible.

Configuración de la topología para el direccionamiento a zonas según preferencias

Considere el siguiente escenario. Tiene 2 centros de datos: Chicago y Londres. Para minimizar el tiempo de respuesta de los clientes, desea que los clientes lean y escriban los datos en su centro de datos local.

Los fragmentos primarios de ObjectGrid se deben colocar en cada uno de los centros de datos para que se puedan grabar las transacciones de forma local desde

cada ubicación. De forma adicional, los clientes tendrán que conocer las zonas para poder realizar el direccionamiento a la zona local.

La colocación por contenedor localiza los fragmentos primarios nuevos en cada uno de los contenedores iniciados. Las réplicas se colocan de acuerdo con las reglas de zona y colocación especificadas por la política de despliegue. De forma predeterminada, se coloca una réplica en una zona que no es su zona primaria. Tenga en cuenta la siguiente política de despliegue para este escenario.

```
<?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>
```

Cada contenedor que se inicia con la política de despliegue recibirá 3 primarios nuevos. Cada primario tendrá una réplica asíncrona. Inicie cada contenedor con el nombre de zona apropiado. Utilice el parámetro -zone si va a iniciar los contenedores con el script startOgServer.

Para un servidor de contenedor de 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 los contenedores se ejecutan en WebSphere Application Server, debe crear un grupo de nodos y darle un nombre con el prefijo “ReplicationZone”. Los servidores que se ejecutan en los nodos de dichos grupos de nodos se colocarán en la zona apropiada. Por ejemplo, los servidores que se ejecutan en un nodo de Chicago podrían estar en un grupo de nodos llamado “ReplicationZoneChicago”.

Los primarios para la zona de Chicago tendrán réplicas en la zona de Londres. Los primarios para la zona de Londres tendrán réplicas para la zona de Chicago.

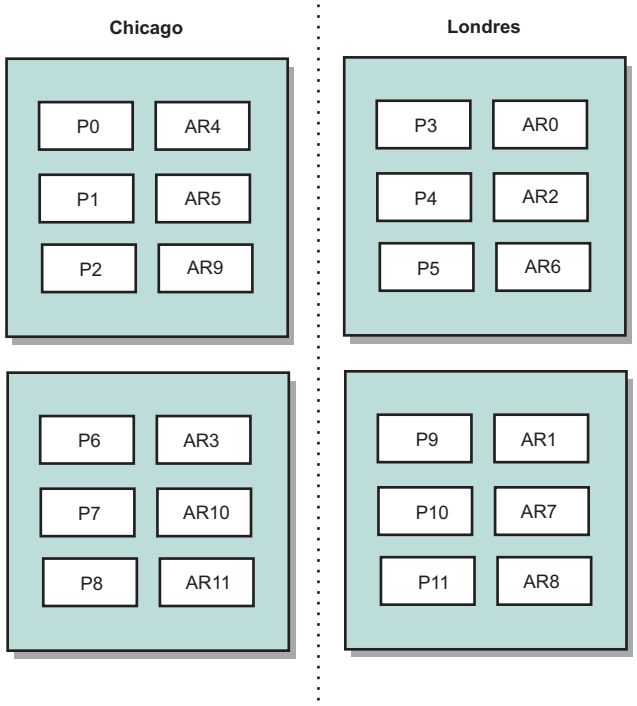


Figura 40. Primarios y réplicas en las zonas

Establezca las zonas preferidas para los clientes. Esta información se puede proporcionar de varias formas distintas. La forma más directa es proporcionar un archivo de propiedades de cliente a la JVM cliente. Cree un archivo llamado objectGridClient.properties y asegúrese de que está en su classpath. Si desea más información, consulte el tema sobre el archivo de propiedades de cliente en la *Guía de administración*.

Incluya la propiedad preferZones en el archivo. Establezca el valor de propiedad en la zona apropiada. Los clientes de Chicago deben tener lo siguiente en el archivo objectGridClient.properties.

```
preferZones=Chicago
```

El archivo de propiedades para los clientes de Londres debe contener

```
preferZones=London
```

Esta propiedad indica a cada cliente que direccione las transacciones a su zona local, si es posible. Los datos que se insertan en un primario de la zona local se duplicarán de forma asíncrona en la zona foránea.

Utilización de SessionHandle para realizar el direccionamiento a la zona local

La estrategia de colocación por contenedor no utiliza ningún algoritmo basado en hash para determinar la ubicación de los pares de clave-valor en ObjectGrid. ObjectGrid debe sacar partido a SessionHandles para asegurarse de que las transacciones se direccionan a la ubicación correcta cuando se utiliza esta estrategia de colocación. Cuando se confirma una transacción, se enlaza SessionHandle a una Session si todavía no se ha establecido ninguna. De forma alternativa, SessionHandle se puede enlazar a Session llamando a Session.getSessionHandle

antes de confirmar la transacción. El siguiente fragmento de código muestra un SessionHandle que se enlaza antes de confirmar la transacción.

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

// enlazando SessionHandle
SessionHandle sessionHandle = ogSession.getSessionHandle();

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

// la transacción se direcciona a la partición especificada por SessionHandle
ogSession.commit();
```

Suponga que el código anterior se ejecutaba en un cliente en el centro de datos de Chicago. Puesto que el atributo preferZones se ha establecido en Chicago para este cliente, esta transacción se direccionará a una de las particiones primarias de la zona de Chicago, la partición 0, 1, 2, 6, 7 o 8.

Este SessionHandle es la vía de vuelta a la partición que almacena estos datos confirmados. SessionHandle se debe volver a utilizar o reconstruir y establecer en la Session para poder volver a la partición que contiene los datos confirmados.

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

// el valor devuelto será "mercury "
String value = map.get("planet1");
ogSession.commit();
```

Puesto que esta transacción reutiliza el SessionHandle que se creó durante la transacción de insertar, la transacción de obtener se direccionará a la partición que contiene los datos insertados. Esta transacción no podrá recuperar los datos insertados previamente, si no se ha establecido el SessionHandle.

Cómo afectan los errores de contenedor y zona en el direccionamiento basado en zonas

Un cliente con la propiedad preferZones establecida tendrá todas sus transacciones direccionadas a la zona o zonas especificadas bajos circunstancias normales. Sin embargo, la pérdida de un contenedor generará una réplica que pasará a ser primaria en una zona foránea. Es posible que se asigne a la zona foránea un cliente que previamente direccionaba las transacciones a las particiones de la zona local para poder recuperar los datos insertados antes.

Considere el siguiente escenario. Se ha perdido un contenedor de la zona de Chicago. Previamente, contenía los primarios para las particiones 0, 1 y 2. Los nuevos primarios para estas particiones estarán en la zona de Londres puesto que esta zona contenía las réplicas de estas particiones.

Cualquier cliente de Chicago que utilice un SessionHandle que señale a una de las particiones que ha fallado ahora será direccionado a Londres. Los clientes de Chicago que utilizan nuevos SessionHandles serán direccionados a los primarios con base en Chicago.

De forma similar, si se pierde toda la zona de Chicago, todas las réplicas de la zona de Londres pasarán a ser primarios. En este caso, todos los clientes de Chicago tendrán sus transacciones direccionadas a Londres.

Topologías de réplica de cuadrícula con varios maestros (AP)

Con la característica réplica asíncrona con varios maestros, dos o más cuadrículas pueden pasar a ser duplicados exactos las unas de las otras. Esta duplicación se lleva a cabo con la réplica asíncrona entre enlaces que conectan juntas las cuadrículas. Cada cuadrícula se hospeda en un "dominio" completamente independiente, que procesa su propio servicio de catálogo, los servidores de contenedor y un nombre de dominio único. Con la característica réplica asíncrona con varios maestros, puede utilizar los enlaces para interconectar una colección de estos dominios y luego sincronizarlos, mediante la réplica en los enlaces.eXtreme Scale permite construir casi cualquier topología, porque la definición de enlaces entre dominios le corresponde a usted.

Dominios: cuadrículas con características específicas

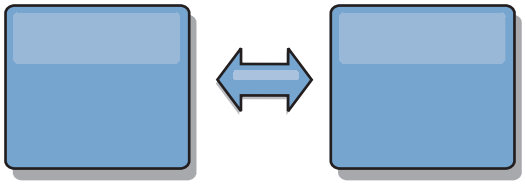
Las cuadrículas utilizadas en topologías de réplica con varios maestros se conocen como *dominios*. Todos los dominios deben tener estas características:

- Tener un servicio de catálogo privado con un nombre de dominio único
- Tener el mismo nombre de cuadrícula que otras cuadrículas del dominio
- Tener el mismo número de particiones que otras cuadrículas del dominio
- Ser una cuadrícula FIXED_PARTITION (no se puede hacer una réplica de las cuadrículas PER_CONTAINER)
- Tener el mismo número de particiones (podrían tener o no tener el mismo número y tipos de réplicas)
- Tener los mismos tipos de datos de los que se va a hacer una réplica que otras cuadrículas del dominio
- Tener los mismos nombres de conjunto de correlaciones, nombres de correlación y plantillas de correlación dinámica que otras cuadrículas del dominio

Después de que se han iniciado los dominios de la topología, se hará una réplica de todas las cuadrículas con las características anteriores. Recuerde que se omitirá su política de réplica.

Enlaces que conectan los dominios

Una infraestructura de cuadrícula de réplica es un gráfico de dominios conectados con enlaces bidireccionales entre ellos. Un enlace permite que dos dominios intercambien cambios de datos. Por ejemplo, la topología más sencilla es un par de dominios con un solo enlace entre ellos. Se nombran los dominios comenzando por una “A” y seguidamente una “B”, etc., desde la izquierda. En enlace podría atravesar una red de área amplia (WAN), que abarca largas distancias. Si se interrumpe el enlace, se pueden realizar todavía cambios en los datos en cualquier dominio. Más tarde se reconcilian los cambios, cuando el enlace se vuelve a conectar a los dominios. Los enlaces intentan volverse a conectar automáticamente si se interrumpe la conexión de red.

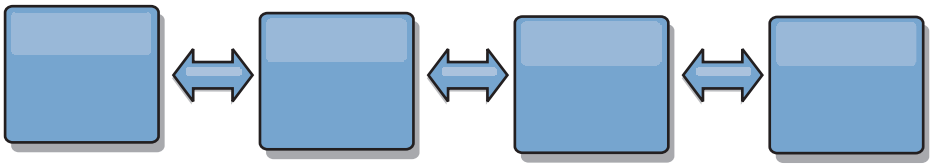


Después de haber configurado los enlaces, eXtreme Scale intentará primero hacer que cada dominio sea idéntico y luego intentará mantener las condiciones idénticas

cuando se produzcan cambios en algún dominio. El objetivo de eXtreme Scale es que cada dominio sea un duplicado exacto de cada otro dominio conectado mediante los enlaces. Los enlaces de réplica entre los dominios ayudan a garantizar que los cambios realizados en un dominio se copian en los otros dominios.

Topologías de línea

Aunque está entre las topologías más sencillas, una topología de línea muestra algunas cualidades de los enlaces. En primer lugar, no es necesario que un dominio esté conectado directamente a todos los demás dominios para recibir los cambios. El Dominio B extraerá los cambios del Dominio A. El Dominio C recibe los cambios del Dominio A a través del Dominio B, que conecta los Dominios A y C. De forma similar, el Dominio D recibe los cambios de los otros dominios a través del Dominio C. Esta capacidad esparce la carga de distribuir los cambios lejos del origen de los cambios.



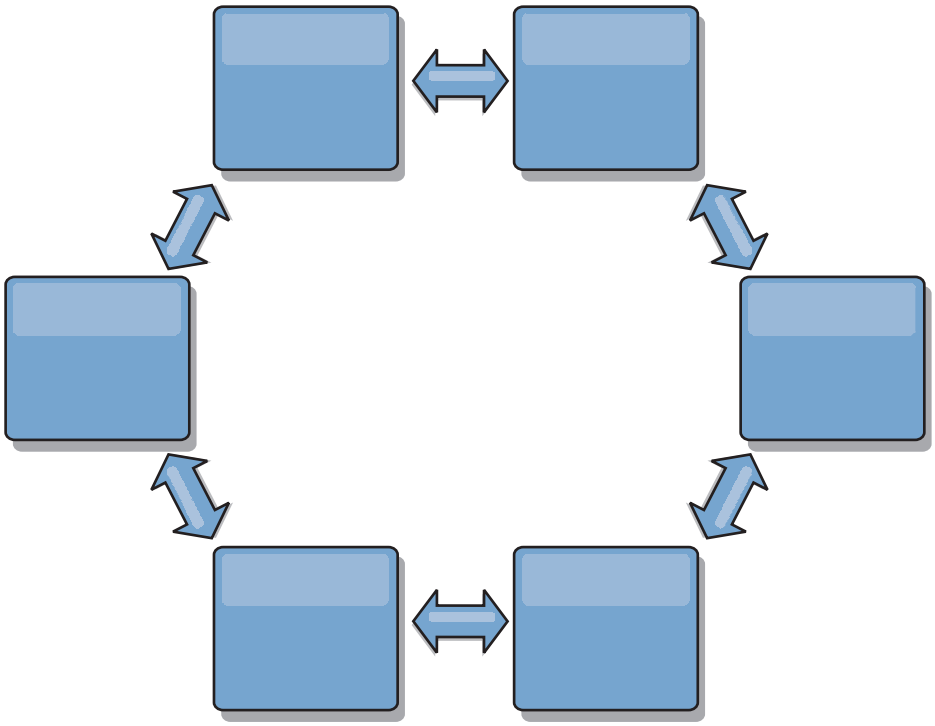
Recuerde que si se produce un error en el Dominio C, se producen los sucesos siguientes.

1. El Dominio D se quedará huérfano hasta que se reinicie el Dominio C
2. El Dominio C se sincronizará a sí mismo con el Dominio B, que es una copia del Dominio A
3. El Dominio D utilizará el Dominio C para sincronizarse a sí mismo con los cambios en los Dominios A y B producidos cuando el Dominio D estaba huérfano (cuando el Dominio C estaba inactivo)

Al final, los Dominios A, B, C y D serán idénticos entre sí de nuevo.

Topologías de anillo

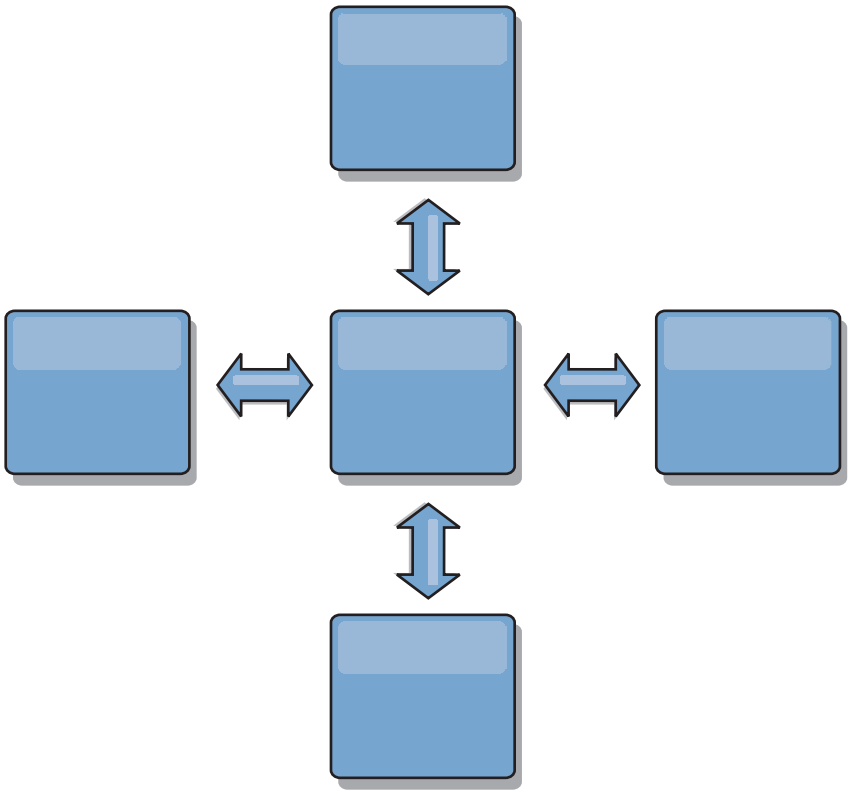
Las topologías de anillo son un ejemplo de una topología más flexible. Aunque un dominio o un enlace único puede producir un error, los dominios supervivientes todavía pueden obtener los cambios viajando por el anillo, lejos de la anomalía. Cada dominio tiene dos enlaces con los otros dominios. Cada dominio tiene a lo sumo dos enlaces, no importa lo grande que sea una topología de anillo. La latencia para propagar los cambios puede ser elevada, porque es posible que los cambios de un dominio en particular tengan que viajar por varios dominios antes de que todos los dominios vean los cambios. Una topología de línea tiene el mismo problema.



Representa una topología de anillo más sofisticada, con un dominio raíz en el dentro del anillo. El dominio raíz funciona como un centro de intercambio de información central, mientras que los demás dominios actúan como centros de intercambio de información remotos para los cambios que se producen en el dominio raíz. El dominio raíz puede arbitrar los cambios entre los dominios. Si una topología de anillo contiene más de un anillo alrededor de un dominio raíz, este último solo puede arbitrar los cambios entre los dominios del anillo más recóndito. No obstante, los resultados del arbitraje se diseminan a los dominios de los demás anillos.

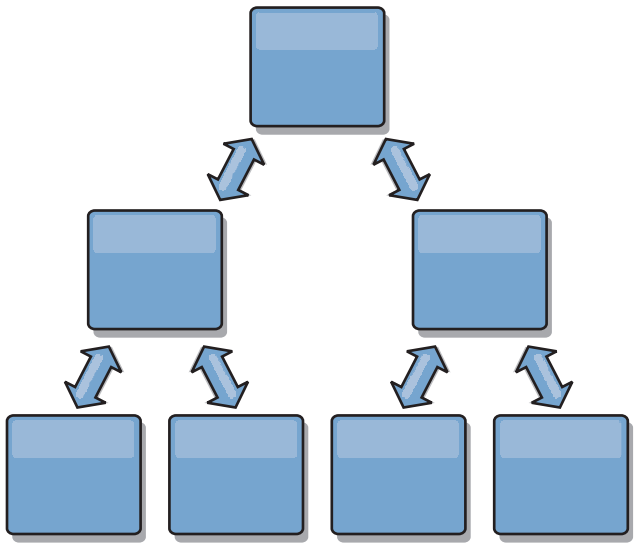
Topologías de hub y radio

Una topología de hub y radio tiene mejor latencia, lo que significa que los cambios viajan a lo sumo por un dominio intermedio (el hub), pero introduce algún otro problema. Tiene un dominio central que actúa como un hub. Este "dominio de hub" se conecta a todos los "dominios de radio" con un enlace. Claramente, la carga de distribuir los cambios entre los dominios recae sobre el hub. El hub actúa como un centro de intercambio de información para las colisiones, una configuración que puede ser importante para determinados casos. En un entorno con una alta tasa de actualizaciones, quizá sea necesario que el hub se ejecute en más hardware que los radios, así puede mantener el ritmo. eXtreme Scale está diseñado para escalar de forma lineal, lo que significa que puede ampliar el hub, según sea necesario, sin dificultad. No obstante, si el hub produce un error, no se distribuirán los cambios hasta que no se reinicie el hub. Los cambios en los dominios de radio se distribuirán después de que se vuelva a conectar el hub.



Topologías de árbol

Un último ejemplo de topología es un árbol dirigido acíclico. Acíclico significa que no hay ciclos ni bucles. Dirigido significa que existen los enlaces solo entre padres e hijos. Esta configuración puede resultar de utilidad para las topologías con tantos dominios que no es práctico tener un hub central conectado a todos los radios posibles, o en el que necesita la capacidad de añadir dominios hijo sin actualizar el dominio raíz.



Esta topología puede tener todavía un centro de intercambio de información central en el dominio raíz pero el segundo nivel puede actuar como centros de intercambio de información remotos para los cambios que se producen en el

dominio por debajo de ellos. El dominio raíz puede arbitrar los cambios entre los dominios sólo en el segundo nivel. También son posibles los árboles N-arios. Un árbol N-ario tiene N hijos en cada nivel. Cada dominio tiene una diseminación de N.

Consideraciones sobre arbitraje en el diseño de topología

Se podrían producir colisiones de cambio si se pueden cambiar en dos lugares a la vez los mismos registros. Configure todos los dominios para que tengan aproximadamente la misma cantidad de recursos de CPU, memoria y red. Podría observar que los dominios que realizan una gestión de colisiones de cambios (arbitraje) utilizan más recursos que otros dominios. Las colisiones se detectan automáticamente. Se resuelven utilizando uno de estos dos mecanismos:

- **Árbitro de colisión predeterminado.** El protocolo predeterminado es utilizar los cambios del dominio con un nombre de dominio menor alfabéticamente. Por ejemplo, si el dominio A y el B generan un conflicto de un registro, se pasa por alto el cambio del dominio B. El dominio A conserva su versión y el registro en el dominio B se cambia para coincidir con el registro del dominio A. Esto se aplica también para las aplicaciones en las que los usuarios o las sesiones se enlazan normalmente o tienen afinidad con una de las cuadrículas.
- **Árbitro de colisión personalizado.** Las aplicaciones pueden proporcionar un árbitro personalizado. Cuando un dominio detecta una colisión, invoca el árbitro. Para obtener información sobre cómo desarrollar un 'buen' árbitro personalizado, consulte Desarrollo de árbitros personalizados para la réplica con varios maestros.

Para las topologías en que son posibles las colisiones, considere utilizar una topología de hub y red o una topología de árbol. Estas dos topologías son propicias para impedir colisiones sin fin, que pueden suceder cuando:

1. Varios dominios sufren una colisión
2. Cada dominio resuelve la colisión de forma local, que produce revisiones
3. Las revisiones colisionan, con lo que se producen revisiones de revisiones
4. Y así sucesivamente, porque las revisiones se propagan entre los distintos dominios intentando alcanzar la sincronicidad

Para evitar las colisiones sin fin, elija un dominio específico – un *dominio de arbitraje* – como manejador de colisiones para un subconjunto de dominios. Por ejemplo, una topología de hub y radio podría utilizar el hub como manejador de colisiones. El manejador de colisiones de radio pasa por alto todas las colisiones detectadas por los dominios de radio. El dominio de hub creará revisiones, con lo que se evitan las revisiones de colisión fuera de control. El dominio asignado para manejar las colisiones debe enlazarse a todos los dominios de los que tiene la responsabilidad de resolver las colisiones. En una topología de árbol, los dominios padre internos resuelven las colisiones de sus hijos inmediatos. A diferencia con esta topología, si utiliza una topología de anillo, no puede designar un dominio en el anillo como solucionador.

En la tabla siguiente se resumen los enfoques de arbitraje que son más compatibles con distintas topologías.

Tabla 14. Enfoques de arbitraje. En esta tabla se indica si el arbitraje de la aplicación es compatible con distintas tecnologías.

Topología	¿Arbitraje de aplicación?	Notas
Una línea de dos dominios	Sí	Seleccione un dominio como árbitro.
Una línea de tres dominios	Sí	El dominio del medio debe ser el árbitro. Piense en el dominio del medio como el hub de una topología sencilla de hub y radio.
Una línea de más de tres dominios	No	No se admite el arbitraje de aplicaciones.
Un hub con N radios	Sí	El hub con enlaces a todos los radios debe ser el dominio de arbitraje.
Un anillo de N dominios	No	No se admite el arbitraje de aplicaciones.
Un árbol dirigido acíclico (árbol N-ario)	Sí	Todos los nodos raíz deben arbitrar sus descendientes directos solo.

Consideraciones sobre enlaces en el diseño de topología

De forma ideal, una topología incluye el número mínimo de enlaces cuando optimiza los compromisos entre las características de latencia de cambios, tolerancia a errores y rendimiento.

- **Latencia de cambios**

  La latencia de cambios viene determinada por el número de dominios intermedios por los que debe pasar un cambio antes de llegar a un dominio concreto.

  Una topología tiene la mejor latencia de cambios cuando elimina los dominios intermedios enlazando cada dominio a todos los demás dominios. No obstante, un dominio debe realizar un trabajo de réplica en proporción a su número de enlaces. Para las topologías de gran tamaño, el verdadero número de enlaces a definir podría representar una carga administrativa.

  La velocidad a la que se copia un cambio en otros dominios depende de factores adicionales, como:

  – CPU y ancho de banda en el dominio de origen
  – El número de dominios intermedios y enlaces entre el dominio de origen y de destino
  – Los recursos de CPU y de red disponibles para los dominios de origen, destino e intermedio

- **Tolerancia a errores**

  La tolerancia a errores viene determinada por el número de vías de acceso que existen entre dos dominios para la réplica de cambios.

  Si solo existe un único enlace entre dominios y el enlace produce un error, no se propagan los cambios. Si el enlace único de un dominio a otro pasa por dominios intermedios, no se propagan los cambios si alguno de los dominios intermedios está inactivo.

  Considere la topología de línea con tres dominios A, B y C:

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

  Si cualquiera de estas condiciones se mantiene, el Dominio C no verá los cambios de A:

– El dominio A está activo y el dominio B está inactivo
– El enlace entre A y B está inactivo
– El enlace entre B y C está inactivo

A diferencia con esta topología, la topología de anillo permite que cada dominio extraiga los cambios de cualquier dirección.

```
A <-> B <-> C <-> de nuevo a A
```

Por ejemplo, si el dominio B está inactivo, el dominio C todavía puede extraer los cambios directamente del dominio A.

Un diseño de hub y radio es propenso a que el hub se quede inactivo porque todos los cambios pasan alrededor a través del hub. No obstante, merece la pena recordar que un solo dominio sigue siendo una cuadrícula totalmente tolerante a errores que podría tener anomalías graves como problemas de WAN o del centro de datos físico.

- **Rendimiento**

  El número de enlaces definidos en un dominio influye en el rendimiento. Si hay más enlaces se utilizan más recursos y a raíz de esto podría disminuir el rendimiento. La capacidad de extraer los cambios de un dominio A a través de otros dominios descarga de forma efectiva el domino A de replicar sus transacciones por todas partes. *La carga de distribución de cambios en un dominio está limitada al número de enlaces que utiliza. No se puede hacer nada con el número de dominios de la topología.* Esta propiedad proporciona escalabilidad y permite que la carga de la distribución de cambios la compartan los dominios de la topología, en lugar de situar la carga en un solo dominio.

  Un dominio puede extraer los cambios de forma indirecta a través de otros dominios. Considere una topología de línea con cinco dominios.

  ```
  A <=> B <=> C <=> D <=> E
  ```

  – A extrae los cambios de B, C, D y E a B
  – B extrae los cambios de A y C directamente y los cambios de D y E a C
  – C realiza los cambios de B y D directamente y los cambios de A a B y de E a D
  – D extrae los cambios de C y E directamente y los cambios de A y B a C
  – E extrae los cambios de D directamente, y los cambios de A, B y C a D

  La carga de distribución en los dominios A y E es menor, porque cada uno tiene un enlace solo a un solo dominio. La carga de distribución en los dominios B, C y D tiene el doble de carga en los dominios A y E porque los dominios B, C y D tienen cada uno un enlace a dos dominios. Esta distribución de cargas permanecería constante, aun cuando la línea contuviera 1000 dominios, porque la carga depende del número de enlaces de cada dominio, no del número global de dominios de la topología.

Consideraciones sobre el rendimiento

Tenga en cuenta estas limitaciones al utilizar las topologías de réplica con varios maestros.

- **Ajuste de la distribución de cambios** (se describe anteriormente)
- **Latencia de réplica** (se describe anteriormente)
- **Rendimiento de enlace de réplica** eXtreme Scale crea un único socket TCP/IP entre cualquier par de JVM. Todo el tráfico entre esas JVM se produce en ese socket, incluida la réplica con varios maestros. Dado que los dominios se alojan en N JVM de contenedor como mínimo, si se proporcionan como mínimo N

enlaces TCP a los dominios de igual, los dominios con mayor número de contenedores tendrán niveles más altos de rendimiento de réplica. Más contenedores significa más recursos de CPU y de red.

- **Ajuste de la ventana deslizante TCP y RFC 1323** Si se habilita el soporte de RFC 1323 en los dos extremos de un enlace se permiten más datos para una transmisión de ida y vuelta, con lo que aumenta el rendimiento. La técnica amplía la capacidad de la ventana en un factor de aproximadamente 16.000.

  Recuerde que los sockets TCP utilizan un mecanismo de ventana deslizante para controlar el flujo de los datos en bloque, que suele limitar el socket a 64 KB para un intervalo de transmisiones de ida y vuelta. Si el intervalo de transmisión de ida y vuelta es de 100 mseg., el ancho de banda está limitado a 640 KB/segundo sin un ajuste adicional. El uso de todo el ancho de banda disponible en un enlace podría requerir un ajuste específico de un sistema operativo. La mayoría de sistemas operativos incluyen parámetros de ajuste, incluidas las opciones de RFC 1323, para mejorar el rendimiento en enlaces de alta latencia.

