WebSphere® eXtreme Scale versión 7.0

Visión general del producto

WebSphere® eXtreme Scale versión 7.0

Visión general del producto

Esta edición se aplica a la versión 7, release 0, de WebSphere eXtreme Scale y a todos los releases y modificaciones posteriores hasta que se indique lo contrario en nuevas ediciones.

© Copyright International Business Machines Corporation 2009, 2009.

Contenido

© Copyright IBM Corp. 2009, 2009

Figuras

© Copyright IBM Corp. 2009, 2009

Tablas

© Copyright IBM Corp. 2009, 2009

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.

© Copyright IBM Corp. 2009, 2009

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 y proporciona calidades de servicio como, por ejemplo, integridad transaccional, alta disponibilidad y tiempos de respuesta predecibles.

La escalabilidad elástica de WebSphere eXtreme Scale se hace posible a través del almacenamiento en la memoria caché de objeto distribuido. Elástico significa que la cuadricula se supervisa y gestiona a sí misma, permite la ampliación y la reducción y se repara a sí misma recuperándose automáticamente de los errores. La ampliación permite añadir capacidad de memoria mientras la cuadrícula está en ejecución, sin necesitar reiniciarse. Por el contrario, la reducción permite eliminar inmediatamente la capacidad de memoria.

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.

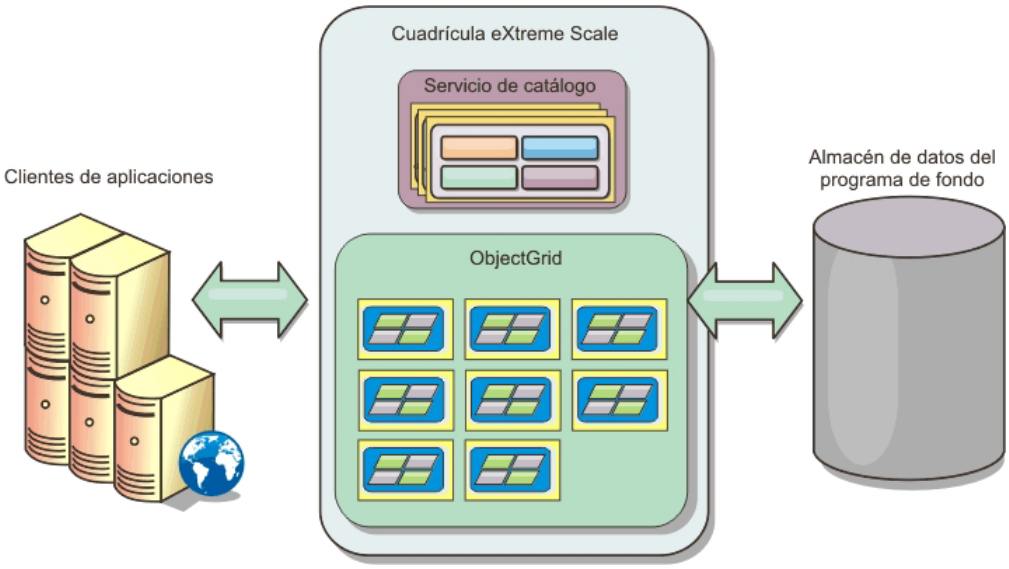
Sin embargo, es importante tener en cuenta que no se puede considerar a eXtreme Scale como una base de datos de memoria real, en su mayor parte porque el último es demasiado sencillo para manejar algunas de las complejidades que puede gestionar eXtreme Scale. Tendría algunas de las mismas ventajas con 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), cada una de ellas existe como una copia primaria (fragmento) y también fragmentos de réplicas para la 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, por supuesto, 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.

La característica de memoria caché de grabación diferida permite a WebSphere eXtreme Scale actuar como una memoria caché frontal para una base de datos para aumentar el rendimiento, mientras se reduce la carga y los conflictos. WebSphere eXtreme Scale proporciona una escalada horizontal y una escalada vertical a un coste de proceso predecible.