  Varios factores pueden afectar al rendimiento de la réplica:

  – La velocidad a la que eXtreme Scale puede hacer los cambios.
  – La velocidad a la que eXtreme Scale puede atender la solicitudes de extracción de réplica.
  – La capacidad de la ventana deslizante.
  – El ajuste del almacenamiento intermedio de red en los dos extremos del enlace para permitir que eXtreme Scale extraiga los cambios en el socket lo más ágil posible.
- **Serialización de objetos** Todos los datos deben ser serializables. Si un dominio no utiliza COPY_TO_BYTES, el dominio debe utilizar la serialización de Java u ObjectTransformers para optimizar el rendimiento de la serialización.
- **Compresión** eXtreme Scale comprime todos los datos enviados entre dominios de forma predeterminada. No hay opción para inhabilitar la compresión en el release actual.
- **Ajuste de memoria** *El uso de memoria para una topología de réplica con varios maestros es muy independiente del número de dominios de la topología.*

  La habilitación de la réplica con varios maestros añade una sobrecarga fija por entrada Map para gestionar el mantenimiento de versiones. Cada contenedor realiza un seguimiento también de una cantidad fija de datos de cada dominio de la topología. Una topología con dos dominios utiliza aproximadamente la misma memoria que una topología con 50 dominios. eXtreme Scale no utiliza registros de reproducción o colas similares en su implementación, lo que significa que si un enlace de réplica no está disponible durante un periodo de tiempo sustancial, no se produce un aumento en tamaño de la estructura de datos, a la espera de reanudar la réplica cuando se inicia el enlace.

Varios centros de datos con FIXED_PARTITION

Ahora puede utilizar una cuadrícula FIXED_PARTITION entre dos o más centros de datos. Cada centro de datos necesita su propio dominio, en lo que se refiere a réplica con varios maestros. Cada centro de datos lee y graba los datos en el dominio local. Estos cambios se propagarán a los otros centros de datos con los enlaces que ha definido.

Clientes totalmente replicados

En esta variación de topología interviene un par de servidores eXtreme Scale que se ejecutan como un hub. Todos los clientes crean una cuadrícula de contenedor

única autocontenida con un catálogo en la JVM cliente. El cliente utiliza su cuadrícula para conectarse al catálogo de hub, que hace que el cliente se sincronice con el hub en cuanto el cliente obtiene una conexión al hub.

Los cambios realizados por el cliente son locales al cliente y se hace una réplica de ellos asíncrona en el hub. El hub actúa como un dominio de arbitraje, que distribuye los cambios a todos los clientes conectados. La topología de clientes totalmente replicados proporciona una buena memoria caché L2 para un correlacionador relacional de objetos, como OpenJPA. Los cambios se distribuirán rápidamente entre las JVM cliente por el hub. Siempre que el tamaño de la memoria caché se pueda incluir en el espacio de almacenamiento dinámico disponible de los clientes, esta topología es una buena arquitectura para este estilo de L2.

Utilice varias particiones para escalar el dominio del hub en varias JVM, si es necesario. Dado que todos los datos todavía deben caber en una sola JVM cliente, el uso de varias particiones aumenta la capacidad del hub para distribuir y arbitrar los cambios, pero no cambia la capacidad de un dominio único.

Limitaciones

Tenga en cuenta estas limitaciones al determinar si va a utilizar y cómo utilizar las topologías de réplica con varios maestros.

- **Tenga cuidado al configurar los cargadores de clases con varios dominios**

  Los dominios deben tener acceso a todas las clases que se utilizan como claves y valores. Todas las dependencias se deben reflejar en todas las vías de acceso de clases de las JVM de contenedor de cuadrícula para todos los dominios. Si un plug-in CollisionArbiter recupera el valor para una entrada de memoria caché, las clases de los valores deben estar presentes para el dominio que invoca el árbitro.

- **No se recomienda utilizar cargadores**

  Los cargadores se pueden utilizar para hacer de interfaz de los cambios entre una cuadrícula y una base de datos. Es improbable que todas las cuadrículas (dominios) de una topología se den en combinación geográfica con la misma base de datos. La latencia de WAN y otros factores podrían representar este caso de uso no deseable.

  La carga previa de cuadrícula es otro problema que requiere un diseño cuidadoso. Normalmente, cuando se reinicia una cuadrícula, se vuelve a cargar previamente de nuevo. No es necesaria la precarga o incluso no es deseable al utilizar la réplica con varios maestros. En cuanto un dominio está en línea, se vuelve a cargar automáticamente a sí mismo con el contenido de los dominios a los que está enlazado. Como consecuencia, no es necesario iniciar una precarga manual para una cuadrícula que es un dominio de una topología de réplica con varios maestros.

  Los cargadores suelen seguir las reglas de inserción y actualización. Con la réplica con varios maestros, las inserciones se deben tratar como fusiones. Cuando se extraen los datos de forma remota después de que se inicia un dominio, los datos existentes se 'insertarán' en el dominio local. Dado que estos datos podrían estar ya en la base de datos local, una inserción típica producirá un error con una excepción de clave duplicada en la base de datos. En su lugar, se debe utilizar la semántica de fusión.

  eXtreme Scale se puede configurar para hacer una precarga basada en fragmentos, con los métodos de precarga en los plug-in Loader. No utilice esta técnica en una topología de réplica con varios maestros. En su lugar, utilice una

precarga basada en cliente cuando se inicia la topología (inicialmente). Permita que la topología con varios maestros renueve los dominios reiniciados con una copia actual que está almacenada en otros dominios de la topología. Después de que se han iniciado los dominios, la topología con varios maestros se encarga de conservarlos en sincronismo.

- **No se admite EntityManager**

  Un conjunto de correlaciones que contiene una correlación de entidad no se replica entre dominios.
- **No se admiten las correlaciones de matriz de bytes**

  Un conjunto de correlaciones que contiene una correlación que está configurada con COPY_TO_BYTES no se replica entre dominios.
- **No se admite la grabación diferida**

  Un conjunto de correlaciones que contiene una correlación que está configurada con soporte de grabación diferida no se replica entre dominios.

JMS para distribuir los cambios de transacciones

Utilice JMS (Java Message Service) para los cambios de transacciones distribuidas entre las distintas capas o en entornos en plataformas combinadas.

JMS es un protocolo ideal para distribuir cambios entre distintos niveles o en entornos con diferentes plataformas. Por ejemplo, algunas aplicaciones que utilizan eXtreme Scale se podrían desplegar en IBM WebSphere Application Server Community Edition, Apache Geronimo o Apache Tomcat, mientras que otras aplicaciones se podrían ejecutar en WebSphere Application Server versión 6.x. JMS es ideal para los cambios distribuidos entre los iguales de eXtreme Scale en estos distintos entornos. El transporte de mensajes de High Availability Manager es muy rápido, pero sólo puede distribuir cambios a Máquinas virtuales Java que estén en un único grupo principal. JMS es más lento, pero permite que conjuntos más grandes y más diversos de clientes de aplicación puedan compartir un ObjectGrid. JMS es ideal si se comparten datos en un ObjectGrid entre un cliente grueso de Swing y una aplicación desplegada en WebSphere Extended Deployment.

El mecanismo de invalidación de clientes incorporado y la réplica de igual a igual son ejemplos de distribución de cambios transaccionales basados en JMS. Consulte la información sobre cómo configurar la réplica de igual a igual con JMS en *Guía de administración* si desea más información.

Implementación de JMS

JMS se implementa para distribuir los cambios de transacciones utilizando un objeto Java que se comporta como un ObjectGridEventListener. Este objeto puede propagar el estado de las cuatro formas siguientes:

1. Invalidación: las entradas desalojadas, actualizadas o suprimidas se eliminan de todas las Mäquinas virtuales Java de iguales al recibir el mensaje.
2. Invalidación condicional: la entrada sólo se desaloja si la versión local es la misma o más antigua que la versión del editor.
3. Envío: las entradas desalojadas, actualizadas, suprimidas o insertadas se añaden o se sobrescriben en todas las Máquinas virtuales Java de iguales al recibir el mensaje JMS.
4. Envío condicional: la entrada sólo se actualiza o se añade en el lado del receptor si la entrada local es menos reciente que la versión que se va a publicar.

Escuchar cambios de publicación

El plug-in implementa la interfaz ObjectGridEventListener para interceptar el suceso transactionEnd. Cuando eXtreme Scale invoca este método, el plug-in intenta convertir la lista LogSequence de cada correlación manipulada por la transacción a un mensaje JMS, que intentará publicar. El plug-in puede haberse configurado para publicar cambios de todas las correlaciones o de un subconjunto. Los objetos LogSequence se procesan para las correlaciones que tienen habilitada la publicación. La clase LogSequenceTransformer ObjectGrid serializa un objeto filtrado LogSequence de cada correlación en una corriente. Después de que todos los objetos LogSequences se serialicen en una corriente, se crea un objeto JMS ObjectMessage y se publica para un tema conocido.

Escuchar mensajes JMS y aplicarlos al objeto ObjectGrid local

El mismo plug-in también inicia una hebra que forma un bucle y recibe todos los mensajes publicados para un tema conocido. Cuando llega un mensaje, se pasa el contenido del mensaje a la clase LogSequenceTransformer, donde se convierte a un conjunto de objetos LogSequence. A continuación, se inicia una transacción de no escritura a través. Cada objeto LogSequence se proporciona al método Session.processLogSequence, que actualiza las correlaciones locales con los cambios. El método processLogSequence entiende la modalidad de distribución. La transacción se confirma y la memoria caché local refleja los cambios. Si desea más información sobre cómo utilizar JMS para distribuir cambios de transacción, consulte la información sobre cómo distribuir los cambios entre máquinas virtuales Java iguales en *Guía de administración*.

Conjuntos de correlaciones para réplica

La réplica se habilita asociando BackingMaps a un MapSet.

Un MapSet es una colección de correlaciones que se categorizan por clave de partición. Esta clave de partición se obtiene de la clave del correlación individual efectuando una operación módulo entre hash y el número de particiones. Por lo tanto, si un grupo de correlaciones dentro de MapSet tiene la clave de partición X, estas correlaciones se almacenarán en la correspondiente partición X en la cuadrícula; si otro grupo tiene la clave de partición Y, todas las correlaciones se almacenarán en la partición Y, etc. Además, los datos de las correlaciones se replican en función de la política definida en MapSet, que sólo se utiliza para topologías de eXtreme Scale distribuidas (no es necesario para instancias locales).

Si desea más información, consulte “Particionamiento” en la página 93.

A los MapSets se les asigna el número de particiones que tendrán y una política de réplica. La configuración de réplica de MapSet simplemente identifica el número de fragmentos de réplicas síncronas y asíncronas que un MapSet debe tener además del fragmento primario. Por ejemplo, si debe haber una réplica síncrona y una asíncrona, en cada una de las BackingMaps asignadas al MapSet se distribuirá automáticamente un fragmento de réplica dentro del conjunto de contenedores disponibles para eXtreme Scale. La configuración de réplica también puede permitir que los clientes lean datos de servidores duplicados de forma síncrona. Esto puede esparcir la carga de las solicitudes de lectura entre servidores adicionales en eXtreme Scale. La réplica sólo tiene un impacto de modelo de programación cuando se realiza la precarga de BackingMaps.

Para obtener detalles sobre las distintas opciones de configuración, consulte más abajo:

Capítulo 6. Visión general del proceso de transacciones

Sesiones y proceso de transacciones

WebSphere eXtreme Scale utiliza las transacciones como su mecanismo para la interacción con datos.

Para interactuar con los datos, la hebra de la aplicación necesita su propio objeto Session. Si la aplicación desea utilizar el ObjectGrid en una hebra, llame a uno de los métodos ObjectGrid.getSession para obtener una hebra. Con la sesión, la aplicación puede trabajar con los datos almacenados en las correlaciones de ObjectGrid.

Cuando una aplicación utiliza utiliza un objeto Session, la sesión debe estar en el contexto de una transacción. Una transacción empieza o se confirma y retrotrae mediante los métodos begin, commit y rollback en el objeto Session. Las aplicaciones también pueden funcionar en la modalidad de confirmación automática, en la que Session empieza automáticamente y confirma una transacción, siempre que se realiza una operación en la correlación. Una modalidad de confirmación automática no puede agrupar varias operaciones en una única transacción, de forma que es la opción más lenta si crea un proceso por lotes de varias operaciones en una única transacción. Sin embargo, para las transacciones que sólo contienen una operación, la confirmación automática es la opción más rápida.

Transacciones

Las transacciones tienen muchas ventajas para el almacenamiento de datos y la manipulación. Puede utilizar las transacciones para proteger la cuadrícula de los cambios simultáneos, para aplicar varios cambios como una unidad simultánea, para replicar datos y para implementar un ciclo de vida para los bloqueos en los cambios.

Cuando se inicia una transacción, WebSphere eXtreme Scale asigna una correlación de diferencias especial para mantener los cambios o copias actuales de pares de clave y valor que la transacción utiliza. Normalmente, cuando se accede a un par de clave y valor, el valor se copia antes de que la aplicación reciba el valor. La correlación de diferencias rastrea todos los cambios para las operaciones como, por ejemplo, insert, update, get, remove, etc. Las claves no se copian porque se da por supuesto que son inmutables. Si se especifica un objeto ObjectTransformer, este objeto se utiliza para copiar el valor. Si la transacción utiliza el bloqueo optimista, también se realiza un seguimiento de las imágenes anteriores de los valores para su comparación cuando se confirma la transacción.

Si se retrotrae una transacción, se descarta la información de correlación de diferencias y se liberan los bloqueos de las entradas. Cuando se confirma una transacción, los cambios se aplican a las correlaciones y se liberan los bloqueos. Si se utiliza el bloqueo optimista, eXtreme Scale compara las versiones de imágenes anteriores de los valores con los valores incluidos en la correlación. Estos valores deben coincidir para que la transacción se confirme. Esta comparación permite un esquema de bloqueo de varias versiones, pero a costa de que se realicen dos copias cuando la transacción accede a la entrada. Se vuelven a copiar todos los valores y

se almacena la nueva copia en la correlación. WebSphere eXtreme Scale realiza esta copia para evitar que la aplicación cambie la referencia de la aplicación por el valor después de una confirmación.

Puede evitar utilizar varias copias de la información. La aplicación puede guardar una copia utilizando el bloqueo pesimista en lugar del bloqueo optimista como coste de limitar la concurrencia. También se puede evitar la copia del valor durante la confirmación si la aplicación acepta no cambiar un valor después de la confirmación.

Ventajas de las transacciones

Utilice transacciones por las siguientes razones:

Mediante el uso de transacciones, puede:

- Retrotraer cambios si se produce una excepción o si la lógica empresarial necesita deshacer los cambios de estado.
- Para aplicar varios cambios como una unidad atómica durante la confirmación.
- Mantener y liberar bloqueos en los datos para aplicar varios cambios como una unidad atómica durante la confirmación.
- Proteger una hebra de los cambios simultáneos.
- Implementar un ciclo de vida para los bloqueos en cambios.
- Producir una unidad atómica de duplicación.

Tamaño de transacción

Las transacciones de mayor tamaño son más eficaces, especialmente para la réplica. Sin embargo, las transacciones de mayor tamaño pueden afectar de forma adversa a la concurrencia porque se mantienen durante más tiempo los bloqueos sobre entradas. Si utiliza transacciones de mayor tamaño, puede aumentar el rendimiento de la réplica. El aumento de este rendimiento es importante cuando se precarga una correlación. Pruebe con distintos tamaños de lotes para determinar lo que funciona mejor en cada caso.

Las transacciones de mayor tamaño también son útiles con los cargadores. Si se está utilizando un cargador que puede realizar el proceso por lotes de SQL, son posibles aumentos significativos de rendimiento en función de la transacción y las reducciones significativas de la carga en el lado de la base de datos. Esta ganancia en el rendimiento dependerá de la implementación del cargador.

Modalidad de confirmación automática

Si no se ha iniciado de forma activa ninguna transacción, cuando una aplicación interactúa con un objeto ObjectMap, empieza una operación automática de inicio y confirmación en nombre de la aplicación. Esta operación automática de inicio y confirmación funciona, pero impide que la retrotracción y el bloqueo funcionen de forma eficaz. La velocidad de réplica síncrona se ve afectado debido al tamaño de transacción muy pequeño. Si utiliza una aplicación de gestor de entidades, no utilice la modalidad de confirmación automática porque los objetos que busca el método EntityManager.find se convierten inmediatamente en no gestionados en la devolución del método y dejan de poderse utilizar.

Coordinadores de transacciones externos

Normalmente, las transacciones se inician con el método session.begin y finalizan con el método session.commit. Sin embargo, cuando se incorpora eXtreme Scale, las transacciones podrían iniciarse y terminarse a través de un coordinador de transacciones externo. Si utiliza un coordinador de transacciones externas, no tendrá que llamar al método session.begin y finalizar el método session.commit. Consulte *Guía de programación* si desea más información sobre eXtreme Scale y la interacción de transacciones externas. Si utiliza WebSphere Application Server, puede utilizar el plug-in WebSphereTranscationCallback. Consulte *Guía de programación* si desea más información sobre los plug-ins que están disponibles con WebSphere eXtreme Scale.

Atributo CopyMode

Puede ajustar el número de copias definiendo el atributo CopyMode de los objetos BackingMap u ObjectMap.

Puede ajustar el número de copias definiendo el atributo CopyMode de los objetos BackingMap u ObjectMap. La modalidad de copia tiene los siguientes valores:

- COPY_ON_READ_AND_COMMIT
- COPY_ON_READ
- NO_COPY
- COPY_ON_WRITE
- COPY_TO_BYTES

El valor COPY_ON_READ_AND_COMMIT es el valor predeterminado. El valor COPY_ON_READ copia los datos iniciales recuperados, pero no copia durante la confirmación. Esta modalidad es segura si la aplicación no modifica un valor después de confirmar una transacción. El valor NO_COPY no copia datos, que sólo es seguro para los datos de sólo lectura. Si los datos nunca cambian, no tendrá que copiarlos por razones de aislamiento.

Tenga cuidado cuando utilice el valor del atributo NO_COPY con las correlaciones que se pueden actualizar. WebSphere eXtreme Scale utiliza la copia en el primer toque para permitir la retrotracción de la transacción. La aplicación sólo ha cambiado la copia y, como resultado, eXtreme Scale descarta la copia. Si se utiliza el valor de atributo NO_COPY, y la aplicación modifica el valor confirmado, no es posible completar una retrotracción. Si se modifica el valor confirmado comportará problemas con índices, réplica, etc, porque los índices y las réplicas se actualizan cuando se confirma la transacción. Si modifica los datos confirmados y, a continuación, retrotrae la transacción, que en realidad no se retrotrae, los índices no se actualizan y la réplica no tiene lugar. Otras hebras pueden ver los cambios no confirmados inmediatamente, incluso si tienen bloqueos. Utilice el valor de atributo NO_COPY para las correlaciones de sólo lectura o para aplicaciones que completan la copia apropiada antes de modificar el valor. Si utiliza el valor de atributo NO_COPY y llama al soporte de IBM con un problema de integridad de datos, se le solicitará que reproduzca el problema con la modalidad de copia establecida en COPY_ON_READ_AND_COMMIT.

El valor COPY_TO_BYTES almacena valores en la correlación de un formato serializado. En el momento de lectura, eXtreme Scale infla el valor a partir de un formato serializado y en el momento de confirmación almacena el valor en un formato serializado. Con este método, se produce una copia durante la lectura y la confirmación.

La modalidad de copia predeterminada para una correlación se puede configurar en el objeto BackingMap. También puede cambiar la modalidad de copia en las correlaciones antes de iniciar una transacción mediante el uso del método ObjectMap.setCopyMode.

A continuación, aparece un ejemplo de un fragmento de código de la correlación de respaldo de un archivo `objectgrid.xml` que muestra cómo establecer la modalidad de copia para una correlación de respaldo dada. Este ejemplo da por supuesto que utiliza `cc` como espacio de nombres de `objectgrid/config`.

```
<cc:backingMap name="RuntimeLifespan" copyMode="NO_COPY"/>
```

Consulte la información sobre los procedimientos recomendados de copyMode en *Guía de programación* si desea más información.

Bloqueo de entrada de correlación

Una BackingMap de ObjectGrid admite diversas estrategias de bloqueo para que las correlaciones mantengan la coherencia de las entradas en memoria caché.

Cada BackingMap puede configurarse de modo que utilice una de las estrategias de bloqueo siguientes:

1. Modalidad de bloqueo optimista
2. Modalidad de bloqueo pesimista
3. Ninguno

La estrategia de bloqueo predeterminada es OPTIMISTIC. Utilice el bloqueo optimista cuando los datos no se modifican frecuentemente. Los bloqueos sólo se mantienen durante un tiempo breve mientras los datos se leen de la memoria caché y se copian en la transacción. Cuando la memoria caché de la transacción se sincroniza con la memoria caché principal, los objetos de la memoria caché actualizados se comprueban contra la versión original. Si la comprobación falla, la transacción se retrotrae y se produce la excepción OptimisticCollisionException.

La estrategia de bloqueo PESSIMISTIC adquiere bloqueos para las entradas de memoria caché y debe utilizarse cuando los datos se cambian con frecuencia. Cada vez que se lee una entrada de la memoria caché, se adquiere un bloqueo, que puede mantenerse condicionalmente hasta que se complete la transacción. La duración de algunos de los bloqueos pueden ajustarse mediante el uso de niveles de aislamiento para la sesión.

Si el bloqueo no es necesario porque los datos nunca se actualizan o sólo se actualizan durante períodos tranquilos, puede inhabilitar el bloqueo mediante el uso de la estrategia de bloqueo NONE. Esta estrategia es muy rápida porque no se necesita ningún gestor de bloqueos. La estrategia de bloqueo NONE es ideal en tablas de búsqueda o en correlaciones de sólo lectura.

Si desea más información sobre las estrategias de bloqueo, consulte la información sobre las estrategias de bloqueo en *Visión general del producto*.

Especificación de una estrategia de bloqueo

El siguiente ejemplo demuestra cómo se puede establecer la estrategia de bloqueo en las correlaciones BackingMaps map1, map2 y map3, donde cada correlación utiliza una estrategia de bloqueo diferente. El primer fragmento de código muestra

cómo utilizar el XML para la configuración de la estrategia de bloqueo y el segundo fragmento de código muestra un enfoque mediante programa.

Enfoque de XML

Configuración de BackingMap - XML de ejemplo

```
<?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>
```

Enfoque mediante programa

Configuración de BackingMap - ejemplo mediante programa

```
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 );
```

Para evitar una excepción de java.lang.IllegalStateException, se debe llamar al método setLockStrategy antes de utilizar los métodos initialize o getSession en una instancia local de ObjectGrid.

Para obtener más información, consulte el tema sobre las estrategias de bloqueo en la *Visión general del producto*.

Configuración del gestor de bloqueos

Cuando se utiliza una estrategia de bloqueo PESSIMISTIC u OPTIMISTIC, se crea un gestor de bloqueos para BackingMap. El gestor de bloqueos utiliza una correlación hash para realizar un seguimiento de las entradas bloqueadas por una o más transacciones. Cuantas más entradas de correlación existan en la correlación hash, mayor será el grupo de bloqueos con un buen rendimiento. El riesgo de las colisiones de sincronización de Java es menor a medida que crece el número de grupos. Un número mayor de grupos también implica mayor simultaneidad. Los ejemplos anteriores muestran cómo una aplicación puede establecer el número de grupos de bloqueos que se deben utilizar en una instancia determinada de BackingMap.

Para evitar una excepción java.lang.IllegalStateException, debe llamarse al método setNumberOfLockBuckets antes que a los métodos initialize o getSession en la

instancia ObjectGrid. El parámetro del método setNumberOfLockBuckets es un entero primitivo de Java que especifica el número de grupos de bloqueo para utilizar. El uso de un número primo puede permitir una distribución uniforme de entradas de correlación en los grupos de bloqueos. Un buen punto de partida para obtener un mejor rendimiento es establecer el número de grupos de bloqueos en un 10 por ciento del número esperado de entradas de BackingMap.

LockDeadlockException

A continuación se ofrece un ejemplo de código que muestra la obtención de la excepción y luego se muestra el mensaje resultante.

```
try {
...
} catch (ObjectGridException oe) {
System.out.println(oe);
}
```

El resultado es el siguiente:

```
com.ibm.websphere.objectgrid.plugins.LockDeadlockException: _Message
```

Este mensaje representa la serie que se pasa como parámetro cuando se crea y se emite la excepción.

Causa de la excepción

El tipo más común de excepción de punto muerto se produce cuando se utiliza la estrategia de bloqueo pesimista y dos clientes distintos tienen un bloqueo compartido sobre un determinado objeto. Entonces, ambos clientes intentan ascender a un bloqueo exclusivo sobre dicho objeto. El diagrama siguiente ilustra esta situación, incluidos los bloques de transacción que causan la emisión de la excepción.

El siguiente fragmento de código Java demuestra cómo pasar un archivo de configuración de XML para crear un ObjectGrid.

Se trata de una visión abstracta de lo que sucede en su programa cuando se produce la excepción. En una aplicación con muchas hebras que actualizan la misma ObjectMap, es posible que se produzca esta situación. A continuación se ofrece un ejemplo de dos clientes que ejecutan los bloques de código de transacción, tal como se ilustra en la figura anterior.

Soluciones posibles

Podría encontrarse con la situación ilustrada en la Figura 1 cuando haya numerosas hebras que inicien transacciones en una determinada correlación. En este caso, la excepción se emite para impedir que su programa se cuelgue. Puede enviarse una notificación y añadir código al bloque catch para obtener más detalles sobre la causa. Puesto que sólo verá esta excepción en una estrategia de bloqueo pesimista, una solución sencilla consiste simplemente en utilizar una estrategia de bloqueo optimista. Si, no obstante, requiere una estrategia de bloqueo pesimista, puede utilizar el método getForUpdate en lugar del método get. De este modo se elimina la recepción de las excepciones para la situación descrita anteriormente.

Estrategias de bloqueo

Las estrategias de bloqueo pueden ser de tipo pesimista, optimista o ninguno. Para elegir la estrategia de bloqueo, debe tener en cuenta cuestiones como el porcentaje de cada tipo de operaciones que realizará, si utilizará un cargador o no, etc.

Los bloqueos son enlazados por transacciones. Puede especificar los siguientes valores de bloqueo:

- **Sin bloqueo**: la ejecución sin el valor de bloqueo es la más rápida. Si utiliza datos de sólo lectura, es posible que no necesite el bloqueo.
- **Bloqueo pesimista**: adquiere bloqueos sobre entradas y luego mantiene los bloqueos hasta que se realiza la confirmación. Esta estrategia de bloqueo proporciona una mayor coherencia a costa del rendimiento.
- **Bloqueo optimista**: toma una imagen anterior de cada registro que toca la transacción y compara la imagen con los valores de entrada actuales cuando se confirma la transacción. Si los valores de entrada cambian, la transacción se retrotrae. No se mantiene ningún bloqueo hasta el momento de la confirmación. Esta estrategia de bloqueo proporciona una mejor concurrencia que la estrategia pesimista, con el riesgo de que la transacción se retrotraiga y el coste de memoria de realizar una copia adicional de la entrada.

Establezca la estrategia de bloqueo en la BackingMap. No puede cambiar la estrategia de bloqueo para cada transacción. A continuación, aparece un fragmento de código XML de ejemplo que muestra cómo establecer la modalidad de bloqueo en una correlación utilizando el archivo XML, que da por supuesto que `cc` es el espacio de nombres para el espacio de nombres de `objectgrid/config`:

```
<cc:backingMap name="RuntimeLifespan" lockStrategy="PESSIMISTIC" />
```

Bloqueo pesimista

Utilice la estrategia de bloqueo pesimista en operaciones de correlación de lectura y grabación cuando no es posible utilizar otra estrategia de bloqueo. Cuando se configura una correlación ObjectGrid para utilizar la estrategia de bloqueo pesimista, se obtiene un bloqueo de transacción pesimista para una entrada de correlación cuando una transacción obtiene por primera vez la entrada de BackingMap. El bloqueo pesimista se mantiene hasta que la aplicación completa la transacción. Por lo general, la estrategia de bloqueo pesimista se utiliza en las situaciones siguientes:

- Cuando BackingMap se configura con o sin un cargador y la información de creación de versiones no está disponible.
- Cuando BackingMap se utiliza directamente en una aplicación que necesita ayuda de eXtreme Scale para el control de simultaneidad.
- Cuando la información de creación de versiones está disponible, pero las transacciones de actualización colisionan con frecuencia en las entradas de respaldo, lo cual produce anomalías optimistas de actualización.

Como la estrategia de bloqueo pesimista tiene el mayor impacto sobre el rendimiento y la escalabilidad, esta estrategia sólo debe utilizarse para correlaciones de lectura y grabación, cuando no es viable ninguna otra estrategia de bloqueo. Por ejemplo, estas situaciones podrían incluir cuando se producen con frecuencia anomalías optimistas de actualización, o cuando es difícil para una aplicación gestionar la recuperación de una anomalía optimista.

Bloqueo optimista

En la estrategia de bloqueo optimista, se presupone que dos transacciones no pueden intentar actualizar la misma entrada de correlación mientras se ejecutan simultáneamente. Por este motivo, la modalidad de bloqueo no necesita mantenerse para el ciclo de vida de la transacción, porque ya que es improbable que más de una transacción actualice la entrada de correlación simultáneamente. Por lo general, la estrategia de bloqueo optimista se utiliza en las situaciones siguientes:

- Cuando BackingMap se configura con o sin un cargador y la información de creación de versiones está disponible.
- Cuando BackingMap tiene mayoritariamente transacciones que realizan operaciones de lectura. En BackingMap, las operaciones insert, update o remove no se producen con frecuencia en las entradas de correlaciones.
- Cuando una correlación BackingMap se inserta, actualiza o elimina con más frecuencia de lo que se lee, pero las transacciones rara vez colisionan en la misma entrada de correlación.

Al igual que en la estrategia de bloqueo pesimista, los métodos de la interfaz ObjectMap determinan cómo eXtreme Scale intenta automáticamente adquirir una modalidad de bloqueo para una entrada de correlación a la que se accede. No obstante, las diferencias entre las estrategias optimistas y pesimistas son:

- Al igual que la estrategia de bloqueo pesimista, los métodos get y getAll adquieren una modalidad de bloqueo S cuando se invoca el método. Sin embargo, con el bloqueo optimista, la modalidad de bloqueo S no se mantiene hasta que finaliza la transacción, sino que se libera antes de que el método vuelva a la aplicación. El propósito de adquirir la modalidad de bloqueo es que eXtreme Scale pueda garantizar que sólo los datos confirmados de otras transacciones sean visibles a la transacción actual. Después de que eXtreme Scale haya comprobado que los datos se han confirmado, la modalidad de bloqueo S se libera. Durante el ciclo de confirmación, se realiza una comprobación de la creación de versiones optimista para garantizar que ninguna otra transacción haya modificado la entrada de correlación después de que la transacción actual haya liberado su modalidad de bloqueo S. Si no se capta una entrada de la correlación antes de que se actualice, invalide o suprima, el tiempo de ejecución de eXtreme Scale capta de forma implícita la entrada de la correlación. Esta operación get implícita se realiza para obtener el valor actual en el momento en que la entrada se solicitó para modificarse.
- A diferencia de la estrategia de bloqueo pesimista, los métodos getForUpdate y getAllForUpdate se manejan exactamente igual que los métodos get y getAll cuando se utiliza la estrategia de bloqueo optimista. Es decir, se adquiere una modalidad de bloqueo S al inicio del método y se libera la modalidad de bloqueo S antes de que se devuelva a la aplicación.

Todos los otros métodos ObjectMap se manejan exactamente igual que si se manejaran para la estrategia de bloqueo pesimista. Es decir, cuando se invoca el método commit, se obtiene una modalidad de bloqueo X para cualquier entrada de correlación que se inserte, actualice, elimine, manipule o invalide, y la modalidad de bloqueo X se mantiene hasta que la transacción complete el proceso de confirmación.

En la estrategia de bloqueo optimista, se presupone que ninguna transacción que se ejecute simultáneamente con otra intentará actualizar la misma entrada de correlación. Por este motivo, la modalidad de bloqueo no necesita mantenerse durante toda la vida de la transacción ya que es improbable que más de una

transacción actualice la entrada de correlación simultáneamente. Sin embargo, como no se ha mantenido la modalidad de bloqueo, otra transacción simultánea podría actualizar de forma potencial la entrada de la correlación, después de que la transacción actual haya liberado su modalidad de bloqueo S.

Para manejar esta posibilidad, eXtreme Scale obtiene un bloqueo X durante el ciclo de confirmación y realiza una comprobación de la creación de versiones optimista para verificar que ninguna otra transacción haya modificado la entrada de correlación después de que la transacción actual haya leído la entrada de correlación en BackingMap. Si otra transacción ha modificado la entrada de correlación, se produce una anomalía en la comprobación de versiones y se muestra una excepción OptimisticCollisionException. Esta excepción obliga a la transacción actual retrotraerse y la aplicación debe volver a intentar toda la transacción. La estrategia de bloqueo optimista es muy útil cuando se producen sobre todo lecturas de una correlación y rara vez se producen actualizaciones de la misma entrada de correlación.

Sin bloqueo

Cuando un objeto BackingMap se configura para que no use ninguna estrategia de bloqueo, no se obtiene ningún bloqueo de transacción para una entrada de correlación.

No usar ninguna estrategia de bloqueo es útil cuando una aplicación es un gestor de persistencia como, por ejemplo, un contenedor EJB (Enterprise JavaBeans™) o cuando una aplicación utiliza Hibernate para obtener los datos persistentes. En este escenario, BackingMap se configura sin cargador y el gestor de persistencia utiliza BackingMap como memoria caché de datos. En este escenario, el gestor de persistencia proporciona control de simultaneidad entre las transacciones que están accediendo a las mismas entradas de correlación.

WebSphere eXtreme Scale no necesita obtener ningún bloqueo de transacción para el control de simultaneidad. Esta situación presupone que el gestor de persistencia no libera sus bloqueos de transacción antes de actualizar la correlación de ObjectGrid con los cambios confirmados. Si el gestor de persistencia libera sus bloqueos, debe utilizarse una estrategia de bloqueo optimista o pesimista. Por ejemplo, suponga que el gestor de persistencia de un contenedor EJB está actualizando una correlación de ObjectGrid con los datos que se confirmaron en la transacción gestionada por el contenedor EJB. Si la actualización de la correlación de ObjectGrid se produce antes de que se liberen los bloqueos de transacción del gestor de persistencia, podrá utilizar la estrategia sin bloqueos. Si la actualización de la correlación de ObjectGrid se produce después de que se liberen los bloqueos de transacción del gestor de persistencia, debe utilizar la estrategia de bloqueo optimista o pesimista.

Otro escenario en el que se puede utilizar una estrategia sin bloqueos es cuando la aplicación utiliza BackingMap directamente y se ha configurado un cargador para la correlación. En este escenario, el cargador utiliza el soporte de control de simultaneidad proporcionado por un sistema de gestión de bases de datos relacionales (RDBMS) mediante el uso de Java Database Connectivity (JDBC) o Hibernate para acceder a los datos de la base de datos relacional. La implementación de cargador puede utilizar un acercamiento optimista o pesimista. Un cargador que utiliza un bloqueo optimista o un procedimiento de creación de versiones favorece un alto nivel de simultaneidad y rendimiento. Si desea más información sobre cómo implementar un enfoque de bloqueo optimista, consulte la sección OptimisticCallback en la información sobre las consideraciones de cargador

en *Guía de administración*. Si utiliza un cargador que utiliza el soporte de bloqueo pesimista de una programa de fondo subyacente, es posible que desee utilizar el parámetro forUpdate que se pasa en el método get de la interfaz Loader. Establezca este parámetro en true si el método getForUpdate de la interfaz ObjectMap ha sido utilizado por la aplicación para obtener los datos. El cargador puede utilizar este parámetro para determinar si debe solicitar un bloqueo actualizable en la fila que se está leyendo. Por ejemplo, DB2 obtiene un bloqueo que se puede actualizar si una sentencia SQL select contiene una cláusula `FOR UPDATE`. Este acercamiento ofrece la misma prevención de situaciones de punto muerto que la descrita en el apartado "Bloqueo pesimista" en la página 165.

Para obtener más información, consulte el tema sobre el manejo de bloqueos en la *Guía de programación* o el tema sobre el bloqueo de entrada de correlación en la *Guía de administración*.

JMS para distribuir los cambios de transacciones

Utilice JMS (Java Message Service) para los cambios de transacciones distribuidas entre las distintas capas o en entornos en plataformas combinadas.

JMS es un protocolo ideal para distribuir cambios entre distintos niveles o en entornos con diferentes plataformas. Por ejemplo, algunas aplicaciones que utilizan eXtreme Scale se podrían desplegar en IBM WebSphere Application Server Community Edition, Apache Geronimo o Apache Tomcat, mientras que otras aplicaciones se podrían ejecutar en WebSphere Application Server versión 6.x. JMS es ideal para los cambios distribuidos entre los iguales de eXtreme Scale en estos distintos entornos. El transporte de mensajes de High Availability Manager es muy rápido, pero sólo puede distribuir cambios a Máquinas virtuales Java que estén en un único grupo principal. JMS es más lento, pero permite que conjuntos más grandes y más diversos de clientes de aplicación puedan compartir un ObjectGrid. JMS es ideal si se comparten datos en un ObjectGrid entre un cliente grueso de Swing y una aplicación desplegada en WebSphere Extended Deployment.

El mecanismo de invalidación de clientes incorporado y la réplica de igual a igual son ejemplos de distribución de cambios transaccionales basados en JMS. Consulte la información sobre cómo configurar la réplica de igual a igual con JMS en *Guía de administración* si desea más información.

Implementación de JMS

JMS se implementa para distribuir los cambios de transacciones utilizando un objeto Java que se comporta como un ObjectGridEventListener. Este objeto puede propagar el estado de las cuatro formas siguientes:

1. Invalidación: las entradas desalojadas, actualizadas o suprimidas se eliminan de todas las Mäquinas virtuales Java de iguales al recibir el mensaje.
2. Invalidación condicional: la entrada sólo se desaloja si la versión local es la misma o más antigua que la versión del editor.
3. Envío: las entradas desalojadas, actualizadas, suprimidas o insertadas se añaden o se sobrescriben en todas las Máquinas virtuales Java de iguales al recibir el mensaje JMS.
4. Envío condicional: la entrada sólo se actualiza o se añade en el lado del receptor si la entrada local es menos reciente que la versión que se va a publicar.

Escuchar cambios de publicación

El plug-in implementa la interfaz ObjectGridEventListener para interceptar el suceso transactionEnd. Cuando eXtreme Scale invoca este método, el plug-in intenta convertir la lista LogSequence de cada correlación manipulada por la transacción a un mensaje JMS, que intentará publicar. El plug-in puede haberse configurado para publicar cambios de todas las correlaciones o de un subconjunto. Los objetos LogSequence se procesan para las correlaciones que tienen habilitada la publicación. La clase LogSequenceTransformer ObjectGrid serializa un objeto filtrado LogSequence de cada correlación en una corriente. Después de que todos los objetos LogSequences se serialicen en una corriente, se crea un objeto JMS ObjectMessage y se publica para un tema conocido.

Escuchar mensajes JMS y aplicarlos al objeto ObjectGrid local

El mismo plug-in también inicia una hebra que forma un bucle y recibe todos los mensajes publicados para un tema conocido. Cuando llega un mensaje, se pasa el contenido del mensaje a la clase LogSequenceTransformer, donde se convierte a un conjunto de objetos LogSequence. A continuación, se inicia una transacción de no escritura a través. Cada objeto LogSequence se proporciona al método Session.processLogSequence, que actualiza las correlaciones locales con los cambios. El método processLogSequence entiende la modalidad de distribución. La transacción se confirma y la memoria caché local refleja los cambios. Si desea más información sobre cómo utilizar JMS para distribuir cambios de transacción, consulte la información sobre cómo distribuir los cambios entre máquinas virtuales Java iguales en *Guía de administración*.

Transacciones de partición única y de partición entre cuadrícula

La diferencia principal entre WebSphere eXtreme Scale y las soluciones de almacenamiento de datos tradicionales como las bases de datos relacionales o las bases de datos en memoria es el uso del particionamiento, que permite a la memoria caché realizar las escaladas de forma lineal. Los tipos importantes de transacciones para tener en cuenta son las transacciones de partición única y las muchas particiones (entre cuadrícula).

En general, las interacciones con la memoria caché se pueden categorizar como transacciones de partición única o transacciones entre cuadrícula, tal como se describe a continuación.

Transacciones de partición única

Las transacciones de partición única son el método preferible para interactuar con las memorias caché alojadas por WebSphere eXtreme Scale. Cuando una transacción está limitada a una única partición, de forma predeterminada, está limitada a una única Máquina virtual Java y, por lo tanto, un único sistema de servidor. Un servidor puede completar *M* número de estas transacciones por segundo y si tiene *N* sistemas, puede completar `M*N` transacciones por segundo. Si el negocio aumenta y debe doblar el rendimiento respecto a muchas de estas transacciones por segundo, puede doblar el valor *N* comprando más sistemas. Puede cumplir las demandas de capacidad sin modificar la aplicación, actualizar el hardware o, incluso, colocando la aplicación fuera de línea.

Además de permitir a la memoria caché realizar escaladas de forma significativa, las transacciones de partición única también maximizan la disponibilidad de la memoria caché. Cada transacción sólo depende de un sistema. Cualquiera de los

otros (N-1) sistemas puede falla sin que esto afecte al éxito o al tiempo de respuesta de la transacción. Por lo tanto, si ejecuta 100 sistemas y uno de ellas falla, sólo el 1 por ciento de las transacciones en curso en el momento en que falla el servidor se retrotrae. Después de que el servidor falle, WebSphere eXtreme Scale reubica las particiones alojadas por el servidor anómalo en los otros 99 sistemas. Durante este breve periodo, antes de que se complete la operación, los otros 99 sistemas pueden seguir completando transacciones. Sólo las transacciones que podrían implicar que las particiones que se están reubicando se bloqueen. Después de que se complete el proceso de migración tras error, la memoria caché puede seguir ejecutándose, plenamente operativa a un 99 por ciento de su capacidad de rendimiento original. Después de que se sustituya el servidor anómalo y se devuelva a la cuadrícula, la memoria caché vuelve al 100 por ciento de capacidad de rendimiento.

Transacciones entre cuadrícula

En términos de rendimiento, disponibilidad y escalabilidad, las transacciones entre cuadrícula son el opuesto de las transacciones de partición única. Las transacciones entre cuadrícula acceden a cada partición y, por lo tanto, a todos los sistemas de la configuración. Se solicita a cada sistema de la cuadrícula que busque algunos datos y, a continuación, devuelva el resultado. La transacción no se puede completar hasta que hayan contestado todos los sistemas y, por lo tanto, el rendimiento de toda la cuadrícula está limitado al sistemas más lento. Añadir sistemas no hace que el sistema más lento sea más rápido y, por lo tanto, no mejora el rendimiento de la memoria caché.

Las transacciones entre cuadrícula tienen un efecto similar en la disponibilidad. Ampliando el ejemplo anterior, si ejecuta 100 servidores y uno falla, el 100 por ciento de las transacciones que están en curso en el momento en el que falló el servidor se retrotraen. Después de que falle el servidor, WebSphere eXtreme Scale empieza a reubicar las particiones alojadas por dicho servidor a los otros 99 sistemas. Durante este tiempo, antes de que se complete el proceso de migración tras error, la cuadrícula no puede procesar ninguna de estas transacciones. Después de que se complete el proceso de migración tras error, la memoria caché puede seguir ejecutándose, pero a una capacidad reducida. Si cada sistema de la cuadrícula presta servicio a 10 particiones, 10 de los 99 sistemas restantes reciben, como mínimo, una partición adicional como parte del proceso de migración tras error. Añadir una partición adicional aumentar la carga de trabajo de dicho sistema en un 10 por ciento, como mínimo. Puesto que el rendimiento de la cuadrícula está limitado al rendimiento del sistema más lento en una transacción entre cuadrícula, de promedio, el rendimiento se reduce en un 10 por ciento.

Las transacciones de partición única son preferibles que las transacciones entre cuadrícula para escalar de forma horizontal con una memoria caché de objeto distribuida y con alta disponibilidad como WebSphere eXtreme Scale. Maximizar el rendimiento de estos tipos de sistemas requiere el uso de técnicas que son diferentes a las metodologías relacionales tradicionales, pero puede convertir las transacciones entre cuadrícula en transacciones de partición única.

Procedimientos recomendados para crear modelos de datos escalables

Los procedimientos recomendados para crear aplicaciones escalables con productos como WebSphere eXtreme Scale incluyen dos categorías: los principios fundacionales y las sugerencias de implementación. Los principios fundacionales son ideas principales que se deben capturar en el diseño de los propios datos. Una

aplicación que no observa estos principios probablemente no realizará bien las escaladas, incluso para sus transacciones principales. Se aplican las sugerencias de implementación para las transacciones problemáticas en una aplicación bien diseñada de otra forma que observa los principios generales para los modelos de datos escalables.

Principios fundacionales

Algunos de los métodos importantes para optimizar la escalabilidad son conceptos o principios básicos que se deben tener en cuenta.

Duplicar en lugar de normalizar

El concepto clave para recordar sobre los productos como WebSphere eXtreme Scale es que se han diseñado para distribuir los datos entre un gran número de sistemas. Si el objetivo es completar la mayoría o todas las transacciones en una única partición, el diseño del modelo de datos debe garantizar que todos los datos que podría necesitar la transacción se encuentran en la partición. La mayoría del tiempo, la única forma de conseguir esto es duplicando los datos.

Por ejemplo, considere una aplicación como un tablón de mensajes. Dos transacciones muy importantes para un tablón de mensajes son mostrar todas las publicaciones de un usuario proporcionado y todas las publicaciones sobre un tema determinado. En primer lugar, considere cómo estas transacciones funcionarían con un modelo de datos normalizado que contiene un registro de usuarios, un registro de temas y un registro de publicaciones que contiene el texto real. Si las publicaciones se particionan con registros de usuarios, la visualización del tema pasa a ser una transacción entre cuadrícula y viceversa. Los temas y los registros no se pueden particionar juntos porque tienen una relación de muchos a muchos.

El mejor método para realizar esta escalada del tablón de mensajes es duplicar las publicaciones, almacenando una copia con el registro de temas y una copia con el registro de usuarios. A continuación, la visualización de las publicaciones de un usuario es una transacción de partición única, la visualización de las publicaciones sobre un tema es una transacción de partición única y la actualización o la supresión de una publicación es una transacción de dos particiones. Todas estas tres transacciones realizarán las escaladas de forma lineal, ya que aumenta el número de sistemas de la cuadrícula.

Escalabilidad en lugar de recursos

El mayor obstáculo para superar cuando se considera eliminar la normalización de los modelos de datos es el impacto que estos modelos tendrían en los recursos. Podría parecer que conservar dos, tres o más copias de algunos datos utiliza demasiados recursos para que sea práctico. Cuando lo confronta con este escenario, recuerde los siguientes hechos: los recursos de hardware son más baratos cada año. En segundo lugar, y más importante, WebSphere eXtreme Scale elimina los costes más ocultos asociados al despliegue de más recursos.

Medir los recursos en términos de coste, en lugar de en términos de sistema como, por ejemplo, megabytes y procesadores. Generalmente, los almacenes de datos que funcionan con datos relacionales normalizados deben estar situados en el mismo sistema. Esta ubicación necesaria significa que se debe adquirir un único gran sistema empresarial, en lugar de varios sistemas pequeños. Con el hardware de empresa, no es raro que

un sistema capaz de completar un millón de transacciones por segundo cueste muchos más que el coste combinado de 10 sistemas capaces de realizar 100.000 transacciones por segundos cada uno.

También existe un coste empresarial en la adición de recursos. Una negocio creciente acaba por agotar la capacidad. Cuando se agota la capacidad, debe concluir mientras se traslada a un sistema mayor y más rápido, o bien crear un segundo entorno de producción al que se puede pasar. De cualquier modo, los costes adicionales vendrán en forma de pérdidas de negocio o en el mantenimiento de casi el doble de la capacidad necesaria durante el periodo de transacción.

Con WebSphere eXtreme Scale, no es necesario concluir la aplicación para añadir capacidad. Si su empresa planea que necesita un 10 por ciento más capacidad para el año que viene, aumente el número de sistemas de la cuadrícula en un 10 por ciento. Puede aumentar este porcentaje sin tiempo de inactividad de la aplicación y sin adquirir excesiva capacidad.

Evitar transformaciones de datos

Cuando se utiliza WebSphere eXtreme Scale, los datos se deben almacenar en un formato que pueda consumir directamente la lógica empresarial. Desglosar los datos en un formato más primitivo es costoso. La transformación se debe realizar cuando los datos se escriben y cuando los datos se leen. Con las bases de datos relacionales, esta transformación se realiza por necesidad, porque los datos se persisten de forma última en el disco con bastante frecuencia, pero con WebSphere eXtreme Scale, no es necesario que realice estas transformaciones. Para la mayoría de las partes, los datos se almacenan en la memoria y, por lo tanto, se almacenan en el formato exacto que necesita la aplicación.

Observar esta regla simple le ayuda a eliminar la normalización de los datos de acuerdo con el primer principio. El tipo más común de transformación para los datos empresariales es las operaciones JOIN que son necesarias para convertir los datos normalizados en un conjunto de resultados que se ajuste a las necesidades de la aplicación. Almacenar los datos en el formato correcto impide de forma implícita realizar estas operaciones JOIN y genera un modelo de datos no normalizados.

Eliminar consultas no enlazadas

Independientemente de cómo se estructuren los datos, las consultas no enlazadas no se escalan bien. Por ejemplo, no se tiene una transacción que solicite una lista de todos los elementos ordenados por un valor. Esta transacción podría funcionar a la primera, si el número total de elementos es 1000, pero si el número total de elementos llega a 10 millones, la transacción devuelve todos los 10 millones de elementos. Si ejecuta esta transacción, los dos resultados más probables son que la transacción agote el tiempo o que el cliente encuentre un error de memoria agotada.

La mejor opción es alterar la lógica empresarial de forma que sólo se puedan devolver los 10 o 20 primeros elementos. La alteración de la lógica mantiene el tamaño de la transacción gestionable, independientemente de cuántos elementos contenga la memoria caché.

Definir esquema

La principal ventaja de normalizar los datos es que el sistema de la base de datos puede ocuparse de la coherencia de los datos de forma interna. Cuando se elimina la normalización de los datos para la escalabilidad, deja de existir esta gestión automática de la coherencia de los datos. Debe

implementar un modelo de datos que pueda funcionar en la capa de la aplicación o como un plug-in en la cuadrícula distribuida para garantizar la coherencia de los datos.

Considere el ejemplo del tablón de mensajes. Si una transacción elimina una publicación de un tema, se debe eliminar la publicación duplicada del registro de usuarios. Sin un modelo de datos, es posible que un desarrollador escriba el código de la aplicación para eliminar la publicación del tema y olvide eliminar la publicación del registro de usuarios. Sin embargo, si el desarrollador estuviera utilizando un modelo de datos, en lugar de interactuando directamente con la memoria caché, el método removePost en el modelo de datos podría extraer el ID de usuario de la publicación, buscar el registro de usuarios y eliminar la publicación duplicada de forma interna.

De forma alternativa, puede implementar un receptor que se ejecuta en la partición real que detecta el cambio en el tema y ajusta automáticamente el registro de usuarios. Un receptor podría ser beneficioso porque el ajuste del registro de usuarios se podría realizar de forma local si la partición parece tener el registro de usuarios, o incluso si el registro de usuarios está en una partición distinta, la transacción se produce entre los servidores, en lugar de entre el cliente y el servidor. Probablemente, la conexión de red entre los servidores es más rápida que la conexión de red entre el cliente y el servidor.

Impedir la competencia

Se impiden escenarios como tener un contador global. La cuadrícula no se escalará si un único registro utiliza un número desproporcionado de tiempos en comparación con el resto de los registros. El rendimiento de la cuadrícula se limitará por el rendimiento del sistema que aloja el registro determinado.

En estas situaciones, intente dividir el registro para que sea gestionado por partición. Por ejemplo, considere una transacción que devuelve el número total de entradas en la memoria caché distribuida. En lugar de tener cada acceso de la operación insert y remove en un único registro que aumenta, tener un receptor en cada partición rastrea las operaciones insert y remove. Con este rastreo del receptor, las operaciones insert y remove se pueden convertir en operaciones de partición única.

La lectura del contador se convertirá en una operación entre cuadrícula, pero para la mayor parte, ya era tan ineficaz como una operación entre cuadrícula, porque su rendimiento estaba unido al rendimiento del sistema que incluye el registro.

Sugerencias de implementación

También puede considerar las siguientes sugerencias para conseguir la mejor escalabilidad.

Utilizar los índices de búsqueda inversa

Considere un modelo de datos que ha eliminado la normalización correctamente donde los registros de clientes se particionan basándose en el número del ID de cliente. Este método de particionamiento es la opción lógica porque casi todas las operaciones empresariales realizadas con el registro de clientes utilizan el número del ID del cliente. Sin embargo, una transacción importante que no utiliza el número del ID del cliente es la

transacción de inicio de sesión. Es más común tener nombres de usuario o direcciones de correo electrónico para el inicio de sesión, en lugar de números de ID de cliente.

El enfoque sencillo del escenario de inicio de sesión es utilizar una transacción entre cuadrícula para encontrar el registro de clientes. Tal como se ha explicado previamente, este enfoque no realiza escaladas.

La siguiente opción podría ser la partición en el nombre del usuario o el correo electrónico. Esta opción no es práctica porque todos las operaciones basadas en el ID de cliente se convierten en transacciones entre cuadrícula. Asimismo, los clientes del sitio podrían desear cambiar su nombre de usuario o dirección de correo electrónico. Los productos como WebSphere eXtreme Scale necesitan el valor que se utiliza para la partición de datos en constantes de permanencia.

La solución correcta es utilizar un índice de búsqueda inversa. Con WebSphere eXtreme Scale, se puede crear una memoria caché en la misma cuadrícula distribuida que la memoria caché que aloja todos los registros de usuarios. Esta memoria caché es altamente disponible, está particionada y es escalable. Se puede utilizar esta memoria caché para correlacionar un nombre de usuario o dirección de correo electrónico con un ID de cliente. Esta memoria caché convierte el inicio de sesión en una operación de dos particiones, en lugar de una operación entre cuadrícula. Este escenario no es tan bueno como una transacción de partición única, pero el rendimiento se sigue escalando de forma lineal a medida que el número de sistemas aumenta.

Calcular en el momento de la escritura

Los valores calculados comúnmente como promedios o totales pueden resultar caros para generarse, porque normalmente estas operaciones requieren leer un gran número de entradas. Puesto que leer es más comunes que escribir en la mayoría de las aplicaciones, es eficaz calcular estos valores en el momento de escribir y, a continuación, almacenar el resultado en la memoria caché. Esta práctica hace que las operaciones de lectura sean más rápidas y más escalables.

Campos opcionales

Considere un registro de usuarios que incluya una empresa, un lugar y un número de teléfono. Un usuario puede tener todos estos números definidos, o ninguno o alguna combinación de éstos. Si los datos se normalizaron, podría existir una tabla de usuarios y una tabla de números de teléfono. Los números de teléfono para un usuario determinado se podrían encontrar utilizando una operación JOIN entre las dos tablas.

Eliminar la normalización de este registro no requiera la duplicación de datos, porque la mayoría de los usuarios no comparten los números de teléfono. En lugar de esto, debe estar permitido vaciar las ranuras del registro de usuarios. En lugar de tener una tabla de números de teléfono, añada tres atributos a cada registro de usuarios, uno para cada tipo de número de teléfono. Esta adición de atributos elimina la operación JOIN y realiza una búsqueda de número de teléfono para un usuario y una operación de partición única.

Colocación de relaciones de muchos a muchos

Considere una aplicación que rastrea los productos y las tiendas en las que se venden los productos. Un único producto se vende en muchas tiendas y una sola tienda vende muchos productos. Suponga que esta aplicación

rastrea 50 tiendas grandes. Cada producto se vende en un máximo de 50 tiendas, con cada tienda que vende miles de productos.

Conservar una lista de tiendas dentro de la entidad de producto (disposición A), en lugar de conservar una lista de productos dentro de cada entidad de tienda (disposición B). Consultando algunas de las transacciones, esta aplicación podría realizar ilustraciones que expliquen por qué la disposición A es más escalable.

En primer lugar, consulte las actualizaciones. Con la disposición A, eliminar un producto del inventario de una tienda bloquea la entidad del producto. Si la cuadrícula contiene 10.000 productos, sólo el 1/10000 de la cuadrícula se debe bloquear para realizar la actualización. Con la disposición B, la cuadrícula sólo contiene 50 tiendas, así que el 1/50 de la cuadrícula debe estar bloqueada para completar la actualización. Así pues, aunque ambas disposiciones se podrían considerar operaciones de partición única, la disposición A se escala de forma horizontal de forma más eficaz.

Ahora, si se consideran las lecturas con la disposición A, buscar las tiendas en las que se vende un producto es una transacción de partición única que se escala y es rápida porque la transacción sólo transmite una pequeña cantidad de datos. Con la disposición B, esta transacción pasa a ser una transacción entre cuadrícula porque se debe acceder a cada entidad de tienda para ver si el producto se vende en dicha tienda, que implica una gran ventaja de rendimiento para la disposición A.