© Copyright IBM Corp. 2009, 2009

La siguiente imagen muestra que en un entorno distribuido de memoria caché coherente, los clientes de la cuadrícula eXtreme Scale envían y reciben datos de la cuadrícula, 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, y se amplía de forma lineal, a medida que la cuadrícula crece en tamaño. De forma opcional, también se pueden duplicar los datos para la tolerancia al error adicional.



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. No está limitado por, ni tendrá repercusión en, 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 básico es un par de clave-valor, donde la cuadrícula almacena valores (objetos Java), con una clave asociada (otro objeto Java), a través del cual se recupera el valor posteriormente. En eXtreme Scale, una correlación es un contenedor para estos pares de clave-valor.

WebSphere eXtreme Scale ofrece una serie de configuraciones de cuadrícula, desde una única, una memoria caché local simple a una memoria caché distribuida grande, utilizando varios JVMS y/o servidores.

Además de almacenar objetos Java sencillos, los objetos con relaciones se puede almacenar y se puede utilizar un lenguaje de consulta parecido a SQL (SELECT ... FROM ... WHERE) para recuperarlos. 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 las entidades de la memoria caché, de la misma forma que 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 EntityManager. Las entidades se pueden conservar en la cuadrícula, así como ser eliminadas, todos 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 el soporte de Real Time en *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, o ObjectGrid en el vocabulario de eXtreme Scale), configure el número de servidores eXtreme que contendrá los datos de cuadrícula 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 asignará 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.

Además de almacenar en la memoria caché los objetos Java sencillos, los objetos con relaciones se pueden almacenar en la memoria caché y se puede utilizar un lenguaje de consulta parecido a SQL (SELECT ... FROM ... WHERE) para recuperarlos. 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. También está soportada una interfaz de programación EntityManager en eXtreme Scale, para almacenar entidades en la memoria caché, de la misma forma que las entidades 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. Por lo tanto, un entidad se puede recuperar de la memoria caché por la clave primaria utilizando el método find de EntityManager. Las entidades se pueden conservar en la cuadrícula, así como ser eliminadas, todos dentro de un límite de transacción.

Características nuevas y en desuso de este release

WebSphere eXtreme Scale incluye muchas características nuevas en la versión 7.0, incluida la integración con la memoria caché dinámica, las correlaciones de matrices de bytes y más características.

Novedades en WebSphere eXtreme Scale versión 7.0

Tabla 1. Nuevas características en WebSphere eXtreme Scale versión 7.0

Característica	Descripción
Integración de memoria caché dinámica	el proveedor de la memoria caché dinámica permite a las aplicaciones que utilizan la memoria caché dinámica de WebSphere Application Server sacar el máximo partido de las características avanzadas y las mejoras de rendimiento de WebSphere eXtreme Scale. Esta característica genera una mayor calidad de servicio, mayor escalabilidad lineal y una alta disponibilidad para llevarla a una amplia variedad de aplicaciones empresariales con los mínimos cambios invasivos. Consulte la información sobre la memoria caché dinámica en *Visión general del producto*.
Correlaciones de matriz de bytes	Con las correlaciones de matrices de bytes, la aplicación puede almacenar el valor de un par clave-valor en una matriz de bytes en lugar de un formato de objeto, que reduce la huella de memoria que puede consumir un gran gráfico de objetos. Consulte la información sobre las correlaciones de matrices de bytes en *Guía de programación*.
WebSphere Real Time	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 el soporte de Real Time en *Guía de administración*.
Habilitación de la métrica	WebSphere eXtreme Scale incluye las implementaciones de los adaptadores de acceso de métrica para mejorar la integración con IBM® Tivoli Monitoring (ITM) e Hyperic HQ, lo que habilita una visión global del comportamiento operacional de las soluciones empresariales. Consulte la información sobre las herramientas de supervisión de proveedor para la habilitación de la métrica en *Guía de administración*..
Correlaciones dinámicas	La creación de aplicaciones de varios inquilinos se ha simplificado en gran medida. Las plantillas de correlación permiten a las aplicaciones crear nuevas correlaciones bajo demanda, lo que evita la necesidad de utilizar los discriminadores de aplicaciones en las claves o crear correlaciones adicionales que no se pueden utilizar nunca. Consulte la información sobre correlaciones dinámicas en *Guía de programación*..
Tiempo de espera de solicitud	WebSphere eXtreme Scale se ha ampliado para manejar muchas de las tareas comunes de la lógica de excepción y reintento dentro de la middleware de cuadrícula. El tiempo de espera de la solicitud para los clientes elimina las cargas para la lógica de reintento del texto modelo para la mayoría de las operaciones de interacción de correlación. La mayoría de las condiciones que se pueden recuperar ahora se manejan automáticamente, de forma que se puede centrar en los aspectos de la lógica empresarial del desarrollo de aplicaciones. Consulte la información sobre el tiempo de espera de la solicitud en *Guía de administración*..
Índice compuesto	Esta característica puede simplificar el uso del índice cuando se consultan varios atributos y reduce la sobrecarga de tener varios índices definidos. La consulta también se ha optimizado para sacar partido de los índices compuestos. Consulte la información sobre índices compuestos en *Guía de programación*..

Características en desuso

Tabla 2. Características en desuso

En desuso	Acción de migración recomendada
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.

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.

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.

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.

Planificación del despliegue de la aplicación

Antes de desplegar WebSphere eXtreme Scale en un entorno de producción, considere las siguientes opciones.

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?

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.

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 (Jave 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.

Capítulo 2. Conceptos relacionados con la 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 eXtreme Scale. En esta sección se describen distintos conceptos y escenarios de almacenamiento en memoria caché y describe las topologías disponibles para desplegar una cuadrícula eXtreme Scale.

Arquitectura y topología

Con WebSphere eXtreme Scale , puede configurar el almacenamiento en memoria caché de datos en memoria local y 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 1. 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.

© Copyright IBM Corp. 2009, 2009

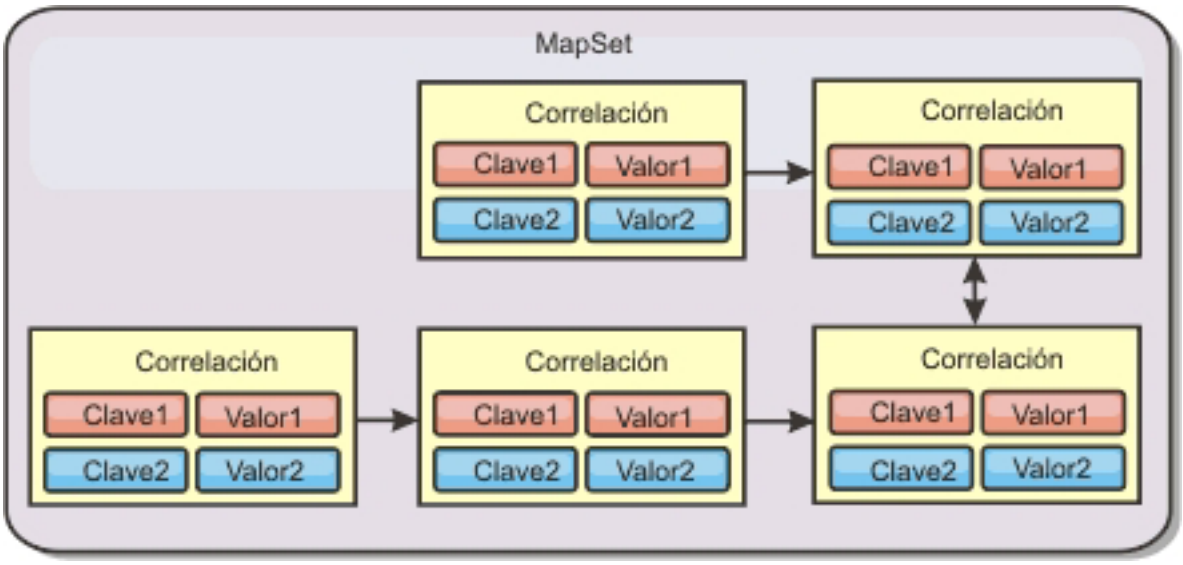


Figura 2. Conjuntos de correlaciones

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.