Escalar con datos normalizados

Un uso legítimo de las transacciones entre cuadrícula es escalar el proceso de datos. Si una cuadrícula tiene 5 sistemas y se envía una transacción entre cuadrícula que clasifica unos 100.000 registros en cada sistema, dicha transacción clasifica unos 500.000 registros. Si el sistema más lento de la cuadrícula puede realizar 10 de estas transacciones por segundo, la cuadrícula es capaz de realizar clasificaciones entre 5.000.000 de registros por segundo. Si los datos de la cuadrícula se doblan, cada sistema realiza una clasificación entre 200.000 registros y cada transacción realiza una clasificación entre 1.000.000 de registros. Estos datos reducen el rendimiento del sistema más lento a 5 transacciones por segundo, por lo tanto, reduce el rendimiento de la cuadrícula a 5 transacciones por segundo. La cuadrícula realiza la clasificación entre 5.000.000 de registros por segundo.

En este escenario, doblar el número de sistemas permite a cada sistema volver a su carga previa de clasificación entre 100.000 registros, lo que permite al sistema más lento procesar 10 de estas transacciones por segundo. El rendimiento de la cuadrícula permanece igual a 10 peticiones por segundo, pero ahora cada transacción procesa 1.000.000 de registros, de forma que la cuadrícula ha doblado su capacidad en términos de proceso de registros a 10.000.000 por segundo.

Las aplicaciones como, por ejemplo, un motor de búsqueda que se debe escalar en términos de proceso de datos para adaptar el tamaño en crecimiento de Internet y el rendimiento para adaptar el crecimiento en el número de usuarios, debe crear varias cuadrículas con un turno circular de las peticiones entre las cuadrículas. Si debe escalar de forma vertical el rendimiento, añada sistemas y añada otra cuadrícula a las solicitudes de servicio. Si el proceso de datos se debe escalar de forma vertical, añada más sistemas y mantenga constante el número de cuadrículas.

Capítulo 7. Visión general de seguridad

WebSphere eXtreme Scale puede proteger el acceso a los datos, incluida la posibilidad de integración con proveedores de datos externos.

Nota: En un almacén de datos no almacenado en memoria caché existente, como una base de datos, probablemente, tendrá características de seguridad incorporadas que podría necesitar para configurar o habilitar de forma activa. No obstante, después de haber almacenado en memoria caché los datos con eXtreme Scale, debe considerar la situación resultante importante de que las características de seguridad del programa de fondo ya no están en vigor. Puede configurar la seguridad de eXtreme Scale en los niveles necesarios de modo que la nueva arquitectura almacenada en memoria caché para los datos también esté protegida.
A continuación, aparece un breve resumen de las características de seguridad de eXtreme Scale. Si desea más información detallada sobre cómo configurar la seguridad, consulte *Guía de administración* y *Guía de programación*.

Conceptos básicos de la seguridad distribuida

La seguridad distribuida de eXtreme Scale se basa en tres conceptos clave:

Autenticación de confianza
: La capacidad de determinar la identidad del solicitante. WebSphere eXtreme Scale da soporte a la autenticación de cliente a servidor y servidor a servidor.

Autorización
: La capacidad de dar permisos para otorgar derechos de acceso al solicitante. WebSphere eXtreme Scale da soporte a distintas autorizaciones para diversas operaciones.

Transporte seguro
: La transmisión segura de datos a través de una red. WebSphere eXtreme Scale soporta los protocolos TLS/SSL (Transport Layer Security/Secure Sockets Layer).

Autenticación

WebSphere eXtreme Scale da soporte a la infraestructura distribuida de cliente-servidor. La infraestructura de seguridad de cliente-servidor existe para proteger el acceso a los servidores de eXtreme Scale. Por ejemplo, cuando el servidor eXtreme Scale requiere una autenticación, el cliente de eXtreme Scale debe proporcionar las credenciales para autenticar el servidor. Estas credenciales pueden ser un par de nombre de usuario y contraseña, un certificado de cliente, un ticket de Kerberos o datos que se presentan en un formato acordado por el cliente y el servidor.

Autorización

Las autorizaciones de WebSphere eXtreme Scale se basan en sujetos y permisos. Puede utilizar JAAS (Java Authentication and Authorization Services) para autoriza el acceso, o puede conectar un método personalizado, como Tivoli Access Manager (TAM), para manejar las autorizaciones. Pueden otorgarse las siguientes autorizaciones a un cliente o grupo:

Autorización de correlaciones
Realizar operaciones de inserción, lectura, actualización o supresión en correlaciones.

Autorización de ObjectGrid
Realizar consultas de objetos o entidades y consultas de secuencias en objetos ObjectGrid.

Autorización de agentes de DataGrid
Permitir que los agentes de DataGrid se desplieguen en un ObjectGrid.

Autorización de correlaciones del lado del servidor
Duplicar una correlación de servidor con el lado del cliente o crear un índice dinámico con la correlación de servidor.

Autorización de administración
Realizar tareas de administración.

Seguridad de transporte

Para proteger la comunicación del servidor cliente, WebSphere eXtreme Scale soporta TLS/SSL. Estos protocolos proporcionan el nivel de seguridad de la capa de transporte con la autenticidad, integridad y confidencialidad para una conexión segura entre un cliente y un servidor de eXtreme Scale.

Seguridad de la cuadrícula

En un entorno seguro, un servidor debe poder comprobar la autenticidad de otro servidor. Para ello WebSphere eXtreme Scale utiliza un mecanismo de serie de clave secreta compartida. Este mecanismo de clave secreta es parecido a una contraseña secreta. Todos los servidores de eXtreme Scale acuerdan una serie secreta compartida. Cuando un servidor se une a la cuadrícula, el servidor se ve obligado a presentar la serie secreta. Si la serie secreta del servidor que se une coincide con una del servidor maestro, este servidor se puede unir a la cuadrícula. De lo contrario, la solicitud se rechaza.

El envío de una serie secreta en texto normal no es seguro. La infraestructura de seguridad de eXtreme Scale proporciona un plug-in SecureTokenManager para permitir al servidor proteger este secreto antes de enviarlo. Puede elegir cómo implementar la operación segura. WebSphere eXtreme Scale proporciona una implementación, en la que se implementa la operación segura para cifrar y firmar la serie secreta.

Seguridad JMX (Java Management Extensions) en una topología de despliegue dinámico

La seguridad de JMX MBean recibe soporte en todas las versiones de eXtreme Scale. Los clientes de MBeans de servidor de catálogo y MBeans de servidor de contenedor pueden autenticarse, y se puede forzar el acceso a operaciones de MBean.

Seguridad de eXtreme Scale local

La seguridad de eXtreme Scale local es distinta del modelo de eXtreme Scale distribuido porque la aplicación crea una instancia y utiliza una instancia de ObjectGrid directamente. La aplicación y las instancias de eXtreme Scale están en la misma JVM (Java Virtual Machine). Puesto que no hay ningún concepto de cliente-servidor en este modelo, no se da soporte a la autenticación. Las

aplicaciones deben gestionar su propia autenticación y, a continuación, pasar el objeto Subject autenticado aeXtreme Scale. Sin embargo, el mecanismo de autorización que se utiliza para el modelo de programación de eXtreme Scale local es el mismo que se ha utilizado para el modelo cliente-servidor.

Configuración y programación

Si desea más información sobre cómo configurar y programar la seguridad, consulte el *Guía de administración* y *Guía de programación*.

Capítulo 8. Visión general de los servicios de datos REST

El servicio de datos REST de WebSphere eXtreme Scale es un servicio HTTP Java compatible con Microsoft WCF Data Services (anteriormente ADO.NET Data Services) e implementa el Protocolo de datos abierto (OData). Microsoft WCF Data Services es compatible con esta especificación al utilizar Visual Studio 2008 SP1 y .NET Framework 3.5 SP1.

Requisitos de compatibilidad

El servicio de datos REST permite a cualquier cliente HTTP acceder a una cuadrícula de datos. El servicio de datos REST es compatible con el soporte de WCF Data Services que se proporciona con Microsoft .NET Framework 3.5 SP1. Se pueden desarrollar aplicaciones RESTful con las útiles herramientas proporcionadas por Microsoft Visual Studio 2008 SP1. En la figura se proporciona una visión general de cómo interactúa WCF Data Services con clientes y bases de datos.

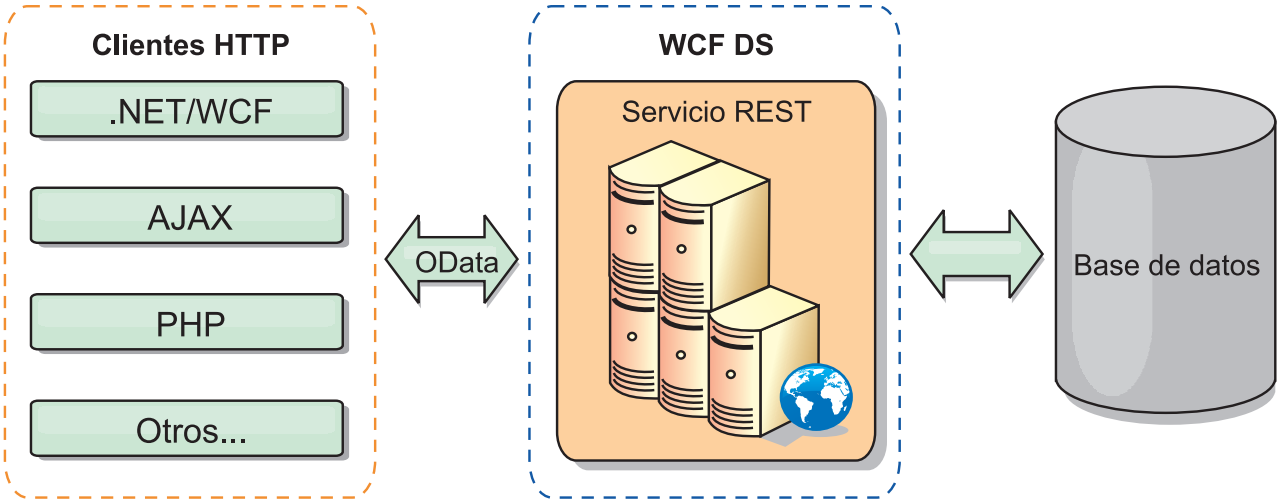


Figura 41. Microsoft WCF Data Services

WebSphere eXtreme Scale incluye una API con muchas funciones para clientes Java. Como se muestra en la figura siguiente, el servicio de datos REST es un pasarela entre los clientes HTTP y la cuadrícula WebSphere eXtreme Scale, que se comunica con la cuadrícula mediante un cliente de WebSphere eXtreme Scale. El servicio de datos REST es un servlet Java, que permite despliegues flexibles para plataformas Java Platform, Enterprise Edition (JEE) comunes, como WebSphere Application Server. El servicio de datos REST se comunica con la cuadrícula WebSphere eXtreme Scale mediante las API Java de WebSphere eXtreme Scale. Permite los clientes de WCF Data Services o cualquier otro cliente que pueda comunicarse con HTTP y XML.

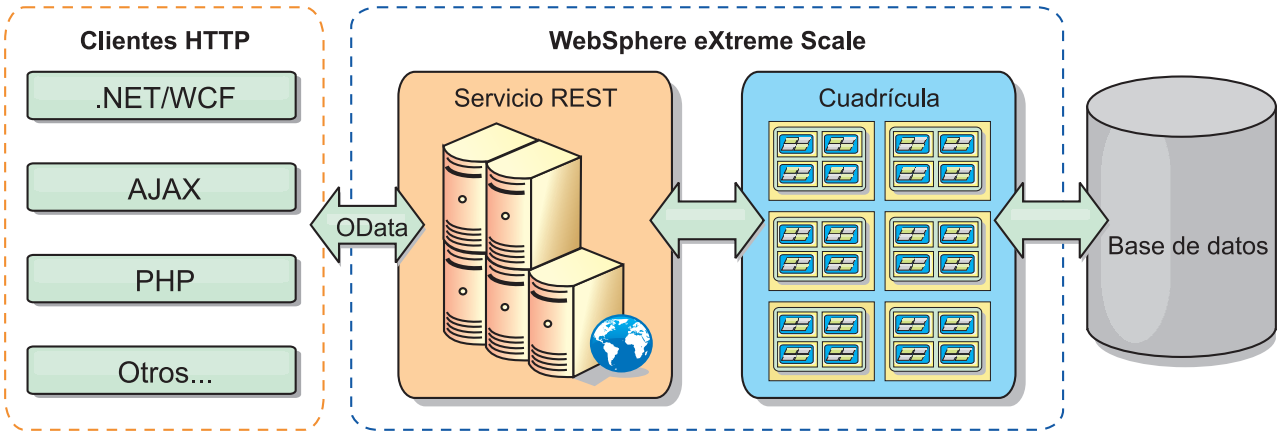


Figura 42. Servicio de datos REST de WebSphere eXtreme Scale

Consulte "Ejemplo de servicios de datos REST y guía de aprendizaje" en la página 221, o utilice los enlaces siguientes para obtener más información sobre WCF Data Services.

- Microsoft WCF Data Services Developer Center
- Visión general de ADO.NET Data Services en MSDN
- Libro blanco: Uso de ADO.NET Data Services
- Protocolo de publicación Atom: URI de servicios de datos y ampliaciones de la carga útil
- Formato de archivo de definición de esquema conceptual
- Formato de empaquetado del modelo de datos de entidad para servicios de datos
- Protocolo de datos abierto
- Preguntas frecuentes sobre el protocolo de datos abierto

Características

Esta versión del servicio de datos REST de eXtreme Scale soporta las características siguientes:

- Modelado automático de entidades de API EntityManager de eXtreme Scale como entidades de WCF Data Services que incluye el soporte siguiente:
  - Conversión del tipo de datos Java al tipo de modelo de datos de entidad
  - Soporte para la asociación de entidades
  - Soporte para la asociación de raíces de esquema y claves, necesario para cuadrículas de datos particionadas

  Consulte el apartado Modelo de entidad si desea más información.
- Atom Publish Protocol (AtomPub o APP) XML y formato de carga útil de datos JavaScript™ Object Notation (JSON).
- Operaciones de creación, lectura, actualización y supresión (CRUD) utilizando los respectivos métodos de solicitud HTTP: POST, GET, PUT y DELETE. Además, la ampliación de Microsoft: MERGE está soportada.
- Consultas simples, con filtros
- Solicitudes de recuperación por lotes y conjuntos de cambios
- Soporte para cuadrículas particionadas para alta disponibilidad
- Interoperatividad con clientes de la API EntityManager de eXtreme Scale

- Soporte para servidores web JEE estándar
- Simultaneidad optimista
- Autorización y autenticación de usuarios entre el servicio de datos REST y la cuadrícula de datos eXtreme Scale

Problemas y limitaciones conocidos

- Las solicitudes a través de túnel no están soportadas.

Capítulo 9. Visión general de integración en la infraestructura de Spring

Spring es una infraestructura popular para desarrollar las aplicaciones Java. WebSphere eXtreme Scale proporciona soporte para permitir a Spring gestionar las transacciones de eXtreme Scale y configurar los clientes y servidores que conforman la cuadrícula de datos en memoria desplegada.

Transacciones nativas gestionadas de Spring

Spring proporciona transacciones gestionadas por contenedor que son similares al servidor de aplicaciones Java Platform, Enterprise Edition. Sin embargo, el mecanismo Spring se puede conectar a distintas implementaciones. WebSphere eXtreme Scale proporciona una integración del gestor de transacciones que permite a Spring gestionar los ciclos de vida de transacción de ObjectGrid. Consulte la información sobre las transacciones nativas en *Guía de programación*, para buscar detalles.

Beans de ampliación gestionados de Spring y soporte de espacio de nombres

Además, eXtreme Scale se integra con Spring para habilitar a los beans de estilo Spring definidos para los puntos o plug-ins de ampliación. Esta característica proporciona configuraciones más sofisticadas y más flexibilidad para configurar los puntos de ampliación.

Además de los beans de amplicación gestionados de Spring, eXtreme Scale proporciona un espacio de nombres Spring denominado "objectgrid". Los beans y las implementaciones incorporadas están definidos previamente en este espacio de nombres, que hace que sea más fácil para los usuarios configurar eXtreme Scale. Consulte Beans de ampliación de Spring y soporte de espacio de nombres, si desea más detalles sobre estos temas y un ejemplo sobre cómo iniciar un servidor de contenedor de eXtreme Scale utilizando las configuraciones de Spring.

Soporte de ámbito de fragmento

Con la configuración de Spring de estilo tradicional, un bean ObjectGrid puede ser un tipo singleton o un tipo de prototipo. Además, ObjectGrid soporta un nuevo ámbito denominado el ámbito de "fragmento". Si un bean está definido como ámbito de fragmento, sólo se crea un bean por fragmento. Todas las solicitudes para los beans con un ID o IDs que coinciden con dicha definición de bean en el mismo fragmento generarán que una instancia de bean específica sea devuelta por el contenedor Spring.

El siguiente ejemplo muestra que un bean com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl está definido con el ámbito establecido en shard (fragmento). Por lo tanto, sólo se crea una instancia de la clase JPAPropFactoryImpl por fragmento.

```
<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl" scope="shard" />
```

Flujo web de Spring

El flujo web de Spring almacena su estado de sesión en la sesión HTTP de forma predeterminada. Si una aplicación web se configura para utilizar eXtreme Scale para la gestión de sesiones, Spring lo utiliza automáticamente para almacenar su estado y se convierte en tolerante a errores del mismo modo que la sesión.

Empaquetado

Las extensiones Spring de eXtreme Scale están en el archivo `ogspring.jar`. Este archivo Java (JAR) debe estar en la classpath para trabajar con el soporte de Spring. Si una aplicación JEE se está ejecutando en un WebSphere Extended Deployment aumentado WebSphere Application Server Network Deployment, la aplicación deberá colocar el archivo `spring.jar` y sus archivos asociados en los módulos archivadores empresariales (EAR). También debe colocar el archivo `ogspring.jar` en la misma ubicación.

Capítulo 10. Guías de aprendizaje, ejemplos y muestras

Hay varias guías de aprendizaje, ejemplos y muestras de WebSphere eXtreme Scale disponibles.

Guías de aprendizaje

Están disponible las siguientes guías de aprendizaje.

- Guía de aprendizaje de ObjectMap
- "Guía de aprendizaje del gestor de entidades: visión general"
-
-

Ejemplos

En los temas siguientes se muestran las características de WebSphere eXtreme Scale clave.

- Consulte el ejemplo de la API de Data Grid en la *Guía de programación*
- Consulte los detalles sobre cómo configurar los despliegues locales en la *Guía de administración*

Ejemplos

Con el producto WebSphere eXtreme Scale se entregan ejemplos que ilustran cómo utilizar API de ObjectGrid en distintos entornos.

Artículos con guías de aprendizaje y ejemplos

Tabla 15. Artículos disponibles por característica

Artículo	Características
Building grid-ready applications	API ObjectMap, API EntityManager, Consulta, Agentes, Java SE y EE, Estadísticas, Particionamiento, Administración/Operaciones, Eclipse
Scalable grid-style computing and data processing	API EntityManager, Agentes
Building a scalable, resilient, high-performance database alternative	API ObjectMap, Réplica, Particionamiento, Administración/Operaciones, Eclipse
Enhancing xsadmin for WebSphere eXtreme Scale	Administración
Redbook: User's Guide	Todos los temas

Guía de aprendizaje del gestor de entidades: visión general

La guía de aprendizaje para el gestor de entidades le muestra cómo utilizar WebSphere eXtreme Scale para almacenar la información de pedidos en un sitio web. Puede crear una aplicación Java Platform, Standard Edition 5 sencilla que utiliza un eXtreme Scale local en memoria local. Las entidades utilizan genéricos y anotaciones Java SE 5.

Antes de empezar

Asegúrese de que satisface los siguientes requisitos antes de empezar la guía de aprendizaje:

- Debe tener Java SE 5.
- Debe tener el archivo `objectgrid.jar` en la vía de acceso de clases.

Guía de aprendizaje del gestor de entidades: creación de una clase de entidad

El primer paso de la guía de aprendizaje del gestor de entidades le muestra cómo crear un ObjectGrid local con una entidad mediante la creación de una clase de entidad, registrando el tipo de entidad con eXtreme Scale y almacenando una instancia de entidad en la memoria caché.

Acerca de esta tarea

Procedimiento

1. Cree el objeto Order. Para identificar el objeto como una entidad ObjectGrid, añada la anotación @Entity. Cuando añade esta anotación, todos los atributos serializables del objeto persisten automáticamente en eXtreme Scale, salvo que utilice anotaciones en los atributos para alterar temporalmente los atributos. El atributo orderNumber lleva la anotación @Id para indicar que este atributo es la clave primaria. A continuación se muestra un ejemplo de un objeto Order:

   Order.java

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

2. Ejecute la aplicación eXtreme Scale Hello World para demostrar las operaciones de entidad. El siguiente programa de ejemplo se puede emitir en modalidad autónoma para demostrar las operaciones de entidad. Utilice este programa en un proyecto Eclipse Java que tiene el archivo `objectgrid.jar` añadido a la classpath. A continuación se muestra un ejemplo de una aplicación Hello World simple que utiliza 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();
    }
}
```

Esta aplicación de ejemplo lleva a cabo las siguientes operaciones:

a. Inicializa un eXtreme Scale local con un nombre generado automáticamente.
b. Registra las clases de entidad con la aplicación utilizando la API registerEntities, aunque no siempre necesario utilizar la API registerEntities.
c. Recupera un elemento Session y una referencia al gestor de entidades para Session.
d. Asocia cada Session de eXtreme Scale con un solo EntityManager y EntityTransaction. Ahora se utiliza EntityManager.
e. El método registerEntities crea un objeto BackingMap que se llama Order, y asocia los metadatos para el objeto Order con el objeto BackingMap. Estos metadatos incluyen los atributos de clave y no de clave, junto con los nombres y tipos de atributos.
f. Se inicia una transacción y crea una instancia Order. La transacción se llena con algunos valores y se conserva utilizando el método EntityManager.persist, que identifica la entidad como en espera para ser incluida en la correlación ObjectGrid asociada.
g. A continuación, la transacción se confirma y la entidad se incluye en ObjectMap.
h. Se ejecuta otra transacción y se recupera el objeto Order utilizando la clave 1. La conversión de tipo de datos en el método EntityManager.find es necesaria porque la prestación de genéricos de Java SE 5 no se utiliza para garantizar que el archivo `objectgrid.jar` funcione en una Máquina virtual Java Java SE 1.4 y posterior.

Guía de aprendizaje del gestor de entidades: creación de relaciones de entidad

Crear una relación simple entre entidades creando dos clases de entidad con una relación, registrando las entidades con el ObjectGrid, y almacenando las instancias de entidades en la memoria caché.

Procedimiento

1. Cree la entidad customer, que se utiliza para almacenar los detalles del cliente independientemente del objeto Order. A continuación se muestra un ejemplo de la entidad customer:

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

Esta clase incluye información sobre el cliente, como el nombre, la dirección y el número de teléfono.

2. Cree el objeto Order, que es parecido al objeto Order del tema "Guía de aprendizaje del gestor de entidades: creación de una clase de entidad" en la página 188. A continuación se muestra un ejemplo del objeto Order:

   Order.java

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

   En este ejemplo, una referencia a un objeto Customer sustituye el atributo customerName. La referencia tiene una anotación que indica una relación de muchos con uno. Una relación de muchos con uno indica que cada pedido tiene un cliente, aunque varios pedidos pueden hacer referencia al mismo cliente. El modificador de anotación en cascada indica que si el gestor de entidades persiste el objeto Order, también debe persistir el objeto Customer. Si decide no establecer la opción de persistencia en cascada, que es la opción predeterminada, debe persistir manualmente el objeto Customer con el objeto Order.

3. Utilizando las entidades, defina las correlaciones para la instancia de ObjectGrid. Cada correlación se define para una entidad específica, y una entidad se denomina Order y la otra Customer. En la siguiente aplicación de ejemplo se muestra cómo almacenar y recuperar un pedido de cliente:

   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();
       }
}
```

Esta aplicación es parecida a la aplicación de ejemplo del paso anterior. En el ejemplo anterior, sólo se registra un objeto Order de una sola clase. WebSphere eXtreme Scale detecta e incluye automáticamente la referencia a la entidad Customer y se crea una instancia Customer para `John Smith`, a la que se hace referencia desde el nuevo objeto Order. Como resultado, el nuevo cliente se persiste automáticamente porque la relación entre dos pedidos incluye el modificador en cascada, que requiere que cada objeto sea persistente. Cuando se encuentra el objeto Order, el gestor de entidad encuentra automáticamente el objeto Customer asociado e inserta una referencia al objeto.

Guía de aprendizaje del gestor de entidades: esquema de entidades Order

Crear cuatro clases de entidades utilizando relaciones unidireccionales y bidireccionales, listas ordenadas y relaciones de claves foráneas. Las API de EntityManager se utilizan para persistir y encontrar las entidades. Basándose en las entidades Order y Customer que se encuentran en las partes anteriores de la guía de aprendizaje, este paso de la guía de aprendizaje añade dos entidades adicionales: las entidades Item y OrderLine.

Acerca de esta tarea

Figura 43. Esquema de entidades Order. Una entidad Order tiene una referencia a un cliente y cero o más OrderLines. Cada entidad OrderLine tiene una referencia a un sólo artículo e incluye la cantidad solicitada.

Procedimiento

1. Cree la entidad de cliente, que es parecida a los ejemplos anteriores.

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

2. Cree la entidad Item, que mantiene información sobre un producto incluido en el inventario de la tienda como, por ejemplo, la descripción del producto, la cantidad y el precio.

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

3. Cree una entidad OrderLine. Cada Order tiene cero o más OrderLines, que identifican la cantidad de cada artículo en el pedido. La clave para OrderLine es una clave compuesta que consta del Order que es propietario de la OrderLine y un entero que asigna un número a el elemento de línea. Añada el modificador de persistencia en cascada a cada relación de sus entidades.

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. Cree el objeto Order final, que tiene una referencia a Customer para el pedido y una colección de objetos 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;
   }
   ```

   Las cascada ALL se utiliza como modificador para líneas. Este modificador indica a EntityManager conectar en cascada la operación PERSIST y la operación REMOVE. Por ejemplo, si la entidad Order se mantiene o elimina, también se mantiene o eliminan todas las entidades OrderLine.

   Si una entidad OrderLine se elimina de la lista de líneas del objeto Order, la referencia se rompe. Sin embargo, la entidad OrderLine no se elimina de la memoria caché. Debe utilizar la API de supresión de EntityManager para eliminar entidades de la memoria caché. La operación REMOVE no se utiliza en la entidad de cliente o la entidad de artículo de OrderLine. Como resultado, la entidad de cliente permanece incluso si se elimina el pedido o el artículo cuando se elimina OrderLine.

   El modificador mappedBy indica una relación inversa con la entidad de destino. El modificador identifica qué atributo en la entidad de destino hace referencia a la entidad de origen, y el lado propietario de una relación uno con uno o muchos con muchos. En general podrá omitir el modificador. Si embargo, se visualiza un error para indicar que debe especificarse si WebSphere eXtreme Scale no puede descubrirlo automáticamente. Una entidad OrderLine que contiene dos tipos de atributos Order en una relación de muchos con uno, normalmente, genera el error.

   La anotación @OrderBy especifica el orden en el que cada entidad OrderLine debe aparece en la lista de líneas. Si la anotación no se especifica, las líneas aparecen en un orden arbitrario. Aunque las líneas se añaden a la entidad Order emitiendo ArrayList, que conserva el pedido, EntityManager no reconoce necesariamente el pedido. Cuando se emite el método find para recuperar el objeto Order de la memoria caché, el objeto de lista no es un objeto ArrayList.

5. Cree la aplicación. En el siguiente ejemplo se muestra el objeto Order final, que tiene una referencia a Customer para el pedido y una colección de objetos OrderLine.
   a. Busque los artículos a solicitar, que luego se convierten en entidades gestionadas.
   b. Cree el elemento de línea y adjúntelo a cada artículo.
   c. Cree un pedido y asócielo a cada elemento de línea y el cliente.
   d. Persiste el pedido, que persiste automáticamente cada elemento de línea.

e. Compromete la transacción, que desconecta cada entidad y sincroniza el estado de las entidades con la memoria caché.

f. Imprimir la información del pedido. Las entidades OrderLine se clasifican automáticamente por el ID de OrderLine.

Application.java

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

        // Añadir algunos elementos al inventario.
        em.getTransaction().begin();
        createItems(em);
        em.getTransaction().commit();

        // Crear un nuevo cliente con los artículos en su carro.
        em.getTransaction().begin();
        Customer cust = createCustomer();
        em.persist(cust);

        // Crear nuevo pedido y añadir un elemento de línea para cada artículo.
        // Cada elemento de línea se persiste automáticamente ya que la opción
    // Cascade=ALL está establecida.
Order order = createOrderFromItems(em, cust, "ORDER_1",
     new String[]{"1", "2"}, new int[]{1,3});
        em.persist(order);
        em.getTransaction().commit();

        // Imprimir el resumen de pedido
        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;
    }
```

El paso siguiente será suprimir una entidad. La interfaz EntityManager tiene un método remove que marca un objeto como suprimido. La aplicación debe eliminar la entidad de todas las colecciones de relaciones antes de llamar al método remove. Edite las referencias y emita el método remove, o em.remove(object), como último paso.