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. eXtreme Scale puede tener varios esquemas de correlación definidos. Consulte la información de la API ObjectMap en *Guía de programación* si desea detalles adicionales.

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. 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.

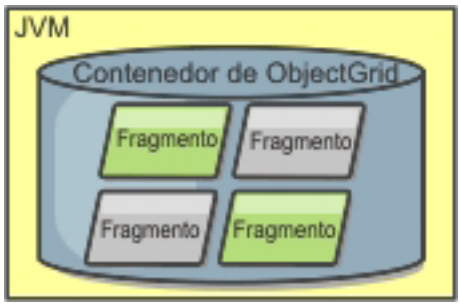


Figura 3. Contenedor

Las particiones alojan un subconjunto de los datos en la cuadrícula. 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 el ObjectGrid no puede volver a realizar hash de todo el conjunto de datos, una vez que se haya desplegado. Calcule en exceso el número de particiones.

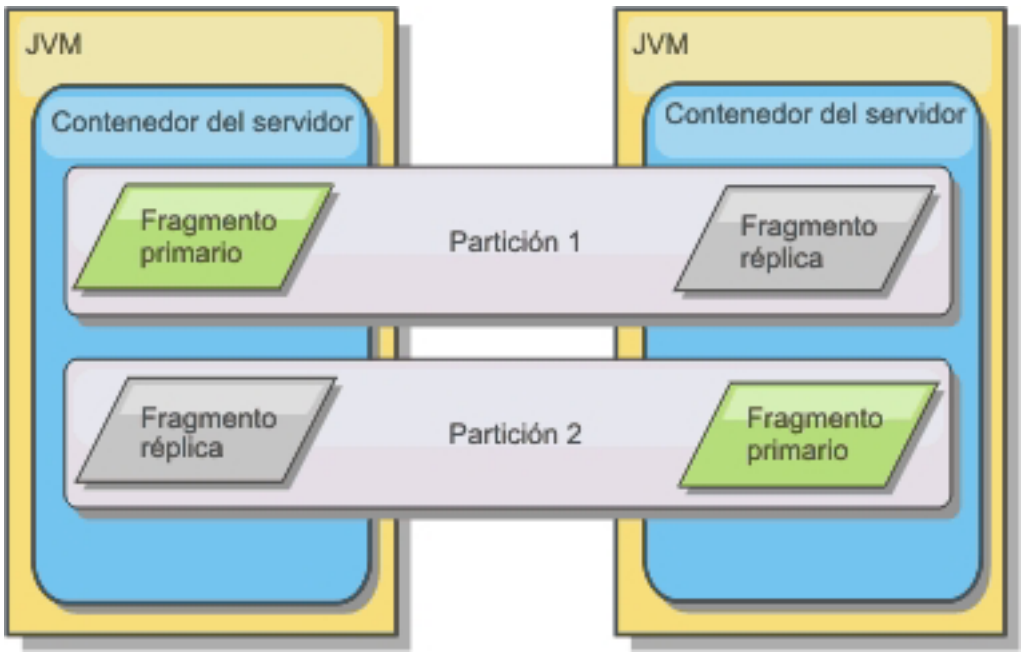


Figura 4. Partición

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.

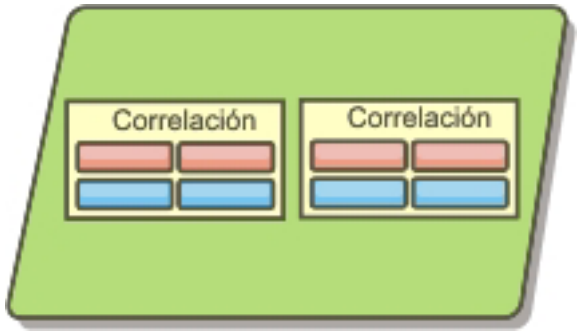


Figura 5. Fragmento

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.

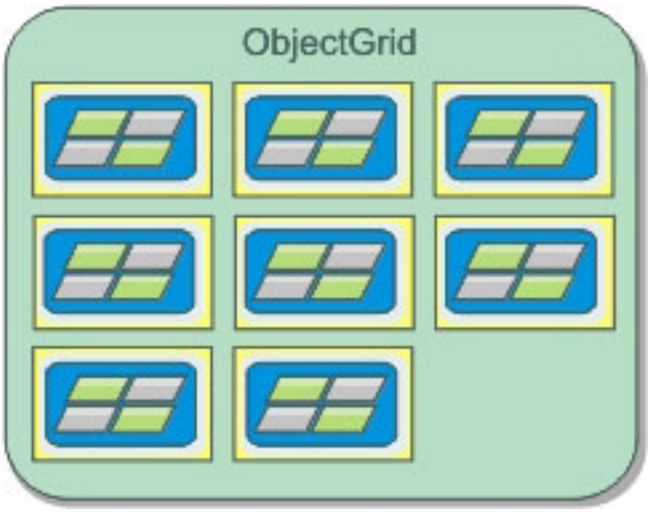


Figura 6. ObjectGrid

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, se direcciona automáticamente el cliente al servidor apropiado que incluyen 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 diversas instancias 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.

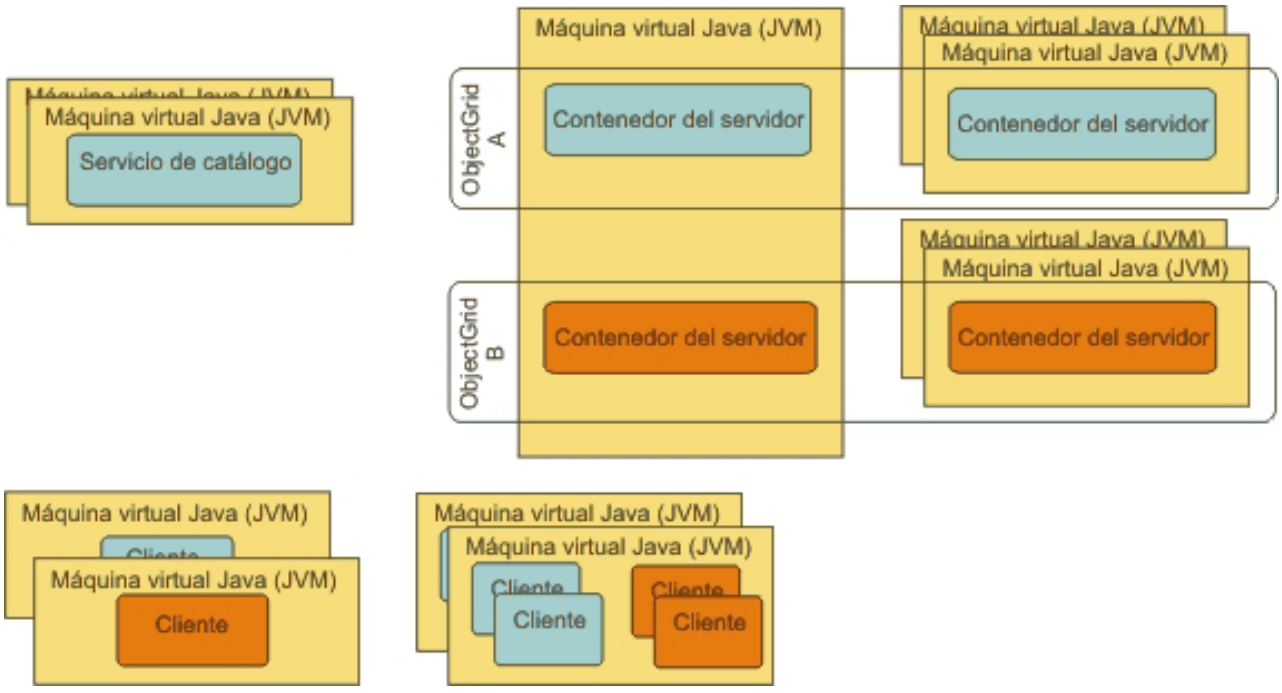


Figura 7. Topologías posibles

Servicio 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.