Guía de aprendizaje del gestor de entidades: actualización de entradas

Si desea cambiar una entidad, puede buscar la instancia, actualizar la instancia y todas las entidades referenciadas, y confirmar la transacción.

Procedimiento

Actualice entradas. En el siguiente ejemplo se muestra cómo buscar la instancia de Order, cambiar la instancia y todas las entidades referenciadas, y confirmar la transacción.

```
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);
    }
}
```

Al desechar la transacción se sincronizan todas las entidades gestionadas con la memoria caché. Cuando se confirma una transacción, automáticamente se produce un desecho. En este caso, el pedido pasa a ser una entidad gestionada. Todas las entidades a las que se hace referencia desde el pedido, cliente y el elemento de pedido también pasan a ser entidades gestionadas. Cuando la transacción se

desecha, se comprueba cada una de las entidades para determinar si se han modificado. Aquéllas que se han modificado se actualizan en la memoria caché. Una vez que se ha completado la transacción, confirmándose o retrotrayéndose, las entidades pasan a estar desconectadas y todos los cambios que se realizan en las entidades no se reflejan en la memoria caché.

Guía de aprendizaje del gestor de entidades: actualización y eliminación de entradas con un índice

Puede utilizar un índice para buscar, actualizar y eliminar entidades.

Procedimiento

Actualice y elimine entidades utilizando un índice. Utilice un índice para buscar, actualizar y eliminar entidades. En los siguientes ejemplos, la clase de entidad Order se actualiza para utilizar la anotación @Index. La anotación @Index señala WebSphere eXtreme Scale para crear un índice de rango para un atributo. El nombre del índice es el mismo nombre que el nombre del atributo y siempre es un tipo de índice 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;  }
```

En el siguiente ejemplo se muestra cómo cancelar todos los pedidos que se someten en el último minuto. Busque el pedido utilizando un índice, vuelva a añadir los artículos del pedido al inventario y elimine del sistema el pedido y los elementos de línea asociados.

```
public static void cancelOrdersUsingIndex(Session s)
  throws ObjectGridException {
    // Cancelar todos los pedidos que se sometieron hace un minuto
    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();
   // Buscar el pedido para eliminarlo.
   Order curOrder = (Order) em.find(Order.class, orderKey);
   // Verificar que el pedido no haya sido actualizado por otro usuario.
   if(curOrder != null && curOrder.date.getTime() >= cancelTime.getTime()) {
    for(OrderLine line : curOrder.lines) {
     // Vuelva a añadir el elemento al inventario.
     line.item.quantityOnHand += line.quantity;
     line.quantity = 0;
    }
   em.remove(curOrder);
   }
  }
 em.getTransaction().commit();
}
```

Guía de aprendizaje del gestor de entidades: actualización y eliminación de entradas utilizando una consulta

Puede actualizar y eliminar entidades utilizando una consulta.

Procedimiento

Actualice y elimine entradas utilizando una consulta.

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 clase de entidad de pedido es la misma que en el ejemplo anterior. La clase sigue proporcionando la anotación @Index porque la serie de consulta utiliza la fecha para buscar la entidad. El motor de consultas utiliza índices cuando pueden utilizarse.

```
public static void cancelOrdersUsingQuery(Session s) {
        // Cancelar todos los pedidos que se sometieron hace un minuto
        java.util.Date cancelTime =
     new java.util.Date(System.currentTimeMillis() - 60000);
        EntityManager em = s.getEntityManager();
        em.getTransaction().begin();

        // Crear una consulta que buscará el pedido en base a la fecha. Como
        // tenemos un índice definido en la fecha del pedido, la consulta
    // lo utilizará automáticamente.
        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();
     // Verificar que otro usuario no haya actualizado el pedido.
     // Dado que la consulta utilizó un índice, no había ningún bloqueo en la fila.
     if(order != null && order.date.getTime() >= cancelTime.getTime()) {
       for(OrderLine line : order.lines) {
        // Vuelva a añadir el elemento al inventario.
        line.item.quantityOnHand += line.quantity;
        line.quantity = 0;
       }
       em.remove(order);
     }
   }
  em.getTransaction().commit();
}
```

Como en el ejemplo anterior, el método cancelOrdersUsingQuery intenta cancelar todos los pedidos que se sometieron en el último minuto. Para cancelar el pedido, busque el pedido utilizando una consulta, vuelva a añadir los elementos al inventario y elimine el pedido y los elementos de línea asociados del sistema.

Guía de aprendizaje ObjectQuery

Con los siguientes pasos, puede desarrollar una ObjectGrid en memoria local que puede almacenar información de pedidos para un sitio web y demostrar cómo utilizar ObjectQuery para consultar los datos de la cuadrícula.

Antes de empezar

Asegúrese de que el archivo `objectgrid.jar` está en la classpath.

Acerca de esta tarea

Cada paso de la guía de aprendizaje se basa en el paso anterior. Siga cada uno de los pasos para crear una aplicación sencilla de Java Platform, Standard Edition versión 1.4 (o posterior) que utiliza un ObjectGrid local y en memoria.

Procedimiento

1. "Guía de aprendizaje de ObjectQuery - Paso 1"
   - Cómo crear un ObjectGrid local
   - Cómo definir un esquema para un único objeto utilizando el acceso a campos
   - Cómo almacenar el objeto
   - Cómo consultar el objeto con ObjectQuery
2. "Guía de aprendizaje de ObjectQuery - Paso 2" en la página 199
   - Cómo crear un índice que la consulta pueda utilizar
3. "Guía de aprendizaje de ObjectQuery - Paso 3" en la página 199
   - Cómo crear un esquema con dos entidades relacionadas
   - Cómo almacenar los objetos con una referencia de clave foránea entre ellos
   - Cómo consultar los objetos utilizando una única consulta con un JOIN
4. "Guía de aprendizaje de ObjectQuery - Paso 4" en la página 201
   - Cómo crear un esquema con varias entidades relacionadas
   - Cómo utilizar el acceso de método o propiedad, en lugar del acceso a campo

Guía de aprendizaje de ObjectQuery - Paso 1

Con los siguientes pasos, podrá seguir desarrollando un ObjectGrid local en memoria que almacena la información de pedidos para una tienda al detalle en línea mediante las API ObjectMap. Defina un esquema para la correlación y ejecute una consulta en la correlación.

Procedimiento

1. Cree un ObjectGrid con un esquema de correlación.

   Cree un ObjectGrid con un esquema de correlación para la correlación y luego inserte un objeto en la memoria y más adelante recupérelo utilizando una consulta simple.

   OrderBean.java

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

2. Defina la clave primaria.

   El código anterior muestra un objeto OrderBean. Este objeto implementa la interfaz java.io.Serializable porque todos los objetos de la memoria caché se deben poder serializar (de forma predeterminada).

El atributo orderNumber es la clave primaria del objeto. El siguiente programa de ejemplo se puede ejecutar en una modalidad autónoma. Debe seguir esta guía de aprendizaje en un proyecto Eclipse Java que tenga el archivo `objectgrid.jar` añadido a la classpath.

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");

        // Definir el esquema
        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();
    }
}
```

Esta aplicación eXtreme Scale primero inicializa un ObjectGrid local con un nombre generado automáticamente. A continuación, la aplicación crea BackingMap y QueryConfig que define qué tipo Java se asocia a la correlación, el nombre del campo que es la clave primaria de la correlación y cómo acceder a los datos del objeto. A continuación, puede obtener una sesión para obtener la instancia de ObjectMap e insertar un objeto OrderBean en la correlación en una transacción.

Después de que se confirmen los datos en la memoria caché, puede utilizar ObjectQuery para encontrar el OrderBean utilizando uno cualquiera de los campos persistentes de la clase. Los campos persistentes son aquellos que no tienen el modificador transient. Puesto que no ha definido ningún índice en BackingMap, ObjectQuery debe explorar cada objeto de la correlación utilizando el reflejo Java.

Qué hacer a continuación

"Guía de aprendizaje de ObjectQuery - Paso 2" en la página 199 demuestra cómo se puede utilizar un índice para optimizar la consulta.

Guía de aprendizaje de ObjectQuery - Paso 2

Con los siguientes pasos, puede seguir creando una ObjectGrid con una correlación y un índice, junto con un esquema para la correlación. A continuación, puede insertar un objeto en la memoria caché y, posteriormente, recuperarlo mediante una simple consulta.

Antes de empezar

Asegúrese de que ha completado "Guía de aprendizaje de ObjectQuery - Paso 1" en la página 197 antes de continuar con este paso de la guía de aprendizaje.

Procedimiento

Esquema e índice

Application.java

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

El índice debe ser una instancia com.ibm.websphere.objectgrid.plugins.index.HashIndex con los siguientes valores:

- El Name es arbitrario, pero debe ser exclusivo para una BackingMap dad.
- El AttributeName es el nombre del campo o propiedad de bean que utilice el motor para realizar una introspección de la clase. En este caso, es el nombre del campo para el que ha creado un índice.
- RangeIndex debe ser siempre true.
- FieldAccessAttribute debe coincidir con el valor establecido en el objeto QueryMapping cuando se creó el esquema de consulta. En este caso, se accede al objeto Java utilizando los campos directamente.

Cuando se ejecuta una consulta que aplica un filtro en el campo itemName, el motor de consulta utiliza automáticamente el índice definido. El uso del índice permite que la consulta se ejecute mucho más rápido y no es necesario una exploración de la correlación. El siguiente paso demuestra cómo se puede utilizar un índice para optimizar la consulta.
Paso siguiente

Guía de aprendizaje de ObjectQuery - Paso 3

Con el paso siguiente, puede crear un ObjectGrid con dos correlaciones y un esquema para las correlaciones con una relación y, más adelante, insertar objetos en la memoria caché y, posteriormente, recuperarlos utilizando una consulta simple.

Antes de empezar

Asegúrese de haber completado el apartado "Guía de aprendizaje de ObjectQuery - Paso 2" antes de llevar a cabo este paso.

Acerca de esta tarea

En este ejemplo, hay dos correlaciones, cada una con un único tipo Java correlacionado. La correlación Orden tiene objetos OrderBean y la correlación Customer tiene objetos CustomerBean.

Procedimiento

Defina las correlaciones con una relación.

OrderBean.java

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

El OrderBean ya no tiene customerName. En su lugar, tiene el customerId, que es la clave primaria para el objeto CustomerBean y la correlación 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 relación entre los tipos o correlaciones es la siguiente:

Application.java

```
public class Application
{

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

        // Definir el esquema
        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();
    }
}
```

El XML equivalente en el descriptor de despliegue de ObjectGrid es el siguiente:

```
<?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>
```

Qué hacer a continuación

"Guía de aprendizaje de ObjectQuery - Paso 4", amplía el paso actual incluyendo los objetos de acceso de propiedad y campo y las relaciones adicionales.

Guía de aprendizaje de ObjectQuery - Paso 4

El siguiente paso muestra cómo crear un ObjectGrid con cuatro correlaciones y un esquema para las correlaciones con varias relaciones unidireccionales y bidireccionales. Puede insertar objetos en la memoria caché y, posteriormente, recuperarlos mediante varias consultas.

Antes de empezar

Asegúrese de haber completado el apartado "Guía de aprendizaje de ObjectQuery - Paso 3" en la página 199 antes de continuar con el paso actual.

Procedimiento

Varias relaciones de correlaciones

OrderBean.java

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

Como en el paso anterior, OrderBean ya no tiene customerName. En su lugar, tiene el customerId, que es la clave primaria para el objeto CustomerBean y la correlación Customer.

CustomerBean.java

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

Después de crear las clases especificadas más arriba, puede ejecutar la aplicación siguiente.

Application.java

```
public class Application
{

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

        // Definir el esquema
        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();
    }
}
```

El uso de la configuración XML siguiente (en el descriptor de despliegue de ObjectGrid) es equivalente al enfoque programático anterior.

```
<?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>
```

Guía de aprendizaje de seguridad Java SE : visión general

Con la siguiente guía de aprendizaje, puede crear un entorno distribuido de eXtreme Scale en un entorno de Java Platform, Standard Edition.

Antes de empezar

Asegúrese de que está familiarizado con los conceptos básicos de una configuración de eXtreme Scale distribuido.

Acerca de esta tarea

En esta guía de aprendizaje, el servidor de catálogo, el servidor de contenedor y el cliente se ejecutan todos en un entorno Java SE. Cada paso de la guía de aprendizaje se basa en el anterior. Siga cada uno de los pasos para proteger un eXtreme Scale distribuido y desarrollar una aplicación Java SE sencilla para acceder al eXtreme Scale seguro.

Inicio de la guía de aprendizaje

Procedimiento

1. "Guía de aprendizaje de seguridad Java SE - Paso 1"
   - Iniciar un servidor de catálogo no seguro
   - Iniciar un servidor de contenedor no seguro
   - Iniciar un cliente para acceder a los datos
   - Utilizar xsadmin para mostrar el tamaño de la correlación
   - Detener el servidor
2. "Guía de aprendizaje de seguridad de Java SE - Paso 2" en la página 208
   - Uso del generador de credenciales
   - Uso del autenticador
   - Iniciar un servidor de catálogo seguro
   - Iniciar un servidor de contenedor seguro
   - Iniciar el cliente para acceder a ObjectGrid seguro
   - Utilizar xsadmin para mostrar el tamaño de la correlación
   - Detener el servidor seguro
3. "Guía de aprendizaje de seguridad de Java SE - Paso 3" en la página 214
   - Uso de la política de autorización JAAS
4. "Guía de aprendizaje de seguridad de Java SE - Paso 4" en la página 218
   - Crear un almacén de claves y un almacén de confianza
   - Configurar propiedades SSL para el servidor
   - Configurar propiedades SSL para el cliente
   - Utilizar xsadmin para mostrar el tamaño de la correlación
   - Detener el servidor seguro

Guía de aprendizaje de seguridad Java SE - Paso 1

En este tema se describe un *ejemplo no seguro simple*. En los pasos de la guía de aprendizaje se añaden características de seguridad adicionales para aumentar la cantidad de seguridad integrada que está disponible.

Antes de empezar

Nota: Todos los archivos necesarios para este paso de la guía de aprendizaje se proporcionan en la siguiente sección.

Procedimiento

Ejecución del ejemplo
Inicie el servicio de catálogo utilizando los siguientes scripts. Si desea más información sobre cómo iniciar el servicio de catálogo, consulte la información sobre cómo iniciar el servicio de catálogo en *Guía de administración*.

1. Vaya al directorio bin: `cd objectgridRoot/bin`
2. Inicie el servidor de catálogo denominado catalogServer:
   - UNIX Linux `startOgServer.sh catalogServer`
   - Windows `startOgServer.bat catalogServer`
3. Vaya hasta el directorio bin `cd objectgridRoot/bin`
4. A continuación inicie un servidor de contenedor llamado c0 con el siguiente script:

- 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`

Ejemplo

Si desea más información sobre cómo iniciar servidores de contenedor, consulte la información sobre cómo iniciar los procesos de contenedor en *Guía de administración*.

Después de iniciar el servidor de catálogo y el servidor de contenedor, inicie el cliente tal como se muestra a continuación:

1. Vaya hasta el directorio bin una vez más.
2. `java -classpath ../lib/objectgrid.jar;../applib/secsample.jar com.ibm.websphere.objectgrid.security.sample.guide.SimpleApp`

El archivo `secsample.jar` contiene la clase SimpleApp.

La salida de este programa es:

```
El nombre de cliente para el ID 0001 es fName lName
```

También puede utilizar xsadmin para mostrar los tamaños de correlación de la cuadrícula "accounting" (contabilidad).

- Vaya hasta el directorio `objectgridRoot/bin`.
- Utilice el mandato `xsadmin` con la opción -mapSizes del modo siguiente.
  - UNIX Linux `xsadmin.sh -g accounting -m mapSet1 -mapSizes`
  - Windows `xsadmin.bat -g accounting -m mapSet1 -mapSizes`

  Verá la siguiente salida.

```
Este programa de utilidad administrativo se proporciona sólo como un
ejemplo y no se considera como un componente completamente soportado del
producto WebSphere eXtreme Scale.
Conexión al servicio de catálogo en localhost:1099
********** Visualización de resultados para la cuadrícula - accounting,
MapSet - mapSet1 **********
*** Listado de correlaciones para c0 ***
Nombre de correlación: customer Núm. de partición: 0 Tamaño de
correlación: 1 Tipo de fragmento: primario
Total de servidores: 1
Recuento total de dominios: 1
```

Cómo detener los servidores

Servidor de contenedor

Utilice el siguiente mandato para detener el servidor de contenedor c0.

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

Windows `stopOgServer.bat c0 -catalogServiceEndPoints localhost:2809`

Verá el siguiente mensaje.

```
CWOBJ2512I: el servidor ObjectGrid c0 se ha detenido.
```

Servidor de catálogo

Puede detener un servidor de catálogo utilizando el siguiente mandato.

UNIX Linux `stopOgServer.sh catalogServer -catalogServiceEndPoints localhost:2809`

Windows `stopOgServer.bat catalogServer -catalogServiceEndPoints localhost:2809`

Si concluye el servidor de catálogo, verá el siguiente mensaje.

```
CWOBJ2512I: el servidor ObjectGrid catalogServer se ha detenido.
```

Archivos necesarios

El archivo siguiente es la clase Java para SimpleApp.

SimpleApp.java

```
// Este programa de ejemplo se proporciona TAL CUAL y se puede usar, ejecutar, copiar y modificar
// sin que el cliente tenga que pagar derechos
// (a) para su propia formación,
// (b) para desarrollar aplicaciones diseñadas para ejecutarse con un producto IBM WebSphere,
// para uso interno del cliente o para su redistribución por su parte, integrado en una
// aplicación de ese tipo, en los productos propios del cliente.
// Material bajo licencia - Propiedad de 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);
    }

    /**
     * leer y grabar la correlación
     * @throws excepción
     */
    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);
    }

    /**
     * Obtener ObjectGrid
     * @return una instancia de ObjectGrid
     * @throws excepción
     */
    protected ObjectGrid getObjectGrid(String[] args) throws Exception {
        ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();

        // Crear ObjectGrid
        ClientClusterContext ccContext = ogManager.connect("localhost:2809", null, null);
        ObjectGrid og = ogManager.getObjectGrid(ccContext, "accounting");

        return og;

    }

}
```

El método getObjectGrid de esta clase obtiene un ObjectGrid, y el método run lee un registro de la correlación del cliente y actualiza el valor.

Para ejecutar este ejemplo en un entorno distribuido, se crean un archivo XML descriptor de ObjectGrid `SimpleApp.xml` y un archivo XML de despliegue `SimpleDP.xml`. Los archivos se muestran en el siguiente ejemplo:

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>
```

El siguiente archivo XML configura el entorno de despliegue.

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>
```

Se trata de una configuración de ObjectGrid sencilla con una instancia de ObjectGrid llamada "accounting" y una correlación llamada "customer" (dentro del mapSet "mapSet1"). El archivo `SimpleDP.xml` incorpora un conjunto de correlaciones que se configura con 1 partición y 0 réplicas mínimas necesarias.

Paso siguiente de la guía de aprendizaje

Guía de aprendizaje de seguridad de Java SE - Paso 2

Basándose en el paso anterior, el siguiente tema muestra cómo implementar la autenticación de cliente en un entorno distribuido de eXtreme Scale.

Antes de empezar

Asegúrese de que ha completado "Guía de aprendizaje de seguridad Java SE - Paso 1" en la página 204.

Acerca de esta tarea

Con la autenticación de cliente habilitada, un cliente se autentica antes de conectarse al servidor eXtreme Scale. Esta sección muestra cómo puede realizarse la autenticación de cliente en un entorno de servidor de eXtreme Scale, e incluye código de ejemplo y scripts para demostrarlo.

Al igual que cualquier otro mecanismos de autenticación, la autenticación mínima consta de los siguientes pasos:

1. El administrador efectúa cambios en las configuraciones de modo que la autenticación sea un requisito.
2. El cliente proporciona una credencial al servidor.
3. El servidor autentica la credencial en el registro.

Procedimiento

1. **Credencial del cliente**

   Una credencial de cliente se representa mediante una interfaz com.ibm.websphere.objectgrid.security.plugins.Credential. Una credencial de cliente puede ser un par de nombre de usuario y contraseña, un ticket Kerberos, un certificado de cliente o datos en cualquier formato que hayan acordado el cliente y el servidor. Consulte la la documentación de la API Credential para ver más detalles.

   Esta interfaz define de forma explícita los métodos equals(Object) y hashCode(). Estos dos métodos son importantes porque los objetos Subject autenticados se almacenan en memoria caché utilizando el objeto Credential como la clave en el lado del servidor.

   eXtreme Scale también proporciona un plug-in para generar una credencial. Este plug-in se representa mediante la interfaz com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator, y se utiliza para generar una credencial de cliente. Esto resulta útil cuando la credencial puede caducar. En este caso, se llama al método getCredential() para renovar una credencial. Consulte la documentación de la API CredentialGenerator para obtener más detalles.

   Puede implementar estas dos interfaces para que el tiempo de ejecución del cliente de eXtreme Scale obtenga credenciales de cliente.

   Este ejemplo utiliza las dos siguientes implementaciones de plug-in de ejemplo proporcionadas por eXtreme Scale.

   ```
   com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential
   com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator
   ```

   Si desea más información sobre estos plug-ins, consulte el tema sobre la programación de autenticación de cliente en la *Guía de programación*.
2. **Autenticación de servidor** Después de que el cliente de eXtreme Scale recupere el objeto Credential mediante el objeto CredentialGenerator, el objeto Credential de este cliente se envía junto con la solicitud del cliente al servidor de eXtreme

Scale. El servidor de eXtreme Scale autentica el objeto Credential antes de procesar la solicitud. Si el objeto Credential se autentica correctamente, se devuelve un objeto Subject para representar este cliente.

A continuación, el objeto Subject se almacena en memoria caché y caduca después de que su vida útil alcance el valor de tiempo de espera de la sesión. El valor de tiempo de espera del inicio de sesión puede establecerse mediante la propiedad loginSessionExpirationTime del archivo XML del clúster. Por ejemplo, establecer `loginSessionExpirationTime="300"` hace que el objeto Subject caduque en 300 segundos.Este objeto Subject se utilizará para autorizar la solicitud que se muestra más adelante.

Un servidor de eXtreme Scale utiliza el plug-in Authenticator para autenticar el objeto Credential. Consulte la documentación de la API de Authenticator para obtener más detalles.

Este ejemplo utiliza una implementación incorporada de eXtreme Scale: KeyStoreLoginAuthenticator, que es para fines de prueba y ejemplo (un almacén de claves es un registro de usuarios simple y no debe utilizarse en un entorno de producción). programación de autenticación de cliente en la *Guía de programación*.

Este KeyStoreLoginAuthenticator utiliza un KeyStoreLoginModule para autenticar el usuario con el almacén de claves utilizando el módulo de inicio de sesión JAAS "KeyStoreLogin". El almacén de claves se puede configurar como una opción para la clase KeyStoreLoginModule. En el siguiente ejemplo se muestra el alias keyStoreLogin configurado en el archivo de configuración de JAAS og_jaas.config:

```
KeyStoreLogin{
com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginModule required
     keyStoreFile="../security/sampleKS.jks" debug = true;
};
```

Los siguientes mandatos crean un almacén de claves sampleKS.jks en el directorio %OBJECTGRID_HOME%/security con la contraseña sampleKS1. Además, se crean tres certificados de usuario que representan el usuario administrator, el usuario manager y el usuario cashier con sus propias contraseñas.

a. Vaya hasta el directorio raíz de eXtreme Scale.

```
cd objectgridRoot
```

b. Cree un directorio llamado "security".

```
mkdir security
```

c. Vaya hasta el directorio de seguridad acabado de crear.

```
cd security
```

d. Utilice keytool (en el directorio `javaHOME/bin`) para crear un usuario "administator" con la contraseña "administrator1" en el almacén de claves sampleKS.jks.

```
keytool -genkey -v -keystore ./sampleKS.jks -storepass sampleKS1
-alias administrator -keypass administrator1
-dname CN=administrator,O=acme,OU=OGSample -validity 10000
```

e. Utilice keytool (en el directorio `javaHOME/bin`) para crear un usuario "manager" con la contraseña "manager1" en el almacén de claves sampleKS.jks.

```
keytool -genkey -v -keystore ./sampleKS.jks -storepass sampleKS1
-alias manager -keypass manager1
-dname CN=manager,O=acme,OU=OGSample -validity 10000
```

f. Utilice keytool (en el directorio `javaHOME/bin`) para crear un usuario "cashier" con la contraseña "cashier1" en el almacén de claves sampleKS.jks.

```
keytool -genkey -v -keystore ./sampleKS.jks -storepass sampleKS1
-alias cashier -keypass cashier1 -dname CN=cashier,O=acme,OU=OGSample
-validity 10000
```

La configuración de seguridad de cliente se configura en el archivo de propiedades del cliente. Utilice el siguiente mandato para crear una copia en el directorio %OBJECTGRID_HOME%/security:

a. Vaya al directorio de seguridad.

```
cd objectgridRoot/security
```

b. Copie el archivo sampleClient.properties en el archivo client.properties.

```
cp ../properties/sampleClient.properties client.properties
```

Las siguientes propiedades aparecen resaltadas en el archivo client.properties en el directorio de seguridad.

a. **securityEnabled:** establecer securityEnabled en true (valor predeterminado) habilita la seguridad de cliente, que incluye la autenticación.

b. **credentialAuthentication:** establezca credentialAuthentication en Supported (valor predeterminado), que significa que el cliente da soporte a la autenticación de credenciales.

c. **transportType:** establezca transportType en TCP/IP, que significa que no se utilizará SSL.

d. **singleSignOnEnabled:** establézcalo en false (valor predeterminado). El inicio de sesión único no está disponible.

3. **Configuración de seguridad de servidor**

La configuración de seguridad de servidor se especifica en el archivo XML de descriptor de seguridad y en el archivo de propiedades de seguridad del servidor.El archivo XML de descriptor de seguridad describe las propiedades de seguridad comunes a todos los servidores (incluidos los servidores de catálogo y los servidores de contenedor). Un ejemplo de propiedad es la configuración de autenticador que representa el mecanismo de autenticación y el registro de usuarios.

A continuación se muestra el archivo security.xml que se va a utilizar en este ejemplo:

```
<?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:** establézcalo en true, que habilita la seguridad de servidor que incluye la autenticación.

b. **loginSessionExpirationTime:** establezca el valor en 300 (valor predeterminado).

c. **authenticator:** añada la clase de autenticador KeyStoreLoginAuthenticator al archivo XML del clúster tal como se muestra a continuación:

```
<authenticator className ="com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginAuthenticator">
        </authenticator>
```

d. **credentialAuthentication:** establezca el atributo credentialAuthentication en Required de forma que el servidor requiera la autenticación

Si desea una explicación más detallada sobre el archivo `security.xml`, consulte la información sobre el archivo XML de descriptor de la seguridad en *Guía de administración*.

Copie el archivo de propiedad de servidor en el directorio de seguridad. En este momento no es necesario modificar nada en este archivo.

a. Vaya hasta el directorio de seguridad.

```
cd objectgridRoot/security
```

b. Copie el archivo de ejemplo de objectGrid `sampleServer.properties` del directorio de propiedades en el nuevo archivo `server.properties`.

```
cp ../properties/containerServer.properties server.properties
```

Realice los cambios siguiente en el archivo `server.properties`:

a. **securityEnabled:** establezca el atributo **securityEnabled** en true.

b. **transportType:** establezca el atributo **transportType** en TCP/IP, que significa que no se utiliza SSL.

c. **secureTokenManagerType:** establezca el atributo **secureTokenManagerType** en none para no configurar el gestor de señales seguro.

4. **Cliente seguro** Conecte la aplicación cliente al servidor de forma segura tal como se muestra en el siguiente ejemplo:

```
// Este programa de ejemplo se proporciona TAL CUAL y se puede utilizar, ejecutar, copiar y modificar
// sin que el cliente tenga que pagar derechos
// (a) para su propia formación,
// (b) para desarrollar aplicaciones diseñadas para ejecutarse con un producto IBM WebSphere,
// para uso interno del cliente o para su redistribución por su parte, integrado en una
// aplicación de ese tipo, en los productos propios del cliente.
// Material bajo licencia - Propiedad de 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);
    }

    /**
     * Obtener ObjectGrid
     * @return una instancia de ObjectGrid
     * @throws excepción
     */
    protected ObjectGrid getObjectGrid(String[] args) throws Exception {
        ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
        ogManager.setTraceFileName("logs/client.log");
        ogManager.setTraceSpecification("ObjectGrid*=all=enabled:ORBRas=all=enabled");

        // Crear un objeto ClientSecurityConfiguration utilizando el archivo especificado
        ClientSecurityConfiguration clientSC = ClientSecurityConfigurationFactory
                .getClientSecurityConfiguration(args[0]);

        // Crear un CredentialGenerator utilizando el usuario y la contraseña pasados.
        CredentialGenerator credGen = new UserPasswordCredentialGenerator(args[1], args[2]);
        clientSC.setCredentialGenerator(credGen);

        // Crear un ObjectGrid conectándose al servidor de catálogo.
        ClientClusterContext ccContext = ogManager.connect("localhost:2809", clientSC, null);
        ObjectGrid og = ogManager.getObjectGrid(ccContext, "accounting");

        return og;
```

```
    }
}
```

Hay tres cosas distintas de la aplicación no segura:

a. Se ha creado un objeto ClientSecurityConfiguration pasando el archivo `client.properties` configurado.
b. Se ha creado un UserPasswordCredentialGenerator utilizando el ID de usuario y la contraseña pasados.
c. Se ha conectado al servidor de catálogo para obtener un ObjectGrid del ClientClusterContext pasado un objeto ClientSecurityConfiguration.

5. **Emita la aplicación**

   Para ejecutar la aplicación, inicie el servidor de catálogo. Emita las opciones de la línea de mandatos -clusterFile y -serverProps para pasar las propiedades de seguridad:

   a. Vaya al directorio bin:

      ```
      cd objectgridRoot/bin
      ```

   b. Inicie el servidor de catálogo:

      - 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"
        ```

   A continuación, inicie un servidor de contenedor seguro utilizando el siguiente script:

   a. Vuelva a ir hasta el directorio bin:

      ```
      cd objectgridRoot/bin
      ```

   b. Inicie un servidor de contenedor seguro:

      - 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"
        ```

   El archivo de propiedades de servidor se pasa emitiendo -serverProps.

   Después de iniciar el servidor, inicie el cliente utilizando el siguiente mandato:

   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
      ```

   El archivo `secsample.jar` contiene la clase SimpleApp.

   SecureSimpleApp utiliza tres parámetros que se proporcionan en la siguiente lista:

   a. El archivo `../security/client.properties` es el archivo de propiedades de seguridad del cliente.
   b. `manager` es el ID de usuario.

c. manager1 es la contraseña.

Después de emitir la clase, se obtiene la siguiente salida:

El nombre de cliente para ID 0001 es fName lName.

También puede utilizar xsadmin para mostrar los tamaños de correlación de la cuadrícula "accounting" (contabilidad).

- Vaya hasta el directorio objectgridRoot/bin.
- Utilice el mandato xsadmin con la opción -mapSizes del modo siguiente.
  - 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

  Obtendrá la siguiente salida.

  ```
  Este programa de utilidad administrativo se proporciona sólo como un
  ejemplo y no se considera como un componente completamente soportado
  del producto WebSphere eXtreme Scale.

  Conexión al servicio de catálogo en localhost:1099

  *********** Visualización de resultados para la cuadrícula -
  accounting, MapSet - mapSet1 ***********

  *** Listado de correlaciones para c0 ***

  Nombre de correlación: customer Núm. de partición: 0 Tamaño de
  correlación: 1 Tipo de fragmento: primario

  Total de servidores: 1

  Recuento total de dominios: 1
  ```

Ahora, puede utilizar el mandato stopOgServer para detener el proceso del servidor de contenedor o de servicio de catálogo. Sin embargo tendrá que proporcionar un archivo de configuración de seguridad. El archivo de propiedades de cliente de ejemplo define las siguientes dos propiedades para generar una credencial ID usuario/contraseña (manager/manager1).

```
credentialGeneratorClass=com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator
credentialGeneratorProps=manager manager1
```

Detenga el contenedor c0 con el siguiente mandato.

- UNIX Linux stopOgServer.sh c0 -catalogServiceEndPoints localhost:2809 -clientSecurityFile ..\security\client.properties
- Windows stopOgServer.bat c0 -catalogServiceEndPoints localhost:2809 -clientSecurityFile ..\security\client.properties

Si no proporciona la opción -clientSecurityFile, verá una excepción con el mensaje siguiente.

```
>> SERVER (id=39132c79, host=9.10.86.47) TRACE START:

>> org.omg.CORBA.NO_PERMISSION: el servidor requiere la autenticación de
credenciales, pero no hay contexto de seguridad del cliente.
Normalmente, esto sucede cuando el cliente no pasar ninguna credencial
al servidor.

vmcid: 0x0

código menor: 0

completado: No
```

También puede concluir el servidor de catálogo utilizando el mandato siguiente. Sin embargo, si desea continuar intentando el siguiente paso de la guía de aprendizaje, podrá dejar el servidor de catálogo ejecutándose.

- UNIX Linux `stopOgServer.sh catalogServer -catalogServiceEndPoints localhost:2809 -clientSecurityFile ..\security\client.properties`
- Windows `stopOgServer.bat catalogServer -catalogServiceEndPoints localhost:2809 -clientSecurityFile ..\security\client.properties`

Si concluye el servidor de catálogo, verá la siguiente salida.

```
CWOBJ2512I: el servidor ObjectGrid catalogServer se ha detenido
```

Ahora el sistema ya es parcialmente seguro y se ha llevado a cabo habilitando la autenticación. Ha configurado el servidor para conectarse en el registro de usuarios, ha configurado el cliente para proporcionar credenciales de cliente y ha cambiado el archivo de propiedades de cliente y el archivo XML del clúster para habilitar la autenticación.

Si proporciona una contraseña no válida, verá una excepción indicando que el nombre de usuario o la contraseña no son correctos.

Si desea más detalles sobre la autenticación de cliente, consulte la información sobre la autenticación del cliente de aplicaciones en *Guía de administración*.

Paso siguiente de la guía de aprendizaje

Guía de aprendizaje de seguridad de Java SE - Paso 3

Tras autenticar un cliente, como en el paso anterior, puede proporcionar privilegios de seguridad a través de mecanismos de autorización de eXtreme Scale.

Antes de empezar

Asegúrese de haber completado el apartado "Guía de aprendizaje de seguridad de Java SE - Paso 2" en la página 208 antes de llevar a cabo esta tarea.

Acerca de esta tarea

El paso anterior de esta guía de aprendizaje ha demostrado cómo habilitar la autenticación en una cuadrícula de eXtreme Scale. Como resultado, un cliente no autenticado se puede conectar al servidor y enviar solicitudes al sistema. No obstante, cada cliente autenticado tiene el mismo permiso o privilegios que el servidor, como por ejemplo, la lectura, la grabación o la supresión de datos que se almacenan en las correlaciones de ObjectGrid. Los clientes también pueden emitir cualquier tipo de consulta. Esta sección demuestra cómo utilizar la autorización de eXtreme Scale para otorgar distintos privilegios variables de usuarios autenticados.

De forma parecida a muchos otros sistemas, eXtreme Scale adopta un mecanismo de autorización basado en permisos. WebSphere eXtreme Scale tiene distintas categorías de permisos representadas por diferentes clases de permisos. Este tema muestra MapPermission. Para obtener una categoría completa de permisos, consulte la la referencia de autorización de cliente en la *Guía de programación*..

En WebSphere eXtreme Scale, la clase com.ibm.websphere.objectgrid.security.MapPermission representa permisos para los recursos de eXtreme Scale, en particular los métodos de las interfaces ObjectMap o JavaMap. WebSphere eXtreme Scale define las siguientes series de permiso para acceder a los métodos de ObjectMap y JavaMap:

- leer: otorga permiso para leer los datos de la correlación.
- grabar: otorga permiso para actualizar los datos de la correlación.
- insertar: otorga permiso para insertar los datos en la correlación.

- eliminar: otorga permiso para eliminar los datos de la correlación.
- invalidar: otorga permiso para invalidar los datos de la correlación.
- todos: otorga todos los permisos anteriores: leer, grabar, insertar, eliminar e invalidar.

La autorización tiene lugar cuando un cliente llama a un método de ObjectMap o JavaMap. El tiempo de ejecución de eXtreme Scale comprueba los distintos permisos de correlación para los métodos diferentes. Si los permisos requeridos no se conceden al cliente, se produce una excepción AccessControlException.

Esta guía de aprendizaje muestra cómo utilizar la autorización Java Authentication and Authorization Service (JAAS) para otorgar accesos a correlaciones de autorizaciones para usuarios distintos.

Procedimiento

1. **Habilitación de la autorización de eXtreme Scale** Para habilitar la autorización en ObjectGrid, debe establecer `true` como valor del atributo securityEnabled para ese ObjectGrid determinado en el archivo XML. La habilitación de la seguridad en el ObjectGrid significa que se habilita la autorización. Utilice los siguientes mandatos para crear un nuevo archivo XML de ObjectGrid con la seguridad habilitada.
   a. Vaya al directorio `bin`.
      ```
      cd objectgridRoot/bin
      ```
   b. Copie el archivo `SimpleApp.xml` en el archivo `SecureSimpleApp.xml`.
      ```
      cp SimpleApp.xml SecureSimpleApp.xml
      ```
   c. Abra el archivo `SecureSimpleApp.xml` y añada `securityEnabled="true"` en el nivel de ObjectGrid tal como se muestra en el XML siguiente:
      ```
      <?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. **Definición de la política de autorización.** En la sección de autenticación previa al cliente, ha creado tres usuarios en el almacén de claves: cashier, manager y administrator. En este ejemplo, el usuario "cashier" sólo tiene permisos de lectura para todas las correlaciones y que el usuario "manager" tiene todos los permisos. La autorización JAAS se utiliza en este ejemplo. La autorización JAAS utiliza el archivo de política de autorización para otorgar permisos a principales. El siguiente archivo se define en el directorio de seguridad:
   ```
   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";
   };
   ```

   Nota:

   - El código base `codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"` es un URL especialmente reservado para ObjectGrid. Todos los permisos de ObjectGrid otorgados a principales deben utilizar esta base de código especial.

- La primera sentencia grant otorga permiso de correlación de "lectura" al principal `"CN=cashier,O=acme,OU=OGSample"`, de modo que el usuario cashier sólo tiene permiso de lectura de correlación para todas las correlaciones en el ObjectGrid accounting.
- La segunda sentencia grant otorga "todos" los permisos de correlación al principal `"CN=manager,O=acme,OU=OGSample"`, de modo que el usuario manager tiene todos los permisos para las correlaciones en el ObjectGrid accounting.

Ahora puede iniciar un servidor con una política de autorización. El archivo de política de autorización de JAAS se puede establecer utilizando la propiedad-D estándar: -Djava.security.auth.policy=../security/ogAuth.policy

3. **Ejecute la aplicación.**

   Después de crear los archivos anteriores, puede ejecutar la aplicación.

   Utilice los siguientes mandatos para iniciar el servidor de catálogo. Si desea más información sobre cómo iniciar el servicio de catálogo, consulte la información sobre cómo iniciar un servicio de catálogo en *Guía de administración*.

   a. Vaya al directorio bin: `cd objectgridRoot/bin`

   b. Inicie el servidor de catálogo.

      - 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"
      ```

      Los archivos `security.xml` y `server.properties` se crearon en el paso anterior de esta guía de aprendizaje.

      T

   c. Entonces puede iniciar un servidor de contenedor seguro utilizando el script siguiente. Ejecute el script siguiente desde el directorio 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"
      ```

   Tenga en cuenta las siguientes diferencias del mandato de inicio de servidor de contenedor anterior:

   - Utilice el archivo `SecureSimpleApp.xml` en lugar del archivo `SimpleApp.xml`.
   - Añada otro argumento `-Djava.security.auth.policy` para establecer el archivo de política de autorización de JAAS para el proceso de servidor de contenedor.

Utilice el mismo mandato que en el paso anterior de la guía de aprendizaje:

a. Desplácese al directorio bin.

b. `java -classpath ../lib/objectgrid.jar;../applib/secsample.jar com.ibm.websphere.objectgrid.security.sample.guide.SecureSimpleApp ../security/client.properties manager manager1`

   Como el usuario "manager" tiene todos los permisos para las correlaciones del accounting ObjectGrid, la aplicación se ejecuta correctamente.

   Ahora, en lugar de utilizar el usuario "manager", utilice el usuario "cashier" para iniciar la aplicación cliente.

c. Desplácese al directorio bin.

d. `java -classpath ../lib/objectgrid.jar;../applib/secsample.jar com.ibm.ws.objectgrid.security.sample.guide.SecureSimpleApp ../security/client.properties cashier cashier1`

Se genera la siguiente excepción:

```
Excepción en la hebra "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 más
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 más
```

Esta excepción se produce porque el usuario "cashier" no tiene permiso de grabación, y por ello no puede actualizar el cliente de correlación.

Ahora el sistema da soporte a la autorización. Puede definir políticas de autorización para otorgar distintos permisos a usuarios diferentes. Si desea más información sobre la autorización, consulte la información sobre la autorización del cliente de aplicaciones en *Guía de programación*.

Qué hacer a continuación

Complete el siguiente paso de la guía de aprendizaje. Consulte "Guía de aprendizaje de seguridad de Java SE - Paso 4".

Guía de aprendizaje de seguridad de Java SE - Paso 4

El siguiente paso le explica cómo habilitar una capa de seguridad para la comunicación entre los puntos finales del entorno.

Antes de empezar

Asegúrese de haber completado el apartado "Guía de aprendizaje de seguridad de Java SE - Paso 3" en la página 214 antes de llevar a cabo esta tarea.

Acerca de esta tarea

La topología de eXtreme Scale da soporte a Transport Layer Security/Secure Sockets Layer (TLS/SSL) para la comunicación segura entre puntos finales de ObjectGrid (cliente, servidores de contenedor y servidores de catálogo). Este paso de la guía de aprendizaje se basa en los pasos anteriores para habilitar la seguridad de transporte.

Procedimiento

1. **Cree almacenes de claves y claves de TLS/SSL**

   Para habilitar la seguridad de transporte, debe crear un almacén de claves y un almacén de confianza. Este ejercicio sólo crea un par de almacén de claves y almacén de confianza. Estos almacenes se utilizan para los servidores de catálogo, servidores de contenedor y clientes ObjectGrid, y se crean con la herramienta de claves de JDK.

   - *Crear una clave privada en el almacén de claves*

     ```
     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
     ```

     Con este mandato, se crea un almacén de claves key.jks con una clave "ogsample" almacenada en él. Este almacén de claves key.jks se utilizará como el almacén de claves SSL.

   - *Exportar el certificado público*

     ```
     keytool -export -alias ogsample -keystore key.jks -file temp.key
     -storepass ogpass
     ```

     Con este mandato, se extrae el certificado público de la clave "ogsample" y se almacena en el archivo temp.key.

   - *Importar el certificado público del cliente en el almacén de confianza*

     ```
     keytool -import -noprompt -alias ogsamplepublic -keystore trust.jks
     -file temp.key -storepass ogpass
     ```

     Con este mandato, el certificado público se ha añadido al almacén de claves trust.jks. Este trust.jks se utiliza como el almacén de confianza SSL.

2. **Configuración de los archivos de propiedades de ObjectGrid**

   En este paso, debe configurar los archivos de propiedades de ObjectGrid para habilitar la seguridad de transporte.

   Primero, copie los archivos key.jks y trust.jks en el directorio objectgridRoot/security.

   Se establecen las siguientes propiedades en el archivo client.properties y 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: el valor de transportType se establece en "SSL-Required", que significa que el transporte requiere SSL. Por lo tanto, todos los puntos finales de ObjectGrid (clientes, servidores de catálogo y servidores de contenedor deben tener establecida la configuración SSL y toda la comunicación de transporte estará cifrada.

   Las otras propiedades se utilizan para establecer las configuraciones SSL. Consulte la información sobre la seguridad de la capa de transporte y la capa de sockets seguros en *Guía de administración* si desea una explicación detallada. Asegúrese de que sigue las instrucciones de este tema para actualizar el archivo orb.properties.

   Asegúrese de que sigue esta página para actualizar el archivo `orb.properties`.

   En el archivo `server.properties` , debe añadir una propiedad adicional clientAuthentication y establecerla en false (falso). En el lado del servidor, no es necesario que confíe en el cliente.

   ```
   clientAuthentication=false
   ```

3. **Ejecute la aplicación**

   Los mandatos son los mismos que en el tema “Guía de aprendizaje de seguridad de Java SE - Paso 3” en la página 214.

   Utilice los siguientes mandatos para iniciar un servidor de catálogo.

   a. Vaya al directorio bin: `cd objectgridRoot/bin`

   b. Inicie el servidor de catálogo:

      - 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"
        ```

      Los archivos `security.xml` y `server.properties` se crearon en la página “Guía de aprendizaje de seguridad de Java SE - Paso 2” en la página 208.

      Utilice la opción -JMXServicePort para especificar explícitamente el puerto JMX para el servidor. Es opción es necesaria para utilizar el mandato xsadmin.

      Ejecute un servidor de contenedor de ObjectGrid seguro:

   c. Vuelva al directorio 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"
```

Tenga en cuenta las siguientes diferencias del mandato de inicio de servidor de contenedor anterior:

- Utilice SecureSimpleApp.xml en lugar de SimpleApp.xml
- Añada otro -Djava.security.auth.policy para establecer el archivo de política de autorización de JAAS para el proceso de servidor de contenedor.

Ejecute el siguiente mandato para la autenticación de cliente:

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
```

Como el usuario "manager" tiene permiso para todas las correlaciones del accounting ObjectGrid, la aplicación se ejecuta satisfactoriamente.

También puede utilizar xsadmin para mostrar los tamaños de correlación de la cuadrícula "accounting" (contabilidad).

- Vaya hasta el directorio `objectgridRoot/bin`.
- Utilice el mandato `xsadmin` con la opción -mapSizes del modo siguiente.
  - 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
    ```

  Tenga en cuenta que se especifica el puerto JMX del servicio de catálogo utilizando aquí -p 11001.

  Obtendrá la siguiente salida.

```
Este programa de utilidad administrativo se proporciona sólo como un ejemplo y no se debe
considerar como un componente completamente soportado del producto WebSphere eXtreme Scale.
Conexión al servicio de catálogo en localhost:1099
********** Visualización de resultados para la cuadrícula - accounting, MapSet - mapSet1 **********
*** Listado de correlaciones para c0 ***
Nombre de correlación: customer Núm. de partición: 0 Tamaño de correlación: 1 Tipo de fragmento: primario
Total de servidores: 1
Recuento total de dominios: 1
```

Ejecución de la aplicación con un almacén de claves incorrecto

Si el almacén de confianza no contiene el certificado público de la clave privada en el almacén de claves, obtendrá una excepción que indica que no se puede confiar en la clave.

Para mostrarlo, cree otro almacén de claves 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
```

Después, modifique server.properties de forma que keyStore señale a este nuevo almacén de claves key2.jks:

```
keyStore=../security/key2.jks
```

Ejecute el siguiente mandato para iniciar el servidor de catálogo:

a. Desplácese al directorio bin: `cd objectgridRoot/bin`

b. Inicie el servidor de catálogo:

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"
```

Verá la siguiente excepción:

```
Caused by: com.ibm.websphere.objectgrid.ObjectGridRPCException:
    com.ibm.websphere.objectgrid.ObjectGridRuntimeException:
        SSL connection fails and plain socket cannot be used.
```

Finalmente, vuelva a cambiar el archivo `server.properties` para utilizar el archivo `key.jks`.

Ejemplo de servicios de datos REST y guía de aprendizaje

En este tema se describe cómo empezar a utilizar rápidamente el servicio de datos REST de WebSphere eXtreme Scale. Se incluyen instrucciones de WebSphere Application Server versión 7.0, WebSphere Application Server Community Edition y de Apache Tomcat.

Acerca de esta tarea

El ejemplo incluido tiene código fuente y binarios compilados para ejecutar una cuadrícula de eXtreme Scale particionada. Este ejemplo demuestra cómo crear una cuadrícula simple y modelar los datos utilizando entidades de eXtreme Scale y proporciona dos aplicaciones cliente de línea de mandatos que permiten añadir y consultar entidades utilizando Java o C# (véase la Figura 1).

El cliente Java de ejemplo utiliza la API EntityManager Java de eXtreme Scale para consultar datos en la cuadrícula. Este cliente se puede ejecutar en Eclipse o utilizando un script de línea de mandatos. Tenga en cuenta que el cliente Java de ejemplo no ofrece una demostración del servicio de datos REST, pero permite actualizar datos en la cuadrícula para que un navegador de web u otros clientes puedan leer los datos. El cliente Java de ejemplo y el navegador web, tal como se muestra en la Figura 1, ilustran clientes HTTP que utilizan en servicio de datos REST y clientes Java de eXtreme Scale utilizando la misma cuadrícula de eXtreme Scale y los datos que ésta contiene.

El cliente C# Microsoft WCF Data Services de ejemplo se comunica con la cuadrícula de eXtreme Scale por medio del servicio de datos REST utilizando la infraestructura .NET. El cliente de WCF Data Services se puede utilizar tanto para

actualizar la cuadrícula como para consultarla.

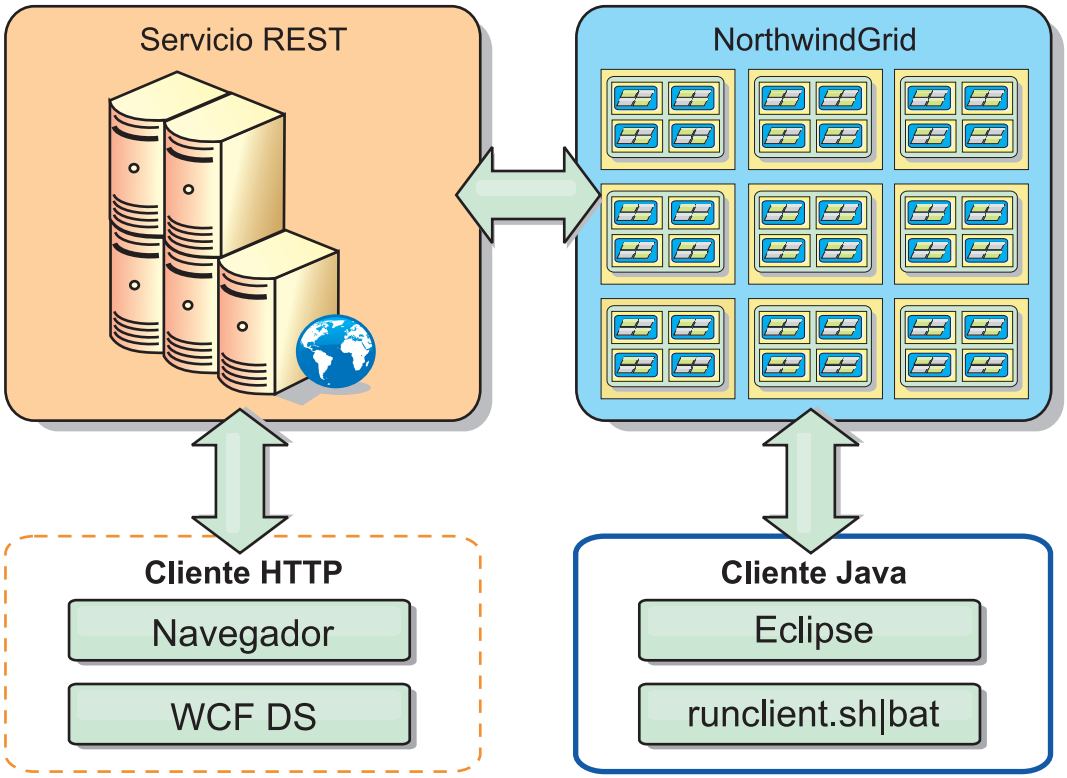


Figura 44. Iniciación a la topología de ejemplo

Procedimiento

1. Configure e inicie la cuadrícula de eXtreme Scale. Consulte "Habilitación del servicio de datos REST" en la página 224.
2. Configure e inicie el servicio de datos REST en un servidor web. Consulte "Configuración de servidores de aplicaciones para el servicio de datos REST" en la página 232.
3. Ejecute un cliente para que interactúe con el servicio de datos REST. Hay dos opciones disponibles:
   a. Ejecute el cliente Java de ejemplo para llenar la cuadrícula de datos utilizando la API EntityManager y consultar los datos de la cuadrícula utilizando un navegador web y el servicio de datos REST de eXtreme Scale. Consulte "Uso de un cliente Java con los servicios de datos REST" en la página 242.
   b. Ejecute el cliente C# de WCF Data Services. Consulte "Cliente Visual Studio 2008 WCF con servicio de datos REST" en la página 243.

Convenciones de directorio

Este tema describe muchos ejemplos y sintaxis de línea de mandatos que debe hacer referencia a directorios especiales como, por ejemplo, *raíz_instal_wxs* e *inicio_wxs*. Estos directorios se definen así:

raíz_intal_wxs
El directorio *raíz_instal_wxs* es el directorio raíz donde se instalan los archivos

del producto WebSphere eXtreme Scale. Puede ser el directorio en el que se extrae el archivo zip de evaluación o el directorio en el que se instala el producto eXtreme Scale.

- Ejemplo al extraer la prueba:
  `/opt/IBM/WebSphere/eXtremeScale`
- Ejemplo cuando eXtreme Scale se instala en un directorio autónomo:
  `/opt/IBM/eXtremeScale`
- Ejemplo cuando eXtreme Scale se integra con WebSphere Application Server:
  `/opt/IBM/WebSphere/AppServer`

inicio_wxs
El directorio *inicio_wxs* es el directorio raíz de las bibliotecas, ejemplos y componentes del producto WebSphere eXtreme Scale. Es el mismo que el directorio *raíz_instal_wxs* cuando se extrae el producto de evaluación. Para las instalaciones autónomas, es el subdirectorio de ObjectGrid en el directorio *raíz_instal_wxs*. Para las instalaciones integradas con WebSphere Application Server, este directorio es el directorio `optionalLibraries/ObjectGrid` en *raíz_instal_wxs*.

- Ejemplo al extraer la prueba:
  `/opt/IBM/WebSphere/eXtremeScale`
- Ejemplo cuando eXtreme Scale se instala en un directorio autónomo:
  `/opt/IBM/eXtremeScale/ObjectGrid`
- Ejemplo cuando eXtreme Scale se integra con WebSphere Application Server:
  `/opt/IBM/WebSphere/AppServer/optionalLibraries/ObjectGrid`

raíz_was
El directorio *raíz_was* es el directorio raíz de una instalación de WebSphere Application Server:

`/opt/IBM/WebSphere/AppServer`

inicio_servicioRest
El directorio *inicio_servicioRest* es el directorio en el que se encuentran las bibliotecas y los ejemplos del servicio de datos REST de eXtreme Scale. Este directorio se denomina *restservice* y es un subdirectorio del directorio *inicio_wxs*.

- Ejemplo para despliegues autónomos:
  `/opt/IBM/WebSphere/eXtremeScale/ObjectGrid/restservice`
- Ejemplo para despliegues integrados de WebSphere Application Server:
  `/opt/IBM/WebSphere/AppServer/optionalLibraries/ObjectGrid/restservice`

raíz_tomcat
raíz_tomcat es el directorio raíz de la instalación de Apache Tomcat.

`/opt/tomcat5.5`

raíz_wasce
raíz_wasce es el directorio raíz de la instalación de WebSphere Application Server Community Edition.

`/opt/IBM/WebSphere/AppServerCE`

inicio_java
inicio_java es el directorio raíz de una instalación de Java Runtime Environment (JRE).

`/opt/IBM/WebSphere/eXtremeScale/java`

Habilitación del servicio de datos REST

El servicio de datos REST representa metadatos de entidades de WebSphere eXtreme Scale para representar cada entidad como un EntitySet.

Inicio de una cuadrícula eXtreme Scale de ejemplo

En general, antes de iniciar el servicio de datos REST, debe iniciar la cuadrícula eXtreme Scale. Los pasos siguientes iniciarán un solo proceso de servicio de catálogo de eXtreme Scale y dos procesos servidor de contenedor.

WebSphere eXtreme Scale se puede instalar utilizando tres métodos diferentes:

- Instalación de prueba
- Despliegue autónomo
- Despliegue integrado de WebSphere Application Server

Modelo de datos escalable de eXtreme Scale

El ejemplo Northwind de Microsoft utiliza la tabla Order Detail para establecer una asociación de muchos a muchos entre Orders y Products.

Las especificaciones de correlación de objeto a relacional (ORM) como ADO.NET Entity Framework y Java Persistence API (JPA) puede correlacionar las tablas y las relaciones utilizando entidades. No obstante, esta arquitectura no se escala. Todo debe encontrarse en la misma máquina, o en un clúster de máquinas caro, para que funcione bien.

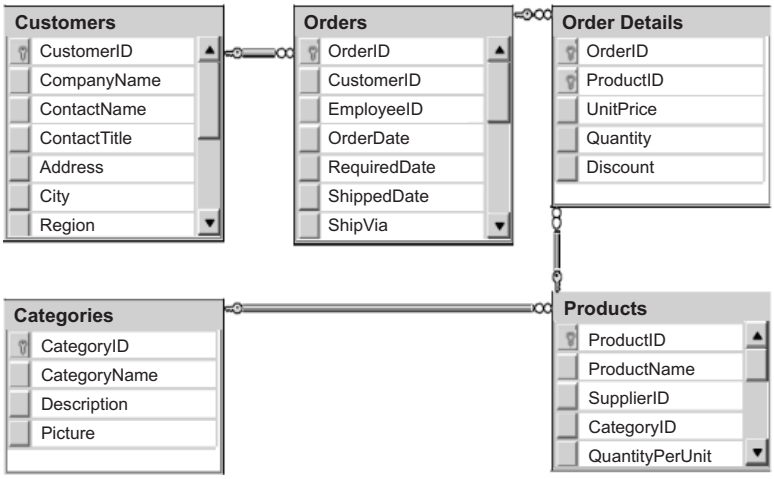


Figura 45. Diagrama del esquema de ejemplo Northwind de Microsoft SQL Server

A fin de crear una versión escalable del ejemplo, las entidades se deben modelar de modo que cada entidad o grupo de entidades relacionadas se pueda particionar basándose en una sola clave. Mediante la creación de particiones en una sola clave, las solicitudes se pueden distribuir entre varios servidores independientes. Para conseguir esta configuración, las entidades se han dividido en dos árboles: el árbol de clientes y pedidos (Customer and Order) y el árbol de productos y categorías (Product and Category). En este modelo, cada árbol se puede dividir independientemente y, por lo tanto, puede crecer a ritmos diferentes, con lo que aumenta la escalabilidad.

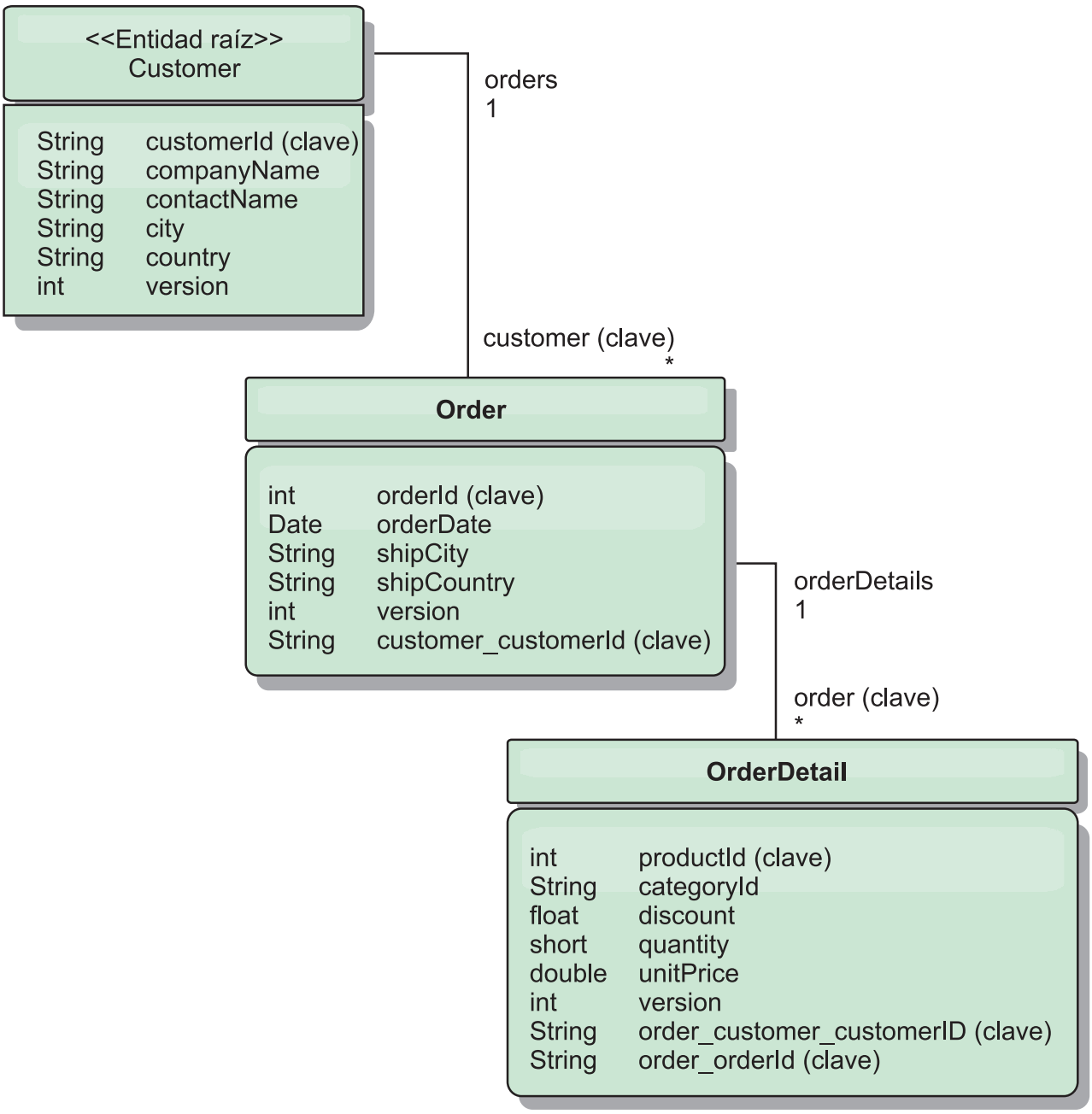


Figura 46. Diagrama del esquema de entidades Customer y Order

Por ejemplo, tanto Order como Product tienen enteros exclusivos distintos como claves. De hecho, las tablas Order y Product son realmente independientes entre sí. Por ejemplo, considere el efecto del tamaño de un catálogo (el número de productos que vende) con el número total de pedidos. Intuitivamente, podría parecer que tener muchos productos también implica tener muchos pedidos, pero esto no es necesariamente así. Si esto fuera verdad, podría aumentar fácilmente las ventas con el simple hecho de añadir más productos a su catálogo. Los pedidos y los productos tienen sus propias tablas independientes. Puede ampliar aún más este concepto de modo que los pedidos y los productos tengan sus propias cuadrículas de datos distintas. Con cuadrículas independientes, puede controlar el número de particiones y servidores, además del tamaño de cada cuadrícula por separado de modo que la aplicación se pueda escalar. Si dobla el tamaño de su

catálogo, debe doblar la cuadrícula de productos, pero la cuadrícula de pedidos puede permanecer inalterada. Y lo contrario es aplicable a una oleada de pedidos o a una oleada de pedidos prevista.

En el esquema, un cliente tiene cero o más pedidos y un pedido tiene elementos de línea (OrderDetail), cada uno con un producto específico. Un producto se identifica mediante el ID (la clave de producto) en cada OrderDetail. Una sola cuadrícula almacena clientes, pedidos y detalles del pedido con Customer como entidad raíz de la cuadrícula. Puede recuperar clientes por ID, pero deberá obtener los pedidos a partir del ID de cliente. Por lo tanto, el ID de cliente se añade al pedido como parte de su clave. Del mismo modo, el ID de cliente y el ID de pedido forman parte del ID de detalle del pedido (OrderDetail).

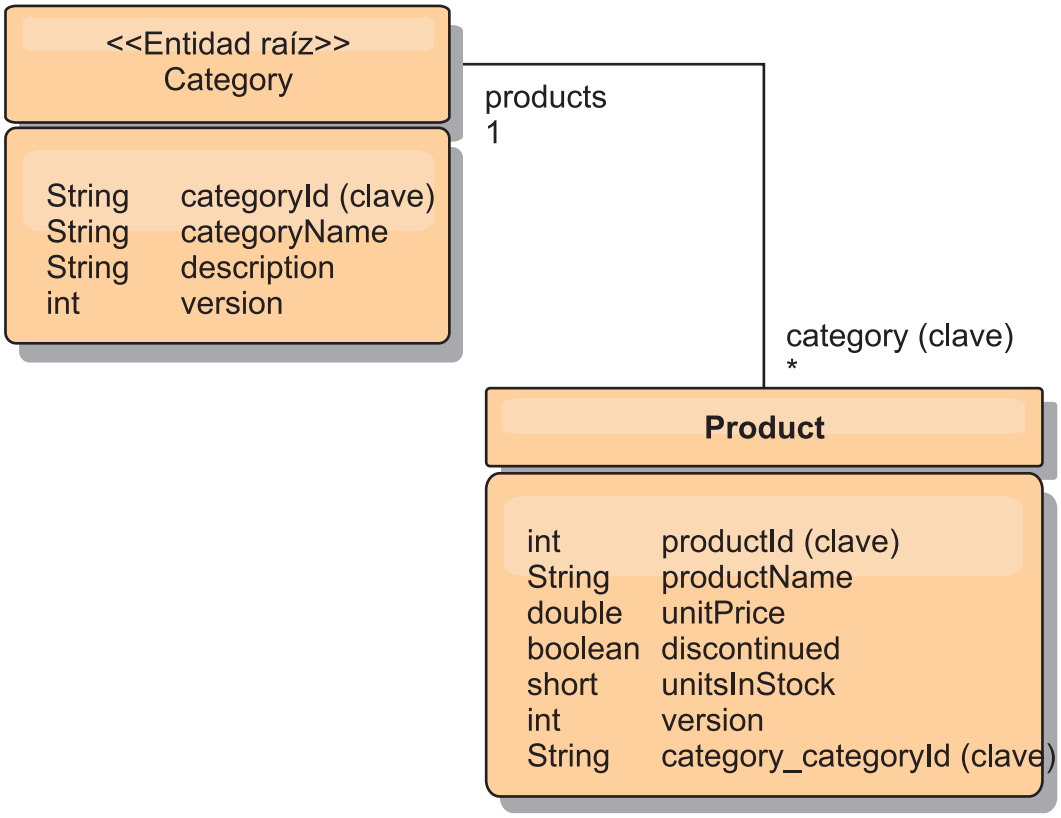


Figura 47. Diagrama del esquema de entidades Category y Product

En el esquema de categorías y productos, la categoría es la raíz del esquema. Con este esquema, los clientes pueden consultar productos por categoría. Consulte "Recuperación y actualización de datos con REST" para obtener detalles adicionales sobre las asociaciones de claves y su importancia.

Recuperación y actualización de datos con REST

El protocolo OData requiere que todas las entidades puedan ser dirigidas por su forma canónica. Esto significa que cada entidad debe incluir la clave de la entidad raíz particionada (la raíz de esquema).

A continuación se ofrece un ejemplo de cómo utilizar la asociación desde una entidad raíz para dirigirse a un hijo en :