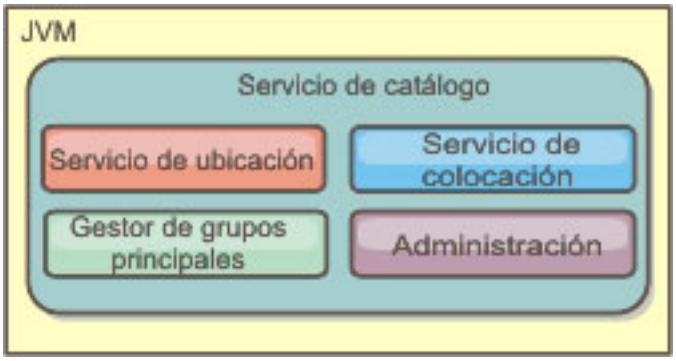


Figura 8. Servicio de catálogo

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 de modo que 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 una cuadrícula de servicio de catálogo. Una cuadrícula de servicio de catálogo está formada por varias máquinas virtuales Java, incluida una JVM maestra y una serie de máquinas virtuales Java de copia de seguridad.

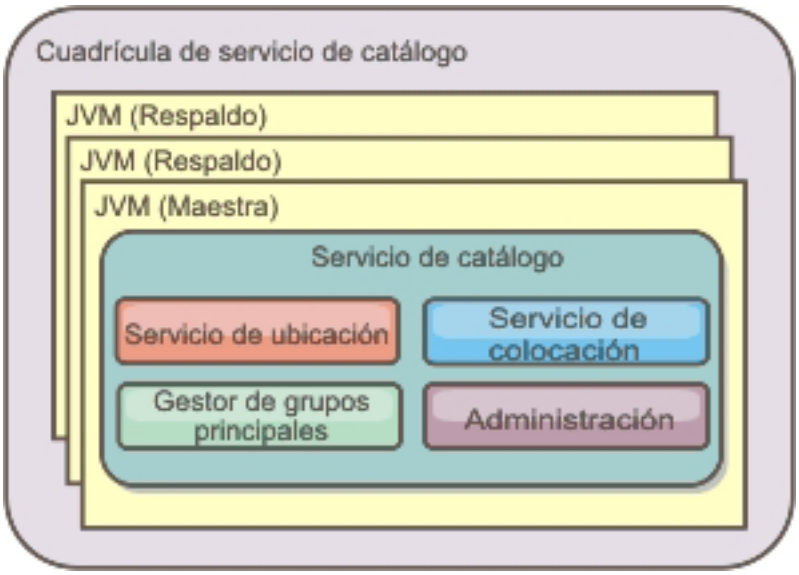


Figura 9. Cuadrícula de servicio de catálogo

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. Esto 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.