```
/Customer('ACME')/order(100)
```

En WCF Data Services, se debe poder dirigir directamente a la entidad hijo, lo que significa que la clave de la raíz del esquemas debe formar parte de la clave del hijo: /Order(customer_customerId='ACME', orderId=100).Esto se consigue creando una asociación con la entidad raíz donde la asociación de uno a uno o de muchos a uno con la entidad raíz también se identifica como una clave. Cuando las entidades se incluyen como parte de la clave, los atributos de la entidad padre se exponen como propiedades clave.

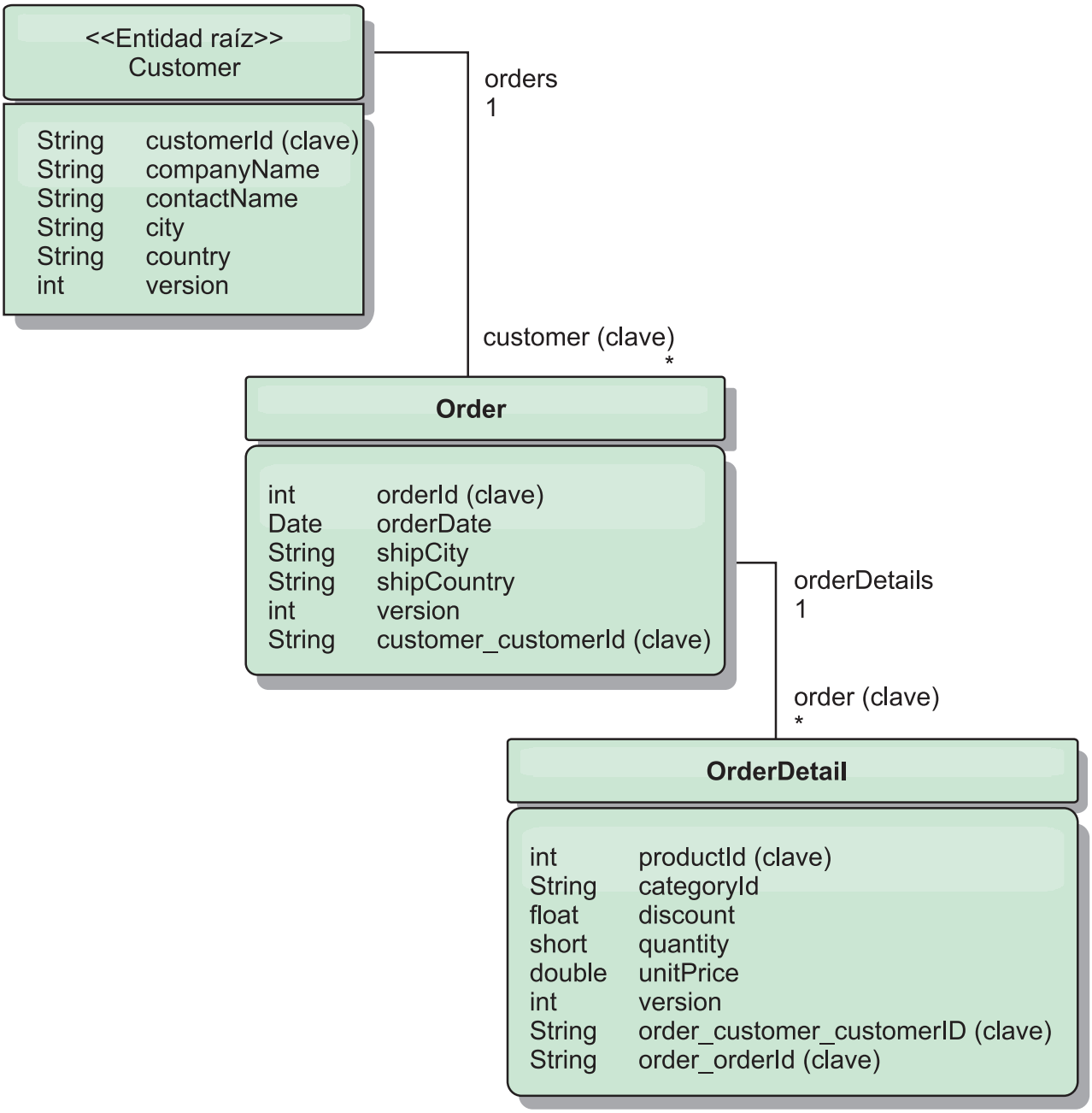


Figura 48. Diagrama del esquema de entidades Customer y Order

El diagrama de esquema de entidad Customer/Order ilustra cómo se particiona cada entidad utilizando Customer. La entidad Order incluye Customer como parte de su clave y, por lo tanto, se puede acceder a ella directamente. El servicio de

datos REST expone todas las asociaciones de clave como propiedades individuales: Order tiene customer_customerId y OrderDetail tiene order_customer_customerId y order_orderId.

Utilizando la API EntityManager, puede encontrar el pedido (Order) utilizando el cliente (Customer) y el ID de cliente:

```
transaction.begin();
// Buscar el pedido mediante el cliente. Sólo incluimos el ID
// en la clase Customer al crear la instancia de clave OrderId.
Order order = (Order) em.find(Order.class,
    new OrderId(100, new Customer('ACME')));
...
transaction.commit();
```

Al utilizar el servicio de datos REST, el pedido se puede recuperar con cualquiera de los URL:

- `/Order(orderId=100, customer_customerId='ACME')`
- `/Customer('ACME')/orders?$filter=orderId eq 100`

Para dirigirse a la clave de cliente se utiliza el nombre de atributo de la entidad Customer, un carácter de subrayado y el nombre de atributo del ID de cliente: customer_customerId.

Una entidad también puede incluir una entidad no raíz como componente de su clave si todos los ancestros de la entidad no raíz tienen asociaciones de clave con la raíz. En este ejemplo, OrderDetail tiene una asociación de clave con Order y Order tiene una asociación de clave con la entidad Customer raíz. Utilizando la API EntityManager:

```
transaction.begin();
// Construir una instancia de clave OrderDetailId. Ésta incluye
// Order y Customer con sólo el conjunto de claves.
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);
...
```

El servicio de datos REST permite dirigirse a OrderDetail directamente:

```
/OrderDetail(productId=500, order_customer_customerId='ACME', order_orderId =100)
```

La asociación de la entidad OrderDetail a la entidad Product se ha roto para permitir particionar el inventario de pedidos (Order) y productos (Product) independientemente. La entidad OrderDetail almacena la categoría y el ID de producto en lugar de una relación estricta. Al desacoplar los dos esquemas de entidad, sólo se accede a una partición a la vez.

El esquema de entidad Category y Product, ilustrado en el diagrama, muestra que la entidad raíz es Category y que cada entidad Product tiene una asociación con una entidad Category. La entidad Category se incluye en la identidad Product. El servicio de datos REST expone una propiedad clave: category_categoryId que permite dirigirse directamente a Product.

Puesto que Category es la entidad raíz, en un entorno particionado, se debe conocer la categoría para encontrar el producto. Utilizando la API EntityManager, la transacción se debe fijar a la entidad Category antes de buscar la entidad Product.

Utilizando la API EntityManager:

```
transaction.begin();
// Crear la entidad raíz Category con sólo la clave. Esto nos
// permite construir un ProductId sin necesidad de encontrar
// la entidad Category en primer lugar. La transacción queda
// fijada a la partición donde se almacena Category "COMP".
Category cat = new Category("COMP");
Product product = (Product) em.find(Product.class,
    new ProductId(500, cat));
...
```

El servicio de datos REST permite dirigirse a Product directamente:

```
/Product(productId=500, category_categoryId='COMP')
```

Inicio de una cuadrícula autónoma para servicios de datos REST

Siga los pasos que se indican a continuación para iniciar la cuadrícula de ejemplo de servicio REST de WebSphere eXtreme Scale para un despliegue autónomo de eXtreme Scale.

Antes de empezar

Instale la versión de evaluación de WebSphere eXtreme Scale o el producto completo:

- Instale la versión autónoma del producto WebSphere eXtreme Scale 7.1 y aplique los arreglos subsiguientes.
- Descargue y extraiga la aplicación WebSphere eXtreme Scale Versión 7.1 de evaluación, que incluye el servicio de datos REST de WebSphere eXtreme Scale.

Acerca de esta tarea

Inicie la cuadrícula de ejemplo de WebSphere eXtreme Scale.

Procedimiento

1. Inicie el proceso del servicio de catálogo. Abra una línea de mandatos o una ventana de terminal y defina la variable de entorno JAVA_HOME:
   - Linux UNIX `export JAVA_HOME=inicio_java`
   - Windows `set JAVA_HOME=inicio_java`
2. `cd restservice_home/gettingstarted`
3. Inicie el proceso del servicio de catálogo. Para iniciar el servicio *sin* la seguridad de eXtreme Scale, utilice los mandatos siguientes.
   - Linux UNIX `./runcat.sh`
   - Windows `runcat.bat`

   Para iniciar el servicio con la seguridad de eXtreme Scale, utilice los mandatos siguientes.
   - Linux UNIX `./runcat_secure.sh`
   - Windows `runcat_secure.bat`
4. Inicie dos procesos de servidor de contenedores. Abra otra línea de mandatos o una ventana de terminal y defina la variable de entorno JAVA_HOME:
   - Linux UNIX `export JAVA_HOME=inicio_java`
   - Windows `set JAVA_HOME=inicio_java`
5. `cd restservice_home/gettingstarted`

6. Inicie un proceso de servidor de contenedor:

   Para iniciar el servidor sin la seguridad de eXtreme Scale, utilice los mandatos siguientes:

   - Linux UNIX `./runcontainer.sh container0`
   - Windows `runcontainer.bat container0`

   Para iniciar el servidor con la seguridad de eXtreme Scale, utilice los mandatos siguientes.

   - Linux UNIX `./runcontainer_secure.sh container0`
   - Windows `runcontainer_secure.bat container0`
7. Abra otra línea de mandatos o una ventana de terminal y defina la variable de entorno JAVA_HOME:
   - Linux UNIX `export JAVA_HOME=inicio_java`
   - Windows `set JAVA_HOME=inicio_java`
8. `cd restservice_home/gettingstarted`
9. Inicie un segundo proceso de servidor de contenedor.

   Para iniciar el servidor sin la seguridad de eXtreme Scale, utilice los mandatos siguientes.

   - Linux UNIX `./runcontainer.sh container1`
   - Windows `runcontainer.bat container1`

   Para iniciar el servidor con la seguridad de eXtreme Scale, utilice los mandatos siguientes.

   - Linux UNIX `./runcontainer_secure.sh container1`
   - Windows `runcontainer_secure.bat container1`

Resultados

Espere hasta que los contenedores de eXtreme Scale estén preparados antes de continuar con los siguientes pasos. Los servidores de contenedor están preparados cuando aparece el mensaje siguiente en la ventana terminal:

`CWOBJ1001I: ObjectGrid Server nombre_contenedor está listo para procesar peticiones.`

Donde *nombre_contenedor* es el nombre del contenedor que se ha iniciado.

Inicio de una cuadrícula para los servicios de datos REST en WebSphere Application Server

Siga estos pasos para iniciar una cuadrícula de datos de ejemplo de servicio REST de WebSphere eXtreme Scale autónomo para un despliegue de WebSphere eXtreme Scale integrado con WebSphere Application Server. Aunque WebSphere eXtreme Scale está integrado con WebSphere Application Server, estos pasos inician un contenedor y un proceso de servicio de catálogo de WebSphere eXtreme Scale.

Antes de empezar

Instale el producto WebSphere eXtreme Scale Versión 7.1 en un directorio de instalación de WebSphere Application Server Versión 7.0.0.5 o posterior (con la seguridad inhabilitada), aumente al menos un perfil de servidor de aplicaciones.

Acerca de esta tarea

Inicie la cuadrícula de ejemplo WebSphere eXtreme Scale.

Procedimiento

1. Inicie el proceso del servicio de catálogo. Abra una línea de mandatos o una ventana de terminal y defina la variable de entorno JAVA_HOME:
   - Linux UNIX `export JAVA_HOME=inicio_java`
   - Windows `set JAVA_HOME=inicio_java`

   `cd restservice_home/gettingstarted`
2. Inicie el proceso del servicio de catálogo.

   Para iniciar el servidor sin la seguridad de eXtreme Scale, utilice los mandatos siguientes.
   - Linux UNIX `./runcat.sh`
   - Windows `runcat.bat`

   Para iniciar el servidor con la seguridad de eXtreme Scale, utilice los mandatos siguientes.
   - Linux UNIX `./runcat_secure.sh`
   - Windows `runcat_secure.bat`
3. Inicie dos procesos de servidor de contenedores. Abra otra línea de mandatos o una ventana de terminal y defina la variable de entorno JAVA_HOME:
   - Linux UNIX `export JAVA_HOME=inicio_java`
   - Windows `set JAVA_HOME=inicio_java`
4. Inicie un proceso de servidor de contenedor.

   Para iniciar el servidor sin la seguridad de eXtreme Scale, utilice los mandatos siguientes.
   a. Abra una ventana de línea de mandatos.
   b. `cd restservice_home/gettingstarted`
   c. Para iniciar el servidor *sin* la seguridad de eXtreme Scale, utilice los mandatos siguientes.
      - Linux UNIX `./runcontainer.sh container0`
      - Windows `runcontainer.bat container0`
   d. Para iniciar el servidor con la seguridad de eXtreme Scale, utilice los mandatos siguientes.
      - Linux UNIX `./runcontainer_secure.sh container0`
      - Windows `runcontainer_secure.bat container0`
5. Inicie un segundo proceso de servidor de contenedor.
   a. Abra una ventana de línea de mandatos.
   b. `cd restservice_home/gettingstarted`
   c. Para iniciar el servidor *sin* la seguridad de eXtreme Scale, utilice los mandatos siguientes.
      - Linux UNIX `./runcontainer.sh container1`
      - Windows `runcontainer.bat container1`

d. Para iniciar el servidor *con* la seguridad de eXtreme Scale, utilice los mandatos siguientes.
   - Linux UNIX `./runcontainer_secure.sh container1`
   - Windows `runcontainer_secure.bat container1`

Resultados

Espere hasta que los servidores de contenedor estén preparados antes de continuar con los siguientes pasos. Los servidores de contenedor están preparados cuando aparece el mensaje siguiente:

`CWOBJ1001I: ObjectGrid Server nombre_contenedor está listo para procesar peticiones.`

Donde *nombre_contenedor* es el nombre del contenedor que se inició en el paso anterior.

Configuración de servidores de aplicaciones para el servicio de datos REST

Inicio de los servicios de datos REST en WebSphere Application Server Versión 7.0

Este tema describe cómo configurar e iniciar el servicio de datos REST de eXtreme Scale utilizando WebSphere Application Server Versión 7.0.

Antes de empezar

Verifique que la cuadrícula de eXtreme Scale de ejemplo se haya iniciado. Consulte el apartado "Habilitación del servicio de datos REST" en la página 224 para obtener información detallada sobre cómo iniciar la cuadrícula.

Procedimiento

1. Descargue e instale WebSphere Application Server for Developers Versión 7.0.

   Restricción: No habilite la seguridad.
2. Descargue e instale WebSphere Application Server Versión 7.0 Fixpack 5 o posterior.
3. Añada el archivo JAR de tiempo de ejecución del cliente de WebSphere eXtreme Scale, el archivo `wsogclient.jar` y el archivo JAR de configuración del servicio de datos REST o el directorio de la vía de acceso de clases del servidor de aplicaciones:
   a. Abra la consola administrativa de WebSphere Application Server.
   b. Vaya a **Entorno** → **Bibliotecas compartidas**
   c. Pulse **Nuevo**
   d. Añada las entradas siguientes a los campos:
      1) Nombre: extremescale_client_v71
      2) Classpath: `inicio_wxs/lib/wsogclient.jar`
   e. Pulse **Aceptar**
   f. Pulse **Nuevo**
   g. Añada las entradas siguientes a los campos apropiados:
      1) Nombre: `extremescale_gettingstarted_config`

2) Classpath:
   - `inicio_restservice/gettingstarted/restclient/bin`
   - `inicio_restservice/gettingstarted/common/bin`

Recuerde: Añada cada vía de acceso a una línea distinta.

h. Pulse **Aceptar**

i. Guarde los cambios en la configuración maestra

4. Instale el archivo EAR del servicio de datos REST, `wxsrestservice.ear`, en WebSphere Application Server con la consola administrativa:
   a. Abra la consola administrativa de WebSphere Application Server.
   b. Vaya a **Aplicaciones → Nueva aplicación**
   c. Vaya al archivo `inicio_restservice/lib/wxsrestservice.ear`, seleccione el archivo y pulse **Siguiente**
   d. Elija las opciones de instalación detalladas y pulse **Siguiente**
   e. En la pantalla de avisos de seguridad de la aplicación, pulse **Continuar**
   f. Elija las opciones de instalación predeterminadas y pulse **Siguiente**
   g. Seleccione un servidor con el que correlacionar la aplicación y pulse **Siguiente**
   h. En la página de recarga de JSP, utilice los valores predeterminados y pulse **Siguiente**
   i. En la página de bibliotecas compartidas, correlacione el módulo `wxsrestservice.war` con las bibliotecas compartidas definidas siguientes:
      - extremescale_client_v71
      - extremescale_ gettingstarted _config
   j. En la página de correlación de la relación de bibliotecas compartidas, utilice los valores predeterminados y pulse **Siguiente**
   k. En la página de correlación de sistemas principales virtuales, utilice los valores predeterminados y pulse **Siguiente**
   l. En la página de correlación de raíces de contexto, defina `wxsrestservice` como raíz de contexto.
   m. En la pantalla Resumen, pulse **Finalizar** para completar la instalación
   n. Guarde los cambios en la configuración maestra
5. Inicie el servidor de aplicaciones y la aplicación de servicio de datos REST de eXtreme Scale wxsrestservice. Después de que la aplicación se haya iniciado, consulte el archivo `SystemOut.log` correspondiente al servidor de aplicaciones y verifique que aparezca el mensaje siguiente: `CWOBJ4000I: Se ha iniciado el servicio de datos REST de WebSphere eXtreme Scale.`
6. Verifique que el servicio de datos REST funcione
   a. Abra un navegador web y vaya a: `http://localhost:9080/wxsrestservice/restservice/NorthwindGrid`. Aparecerá el documento de servicio correspondiente a NorthwindGrid.
   b. Vaya a: `http://localhost:9080/wxsrestservice/restservice/NorthwindGrid/$metadata`. Aparecerá el documento de extensiones de datos del modelo de entidad (EDMX).
7. Para detener los procesos de cuadrícula, utilice la combinación de teclas `CTRL+C` en la ventana de mandato respectiva.

Inicio del servicio de datos REST con WebSphere eXtreme Scale integrado en WebSphere Application Server 7.0

Este tema describe cómo configurar e iniciar el servicio de datos REST de eXtreme Scale utilizando WebSphere Application Server version 7.0 integrado y aumentado con WebSphere eXtreme Scale.

Antes de empezar

Verifique que la cuadrícula de eXtreme Scale autónoma de ejemplo se haya iniciado. Consulte el apartado "Habilitación del servicio de datos REST" en la página 224 para obtener información detallada sobre cómo iniciar la cuadrícula.

Acerca de esta tarea

Para empezar a utilizar el servicio de datos REST de WebSphere eXtreme Scale utilizando WebSphere Application Server, siga los pasos que se indican a continuación:

Procedimiento

1. Añada el JAR de configuración del ejemplo de servicio de datos REST de WebSphere eXtreme Scale a la vía de acceso de clases:
   a. Abra la consola de administración de WebSphere
   b. Vaya a Entorno -> Bibliotecas compartidas
   c. Pulse Nuevo
   d. Añada las entradas siguientes a los campos apropiados:
      1) Nombre: extremescale_gettingstarted_config
      2) Classpath
         - `inicio_restservice/gettingstarted/restclient/bin`
         - `inicio_restservice/gettingstarted/common/bin`

         Recuerde: Cada vía de acceso debe aparecer en una línea distinta.
   e. Pulse **Aceptar**
   f. Guarde los cambios en la configuración maestra
2. Instale el archivo EAR del servicio de datos REST, wxsrestservice.ear, en WebSphere Application Server utilizando la consola de administración de WebSphere:
   a. Abra la consola de administración de WebSphere
   b. Vaya a Aplicaciones -> Nueva aplicación
   c. Vaya al archivo `inicio_restservice/lib/wxsrestservice.ear` en el sistema de archivos. Seleccione el archivo y pulse **Siguiente**.
   d. Elija las opciones de instalación detalladas y pulse **Siguiente**.
   e. En la pantalla de avisos de seguridad de la aplicación, pulse **Continuar**.
   f. Elija las opciones de instalación predeterminadas y pulse **Siguiente**.
   g. Seleccione un servidor con el que correlacionar el módulo wxsrestservice.war y pulse **Siguiente**.
   h. En la página de recarga de JSP, utilice los valores predeterminados y pulse **Siguiente**.
   i. En la página de bibliotecas compartidas, correlacione el módulo "wxsrestservice.war" con las bibliotecas compartidas siguientes, definidas durante el paso 1: `extremescale_ gettingstarted _config`

j. En la página de correlación de la relación de bibliotecas compartidas, utilice los valores predeterminados y pulse **Siguiente**.

k. En la página de correlación de sistemas principales virtuales, utilice los valores predeterminados y pulse **Siguiente**.

l. En la página de correlación de raíces de contexto, defina wxsrestservice como raíz de contexto.

m. En la pantalla Resumen, pulse **Finalizar** para completar la instalación.

n. Guarde los cambios en la configuración maestra.

3. Si se ha iniciado la cuadrícula eXtreme Scale con la seguridad de eXtreme Scale habilitada, establezca la propiedad siguiente en el archivo `inicio_restservice/gettingstarted/restclient/bin/wxsRestService.properties`.