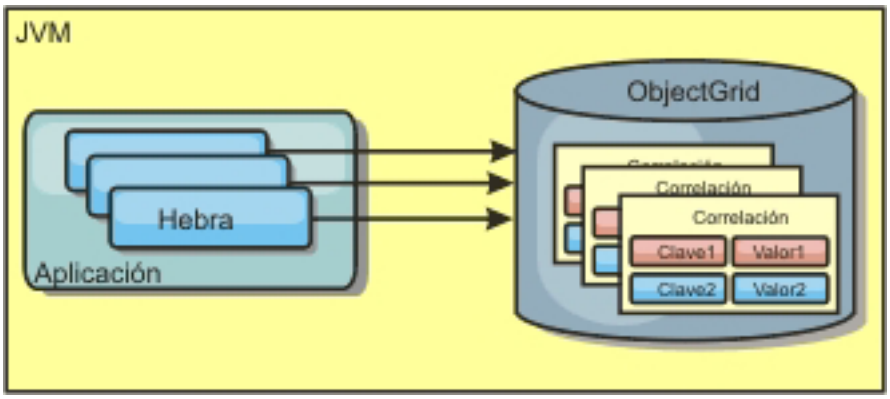


Figura 10. Escenario de memoria caché en memoria local

Ventajas

- Fácil configuración: se puede crear un ObjectGrid a través de un programa o de forma declarativa con el archivo XML de 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

Una de las limitaciones de una memoria caché de WebSphere eXtreme Scale local es que si hay varios procesos con instancias de memoria caché independientes, es difícil mantener la sincronización de la memoria caché.

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

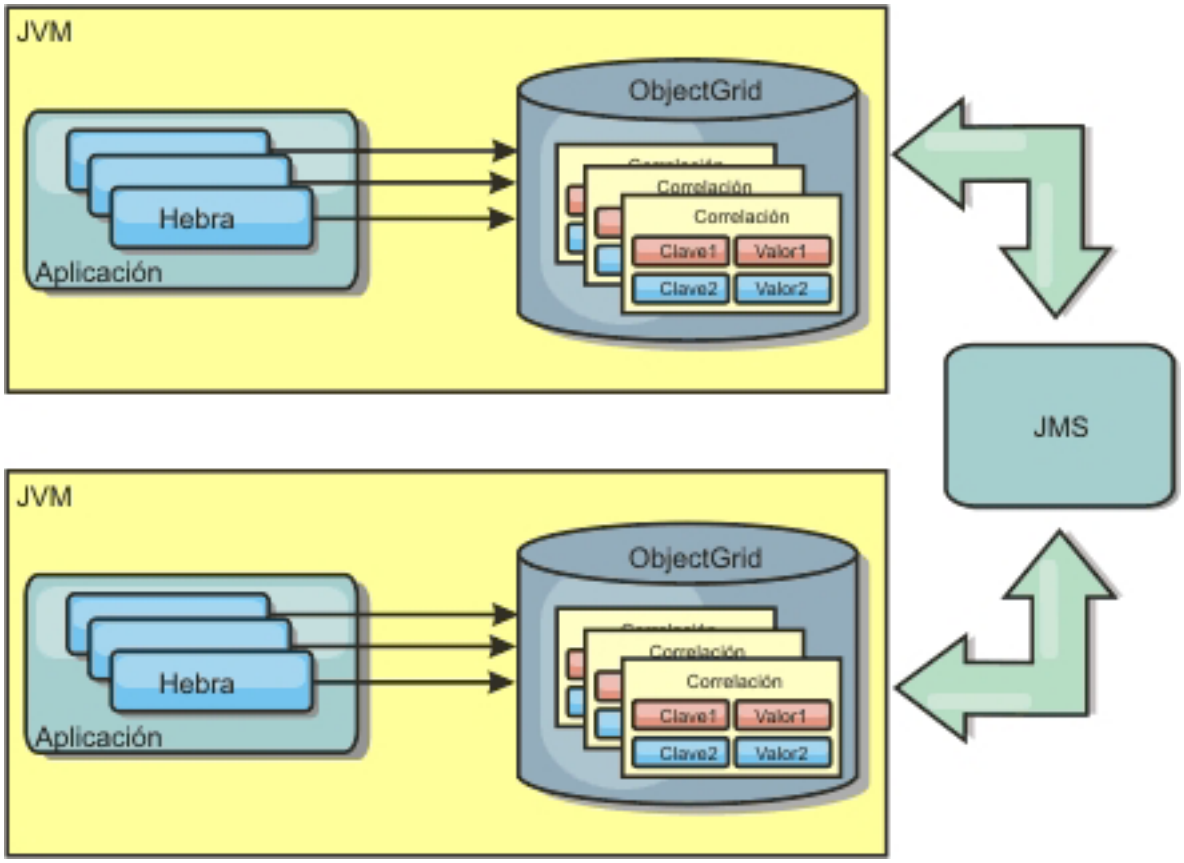


Figura 11. La memoria caché duplicada por un igual con los cambios que se propagan con JMS

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.