```
ogClientPropertyFile=inicio_restservice/gettingstarted/security/security.ogclient.properties
```

4. Inicie el servidor de aplicaciones y la aplicación de servicio de datos REST de eXtreme Scale "wxsrestservice".

   Después de que la aplicación se haya iniciado, consulte el archivo SystemOut.log correspondiente al servidor de aplicaciones y verifique que aparezca el mensaje siguiente: `CWOBJ4000I: Se ha iniciado el servicio de datos REST de WebSphere eXtreme Scale.`

5. Verifique que el servicio de datos REST funcione

   a. Abra un navegador y acceda a:

      http://localhost:9080/wxsrestservice/restservice/NorthwindGrid

      Aparecerá el documento de servicio correspondiente a NorthwindGrid.

   b. Vaya a:

      http://localhost:9080/wxsrestservice/restservice/NorthwindGrid/$metadata

      Aparecerá el documento de extensiones de datos del modelo de entidad (EDMX).

6. Para detener los procesos de cuadrícula, utilice CTRL+C en la ventana de mandato respectiva.

Inicio del servicio de datos REST en WebSphere Application Server Community Edition

Este tema describe cómo configurar e iniciar el servicio de datos REST de eXtreme Scale utilizando WebSphere Application Server Community Edition.

Antes de empezar

Verifique que la cuadrícula de datos de ejemplo se haya iniciado. Consulte el apartado “Habilitación del servicio de datos REST” en la página 224 para obtener información detallada sobre cómo iniciar la cuadrícula.

Procedimiento

1. Descargue e instale WebSphere Application Server Community Edition Versión 2.1.1.3 o posterior en `raíz_wasce`, como: `/opt/IBM/wasce`
2. Inicie el servidor WebSphere Application Server Community Edition ejecutando el mandato siguiente:
   - Linux UNIX `raíz_wasce/bin/startup.sh`
   - Windows `raiz_wasce/bin/startup.bat`

3. Si se ha iniciado la cuadrícula eXtreme Scale con la seguridad de eXtreme Scale habilitada, establezca las propiedades siguientes en el archivo `restservice_home/gettingstarted/restclient/bin/wxsRestService.properties`.

```
ogClientPropertyFile=inicio_restservice/gettingstarted/security/security.ogclient.properties
loginType=none
```

4. Instale el servicio de datos REST de eXtreme Scale y el ejemplo proporcionado en el servidor WebSphere Application Server Community Edition:
   a. Añada el JAR de tiempo de ejecución del cliente ObjectGrid al depósito de WebSphere Application Server Community Edition:
      1) Abra la consola de administración de WebSphere Application Server Community Edition e inicie sesión.

         Consejo: El URL predeterminado es: `http://localhost:8080/console`. El ID de usuario predeterminado es `system` y la contraseña es `manager`.
      2) Pulse el **Depósito**, en la carpeta Servicios.
      3) En la sección **Añadir archivo al depósito**, introduzca los datos siguientes en los recuadros de texto de entrada:

Tabla 16. Archivo al depósito

Recuadro de texto	Valor
Archivo	wxs_home/lib/ogclient.jar
Grupo	com.ibm.websphere.xs
Artefacto	ogclient
Versión	7.0
Tipo	jar

      4) Pulse el botón Instalar.

         Consejo: Consulte la nota técnica siguiente para obtener información detallada sobre los distintos métodos de dependencias configuración de clase y biblioteca: Specifying external dependencies to applications running on WebSphere Application Server Community Edition (Especificación de dependencias externas con aplicaciones que se ejecutan en WebSphere Application Server Community Edition).
   b. Despliegue el módulo del servicio de datos REST, que es el archivo `wxsrestservice.war`, en el servidor WebSphere Application Server Community Edition.
      1) Edite el archivo XML de despliegue de ejemplo `restservice_home/gettingstarted/wasce/geronimo-web.xml` para incluir las dependencias de vía de acceso a los directorios de vía de acceso de clases del ejemplo de iniciación:

         Cambie las vías de acceso classesDirs para los dos GBeans del cliente de iniciación:

         - La vía de acceso "classesDirs" para el GBean GettingStarted_Client_SharedLib se debe definir como: `inicio_restservice/gettingstarted/restclient/bin`
         - La vía de acceso "classesDirs" para el GBean GettingStarted_Common_SharedLib se debe definir como: `inicio_restservice/gettingstarted/common/bin`

2) Abra la consola administrativa de WebSphere Application Server Community Edition e inicie sesión.

   Consejo: El URL predeterminado es: `http://localhost:8080/console`. El ID de usuario predeterminado es `system` y la contraseña es `manager`.

3) Pulse **Desplegar nuevas**.
4) En la página **Instalar aplicaciones nuevas**, escriba los valores siguientes en los recuadros de texto:

Tabla 17. Valores de instalación

Recuadro de texto	Valor
Archivo	`inicio_restservice/lib/wxsrestservice.war`
Plan	`inicio_restservice/gettingstarted/wasce/geronimo-web.xml`

5) Pulse el botón Instalar.

   La página de la consola debe indicar que la aplicación se ha instalado e iniciado satisfactoriamente.

6) Examine el registro de salida del sistema WebSphere Application Server Community Edition o la consola para verificar que el servicio de datos REST se ha iniciado satisfactoriamente comprobando si el mensaje siguiente está presente:

   ```
   CWOBJ4000I: Se ha iniciado el servicio de datos REST de WebSphere eXtreme Scale.
   ```

5. Verifique que el servicio de datos REST funcione
   a. Abra el enlace siguiente en una ventana de navegador: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid`. Aparecerá el documento de servicio correspondiente a la cuadrícula NorthwindGrid.
   b. Abra el enlace siguiente en una ventana de navegador: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/$metadata`. Aparecerá el documento de extensiones de datos del modelo de entidad (EDMX).
6. Para detener los procesos de cuadrícula, utilice la combinación de teclas `CTRL+C` en la ventana de mandato respectiva.
7. Para detener WebSphere Application Server Community Edition, utilice el mandato siguiente:
   - UNIX Linux `raíz_wasce/bin/shutdown.sh`
   - Windows `raíz_wasce\bin\shutdown.bat`

   Consejo: El ID de usuario predeterminado es `system` y la contraseña es `manager`.Si utiliza un puerto personalizado, utilice la opción `-port`.

Inicio de los servicios de datos REST en Apache Tomcat

Este tema describe cómo configurar e iniciar el servicio de datos REST de eXtreme Scale utilizando Apache Tomcat versión 5.5 o posterior.

Antes de empezar

Verifique que la cuadrícula de eXtreme Scale de ejemplo se haya iniciado. Consulte el apartado “Habilitación del servicio de datos REST” en la página 224 para obtener información detallada sobre cómo iniciar la cuadrícula.

Procedimiento

1. Descargue e instale Apache Tomcat Versión 5.5 o posterior en raíz_tomcat, por ejemplo: `/opt/tomcat`
2. Instale el servicio de datos REST de eXtreme Scale y el ejemplo proporcionado en el servidor Tomcat tal como se indica a continuación:
   a. Si utiliza un Sun JRE o JDK, debe instalar IBM ORB en Tomcat:
      - Para Tomcat versión 5.5

        Copie todos los archivos JAR de:

        `inicio_wxs/lib/endorsed`

        a

        `raíz_tomcat/common/endorsed`
      - Para Tomcat versión 6.0
        1) Cree un directorio "endorsed"
           – UNIX Linux `mkdir raíz_tomcat/endorsed`
           – Windows `md raíz_tomcat/endorsed`
        2) Copie todos los archivos JAR de:

           `inicio_wxs/lib/endorsed`

           a

           `raíz_tomcat/endorsed`

   b. Despliegue el módulo del servicio de datos REST: wxsrestservice.war al servidor Tomcat.

      Copie el archivo wxsrestservice.war de:

      `inicio_restservice/lib`

      a:

      `raíz_tomcat/webapps`

   c. Añada el JAR de tiempo de ejecución del cliente ObjectGrid y el JAR de la aplicación a la vía de acceso de clases compartida de Tomcat:
      1) Edite el archivo `raíz_tomcat/conf/catalina.properties`
      2) Añada los nombres de vía de acceso siguientes al final de la propiedad shared.loader en forma de lista delimitada por comas:
         - wxs_home/lib/ogclient.jar
         - inicio_restservice/gettingstarted/restclient/bin
         - inicio_restservice/gettingstarted/common/bin

         Importante: El separador de la vía de acceso debe ser una **barra inclinada**.
3. Si se ha iniciado la cuadrícula eXtreme Scale con la seguridad de eXtreme Scale habilitada, establezca las propiedades siguientes en el archivo `restservice_home/gettingstarted/restclient/bin/wxsRestService.properties`.

```
ogClientPropertyFile=inicio_restservice/gettingstarted/security/security.ogclient.properties
loginType=none
```

4. Inicie el servidor Tomcat con el servicio de datos REST:
   - Si utiliza Tomcat 5.5 en UNIX® o Windows®, o bien Tomcat 6.0 en UNIX:
     a. `cd raíz_tomcat/bin`
     b. Inicie el servidor:
        – UNIX Linux `./catalina.sh run`

– Windows `catalina.bat run`

c. La consola mostrará entonces los registros de Apache Tomcat. Cuando el servicio de datos REST se haya iniciado con éxito, aparecerá el mensaje siguiente en la consola de administración:

`CWOBJ4000I: Se ha iniciado el servicio de datos REST de WebSphere eXtreme Scale.`

- Si utiliza Tomcat 6.0 en Windows:
  a. `cd raíz_tomcat/bin`
  b. Inicie la herramienta de configuración de Apache Tomcat 6 con el mandato siguiente: `tomcat6w.exe`
  c. Pulse el botón Inicio de la ventana de propiedades de Apache Tomcat 6 para iniciar al servidor Tomcat.
  d. Examine los registros siguientes para verificar que el servidor Tomcat se haya iniciado con éxito:
     – `raíz_tomcat/bin/catalina.log`

       Muestra el estado del motor del servidor Tomcat
     – `raíz_tomcat/bin/stdout.log`

       Muestra el registro de salida del sistema.
  e. Cuando el servicio de datos REST se haya iniciado con éxito, aparecerá el mensaje siguiente en el registro de salida del sistema: `CWOBJ4000I: Se ha iniciado el servicio de datos REST de WebSphere eXtreme Scale.`

5. Verifique que el servicio de datos REST funcione
   a. Abra un navegador y acceda a:

      http://localhost:8080/wxsrestservice/restservice/NorthwindGrid

      Aparecerá el documento de servicio correspondiente a NorthwindGrid.
   b. Vaya a:

      http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/$metadata

      Aparecerá el documento de extensiones de datos del modelo de entidad (EDMX).
6. Para detener los procesos de cuadrícula, utilice CTRL+C en la ventana de mandato respectiva.
7. Para detener Tomcat, utilice CTRL +C en la ventana en la cual lo haya iniciado.

Uso de un navegador con los servicios de datos REST

El servicio de datos REST de eXtreme Scale crea canales de información ATOM de forma predeterminada al utilizar un navegador web. Es posible que el formato del canal de información ATOM no sea compatible con navegadores antiguos o que se interprete de forma que los datos no se puedan ver como XML. Los temas siguientes proporcionan información detallada sobre cómo configurar Internet Explorer Versión 8 y Firefox Versión 3 para mostrar los canales de información ATOM y XML en el navegador.

Acerca de esta tarea

El servicio de datos REST de eXtreme Scale crea canales de información ATOM de forma predeterminada al utilizar un navegador web. Es posible que el formato del canal de información ATOM no sea compatible con navegadores antiguos o que se interprete de forma que los datos no se puedan ver como XML. En el caso de los navegadores antiguos, se le solicitará que guarde los archivos en el disco. Cuando

se hayan descargado los archivos, utilice su lector XML favorito para examinar los archivos. Al XML generado no se la da formato para visualizarlo, de modo que todo se imprimirá en una sola línea. La mayor parte de los programas que leen XML, como Eclipse, permiten cambiar el formato XML por un formato legible.

En el caso de los navegadores modernos, Microsoft Internet Explorer Versión 8 y Firefox Versión 3, los archivos XML ATOM se pueden mostrar de forma nativa en el navegador. Los temas siguientes proporcionan información detallada sobre cómo configurar Internet Explorer Versión 8 y Firefox Versión 3 para mostrar los canales de información ATOM y XML en el navegador.

Procedimiento

Configuración de Internet Explorer Versión 8

- Para permitir que Internet Explorer lea los canales de información ATOM que el servicio de datos REST genera. siga los pasos que se indican a continuación:
  1. Pulse **Herramientas** → **Opciones de Internet**
  2. Seleccione el separador **Contenido**
  3. Pulse el botón **Configuración** de la sección **Fuentes y Web Slices**
  4. Quite la marca del recuadro: "Activar la vista de lectura de fuentes"
  5. Pulse **Aceptar** para volver al navegador.
  6. Reinicie Internet Explorer.

Configuración de Firefox Version 3

- Firefox no muestra automáticamente las páginas con el tipo de contenido application/atom+xml. La primera vez que aparezca una página de este tipo, Firefox le solicitará que guarde el archivo. Para visualizar la página, abra el propio archivo con Firefox tal como se indica a continuación:
  1. En el recuadro de diálogo de selección de aplicaciones, seleccione el botón de selección "Abrir con" y pulse el botón **Examinar**.
  2. Vaya al directorio de instalación de Firefox. Por ejemplo: `C:\Archivos de programa\Mozilla Firefox`
  3. Seleccione `firefox.exe` y pulse el botón **Aceptar**.
  4. Marque el recuadro de selección “Utilizar siempre el programa seleccionado para abrir este tipo de archivos”.
  5. Pulse el botón **Aceptar**.
  6. A continuación, Firefox mostrará la página XML ATOM en una ventana o pestaña de navegador nueva
- Firefox representa automáticamente los canales de información ATOM en formato legible. No obstante, los canales de información que el servicio de datos REST crea incluyen XML. Firefox no puede mostrar el XML a menos que se inhabilite el representador de canales de información. A diferencia de Internet Explorer, en Firefox, el plug-in de representación de canales de información ATOM se debe editar explícitamente. Para configurar Firefox para leer canales de información ATOM como archivos XML, siga estos pasos:
  1. Abra el archivo siguiente en un editor de texto: `<raízInstalFirefox>\components\FeedConverter.js`. En la vía de acceso, `<raízInstalFirefox>` es el directorio raíz donde está instalado Firefox.

     Para los sistemas operativos Windows, el directorio predeterminado es: `C:\Archivos de programas\Mozilla Firefox`.
  2. Busque el fragmento de código igual al siguiente:

```
// show the feed page if it wasn't sniffed and we have a document,
// or we have a document, title, and link or id
if (result.doc && (!this._sniffed ||
    (result.doc.title && (result.doc.link || result.doc.id)))) {
```

3. Marque como comentario las dos líneas que empiezan por `if` y `result` colocando `//` (dos barras inclinadas) delante de ellas.
4. Añada la siguiente sentencia al fragmento de código: `if(0) {`.
5. El texto resultante debe ser como el que se indica a continuación:

```
// show the feed page if it wasn't sniffed and we have a document,
// or we have a document, title, and link or id
//if (result.doc && (!this._sniffed ||
//    (result.doc.title && (result.doc.link || result.doc.id)))) {
if(0) {
```

6. Guarde el archivo.
7. Reinicie Firefox
8. Ahora Firefox puede mostrar automáticamente todos los canales de información en el navegador.

- Pruebe su configuración intentando algunos URL.

Ejemplo

Esta sección describe algunos URL de ejemplo que se pueden utilizar para ver los datos añadidos por el ejemplo de la guía de inicio proporcionada con el servicio de datos REST de eXtreme Scale. Antes de utilizar los URL siguientes, añada el conjunto de datos predeterminado a la cuadrícula de ejemplo de eXtreme Scale utilizando el cliente Java de ejemplo o el cliente de WCF Data Services de Visual Studio de ejemplo.

En los ejemplos siguientes se presupone que el puerto es 8080, aunque puede variar. Consulte la sección correspondiente para obtener información detallada sobre cómo configurar los servicios de datos REST en distintos servidores de aplicaciones.

- Ver un solo cliente con el ID "ACME":

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')
  ```

- Ver todos los pedidos del cliente "ACME":

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')/orders
  ```

- Ver el cliente "ACME" y los pedidos:

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')?$expand=orders
  ```

- Ver el pedido 1000 del cliente "ACME":

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
  Order(orderId=1000,customer_customerId='ACME')
  ```

- Ver el pedido 1000 del cliente "ACME" y su cliente asociado:

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=1000,customer_customerId='ACME')?$expand=customer
  ```

- Ver el pedido 1000 del cliente "ACME" y su cliente y detalles de pedido asociados:

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
  Order(orderId=1000,customer_customerId='ACME')?$expand=customer,orderDetails
  ```

- Ver todos los pedidos del cliente "ACME" del mes de octubre de 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'
  ```

- Ver los 3 primeros pedidos y detalles de pedido del cliente "ACME" del mes de octubre de 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
```

Uso de un cliente Java con los servicios de datos REST

La aplicación cliente Java utiliza la API EntityManager de eXtreme Scale para insertar datos en la cuadrícula.

Acerca de esta tarea

Las secciones anteriores describían cómo crear una cuadrícula de eXtreme Scale y configurar e iniciar el servicio de datos REST de eXtreme Scale. La aplicación cliente Java utiliza la API EntityManager de eXtreme Scale para insertar datos en la cuadrícula.No demuestra cómo utilizar las interfaces de REST. El objetivo de este cliente es demostrar cómo se utiliza la API EntityManager para interactuar con la cuadrícula de eXtreme Scale y permitir modificar datos en la cuadrícula. Para ver datos de la cuadrícula utilizando el servicio de datos REST, utilice un navegador web o utilice la aplicación cliente Visual Studio 2008.

Procedimiento

Para añadir rápidamente contenido a la cuadrícula de eXtreme Scale, ejecute el mandato siguiente:

1. Abra una línea de mandatos o una ventana de terminal y defina la variable de entorno JAVA_HOME:
   - Linux UNIX `export JAVA_HOME=`*`inicio_java`*
   - Windows `set JAVA_HOME=`*`inicio_java`*
2. `cd restservice_home/gettingstarted`
3. Inserte datos en la cuadrícula. Los datos que se inserten se recuperarán posteriormente utilizando un navegador web y el servicio de datos REST.

   Si se ha iniciado la cuadrícula *sin* la seguridad de eXtreme Scale, utilice los mandatos siguientes.
   - UNIX Linux `./runclient.sh load default`
   - Windows `runclient.bat load default`

   Si se ha iniciado la cuadrícula *con* la seguridad de eXtreme Scale, utilice los mandatos siguientes.
   - UNIX Linux `./runclient_secure.sh load default`
   - Windows `runclient_secure.bat load default`

   Para un cliente Java client, utilice la sintaxis del mandato siguiente:
   - UNIX Linux `runclient.sh` *`mandato`*
   - Windows `runclient.bat` *`mandato`*

   Están disponibles los siguientes mandatos:
   - `load default`

     Carga un conjunto predefinido de entidades Customer, Category y Product en la cuadrícula y crea un conjunto aleatorio de pedidos para cada cliente.
   - `load category` *`categoryId categoryName firstProductId num_products`*

Crea una categoría de producto y un número fijo de entidades Product en la cuadrícula. El parámetro firstProductId identifica el número de ID del primer producto y a cada producto siguiente se le asigna el ID siguiente hasta que se crea el número de productos especificado.

- `load customer` *`companyCode contactNamecompanyName numOrders firstOrderIdshipCity maxItems discountPct`*

  Carga un cliente nuevo en la cuadrícula y crea un conjunto fijo de entidades Order para cualquier producto aleatorio actualmente cargado en la cuadrícula. El número de entidades Order se determina definiendo el parámetro <numOrders>. Cada Order tendrá un número aleatorio de entidades OrderDetail hasta alcanzar el valor de <maxItems>

- `display customer` *`companyCode`*

  Muestra una entidad Customer y las entidades Order y OrderDetail asociadas.

- `display category` *`categoryId`*

  Muestra una entidad Category de producto y las entidades Product asociadas.

Resultados

- `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`

Ejecución y creación de la cuadrícula de ejemplo y el cliente Java con Eclipse

El ejemplo de iniciación del servicio de datos REST se puede actualizar y ampliar mediante Eclipse. Para obtener información detallada sobre cómo configurar el entorno de Eclipse, consulte el documento de texto: `inicio_restservice/gettingstarted/ECLIPSE_README.txt`.

Después de que el proyecto WXSRestGettingStarted se haya importado a Eclipse y se haya creado satisfactoriamente, el ejemplo volverá a compilar automáticamente los archivos de script utilizados para iniciar el servidor de contenedores y el cliente obtendrá automáticamente los archivos de clases y los archivos XML. El servicio de datos REST también detectará automáticamente cualquier cambio, ya que el servidor web está configurado para leer los directorios de compilación de Eclipse automáticamente.

Importante: Al cambiar los archivos fuente o de configuración, tanto el servidor de contenedores de eXtreme Scale como la aplicación del servicio de datos REST se deben reiniciar. El servidor de contenedores de eXtreme Scale se debe iniciar antes de la aplicación web del servicio de datos REST.

Cliente Visual Studio 2008 WCF con servicio de datos REST

El ejemplo de iniciación del servicio de datos REST de eXtreme Scale incluye un cliente WCF Data Services que puede interactuar con el servicio de datos REST de eXtreme Scale. El ejemplo está escrito como una aplicación de línea de mandatos en C#.

Requisitos de software

El cliente de ejemplo C# de WCF Data Services requiere lo siguiente:

- Sistema operativo
  - Microsoft Windows XP
  - Microsoft Windows Server 2003
  - Microsoft Windows Server 2008
  - Microsoft Windows Vista
- Microsoft Visual Studio 2008 con Service Pack 1

  Consejo: Consulte el enlace anterior para conocer los requisitos adicionales de hardware y software.
- Microsoft .NET Framework 3.5 Service Pack 1
- Microsoft Support: An update for the .NET Framework 3.5 Service Pack 1 is available (Soporte de Microsoft: hay una actualización de NET Framework 3.5 Service Pack 1 disponible)

Creación y ejecución del cliente de iniciación

El cliente de ejemplo de WCF Data Services incluye un proyecto y una solución de Visual Studio 2008 y el código fuente para ejecutar el ejemplo. El ejemplo se debe cargar en Visual Studio 2008 y se debe compilar en un programa Windows ejecutable para poder ejecutarlo. Para crear y ejecutar el ejemplo, consulte el documento de texto siguiente: `inicio_restservice/gettingstarted/VS2008_README.txt`.

Sintaxis del mandato del cliente C# de WCF Data Services

Windows `WXSRESTGettingStarted.exe <URL de servicio> <mandato>`

El <URL de servicio> es el URL del servicio de datos REST de eXtreme Scale configurado en la sección correspondiente.

Están disponibles los siguientes mandatos:

- `load default`

  Carga un conjunto predefinido de entidades Customer, Category y Product en la cuadrícula y crea un conjunto aleatorio de pedidos para cada cliente.
- `load category <categoryId> <categoryName> <firstProductId> <numProducts>`

  Crea una categoría de producto y un número fijo de entidades Product en la cuadrícula. El parámetro firstProductId identifica el número de ID del primer producto y a cada producto siguiente se le asigna el ID siguiente hasta que se crea el número de productos especificado.
- `load customer <companyCode> <contactName> <companyName> <numOrders> <firstOrderId> <shipCity> <maxItems> <discountPct>`

  Carga un cliente nuevo en la cuadrícula y crea un conjunto fijo de entidades Order para cualquier producto aleatorio actualmente cargado en la cuadrícula. El número de entidades Order se determina definiendo el parámetro <numOrders>. Cada Order tendrá un número aleatorio de entidades OrderDetail hasta alcanzar el valor de <maxItems>
- `display customer <companyCode>`

  Muestra una entidad Customer y las entidades Order y OrderDetail asociadas.

- `display category <categoryId>`
  Muestra una entidad Category de producto y las entidades Product asociadas.
- `unload`
  Elimina todas las entidades cargadas utilizando el mandato "default load".

Los ejemplos siguientes ilustran distintos mandatos.

- `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`

Avisos

Las referencias en esta publicación a productos, programas o servicios de IBM no implica que IBM tenga previsto ponerlos a la venta en todos los países en los que IBM opera. Cualquier referencia a un producto, programa o servicio de IBM no pretende indicar ni implica que sólo se pueda utilizar este producto, programa o servicio de IBM. En su lugar, se puede utilizar cualquier producto, programa o servicio funcionalmente equivalente que no vulnere ningún derecho de propiedad intelectual de IBM. La evaluación y la verificación del funcionamiento con otros productos, excepto aquellos expresamente designados por IBM, es responsabilidad del usuario.

IBM puede tener patentes o solicitudes de patentes pendientes que conciernan al tema de este documento. La posesión de este documento no le da ninguna licencia sobre estas patentes. Puede enviar preguntas acerca de licencias por escrito a:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, New York 10594 Estados Unidos

Los propietarios de licencias de este programa que deseen obtener información sobre el mismo con el fin de habilitar: (i) el intercambio de información entre programas creados de forma independiente y otros programas (incluido este) y (ii) el uso mutuo de la información intercambiada, se deben poner en contacto con:

IBM Corporation
Mail Station P300
522 South Road
Poughkeepsie, NY 12601-5400
Estados Unidos
Attention: Information Requests

Esta información puede estar disponible, bajo las condiciones y los términos adecuados, incluyendo en algunos casos, el pago de una cuota.

Marcas registradas

Los siguientes términos son marcas registradas de IBM Corporation en Estados Unidos yen otros países.

- AIX
- CICS
- Cloudscape
- DB2
- Domino
- IBM
- Lotus
- RACF
- Redbooks
- Tivoli
- WebSphere
- z/OS

Java y todas las marcas registradas basadas en Java son marcas registradas de Sun Microsystems, Inc. en Estados Unidos y/o en otros países.

LINUX es una marca registrada de Linus Torvalds en Estados Unidos y/o en otros países.

Microsoft, Windows, Windows NT® y el logotipo de Windows son marcas registradas de Microsoft Corporation en Estados Unidos y/o en otros países.

UNIX es una marca registrada de The Open Group en Estados Unidos y en otros países.

Otros nombres de compañías, productos y servicios pueden ser marcas registradas o de servicio de terceros.

Índice

S

T

V

Impreso en España