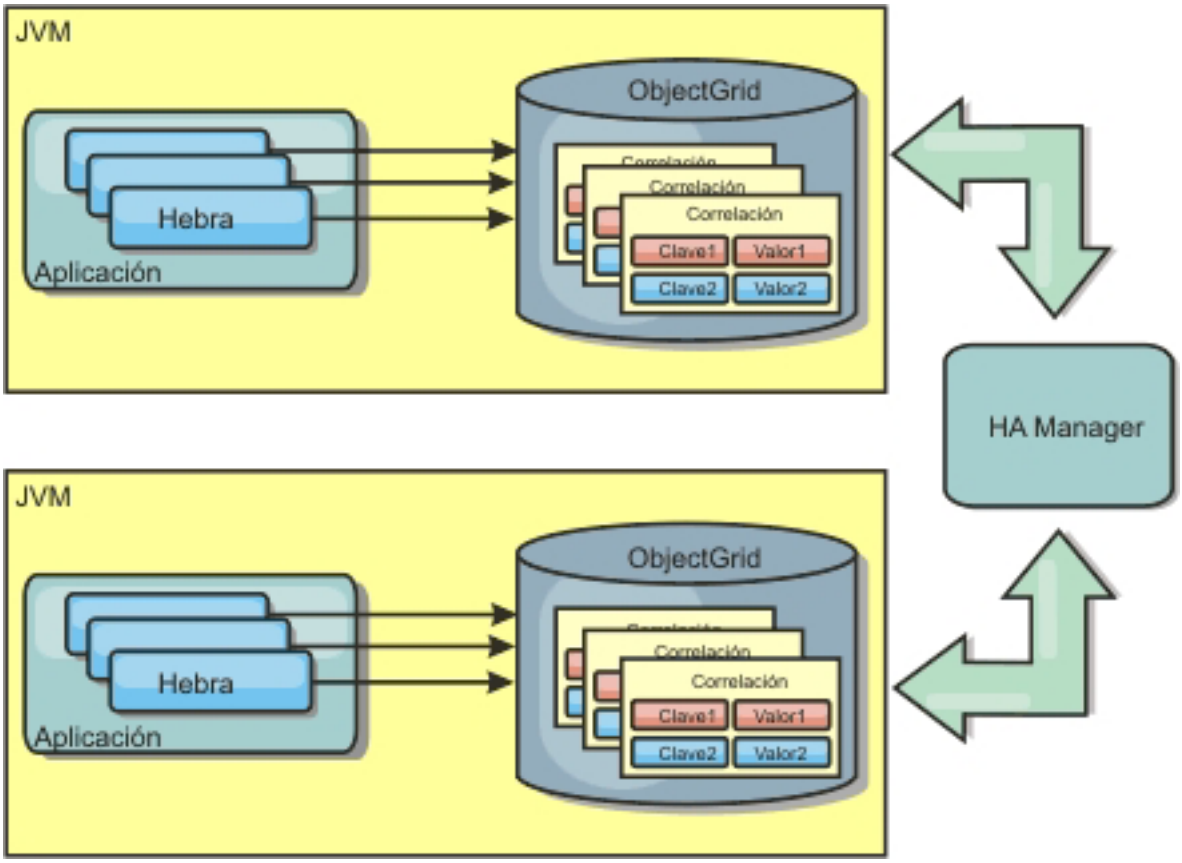


Figura 12. La memoria caché duplicada por un igual con los cambios propagados con el High Availability Manager.

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. De esta manera se facilitan el desarrollo y el despliegue de la aplicación, ya que se 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.

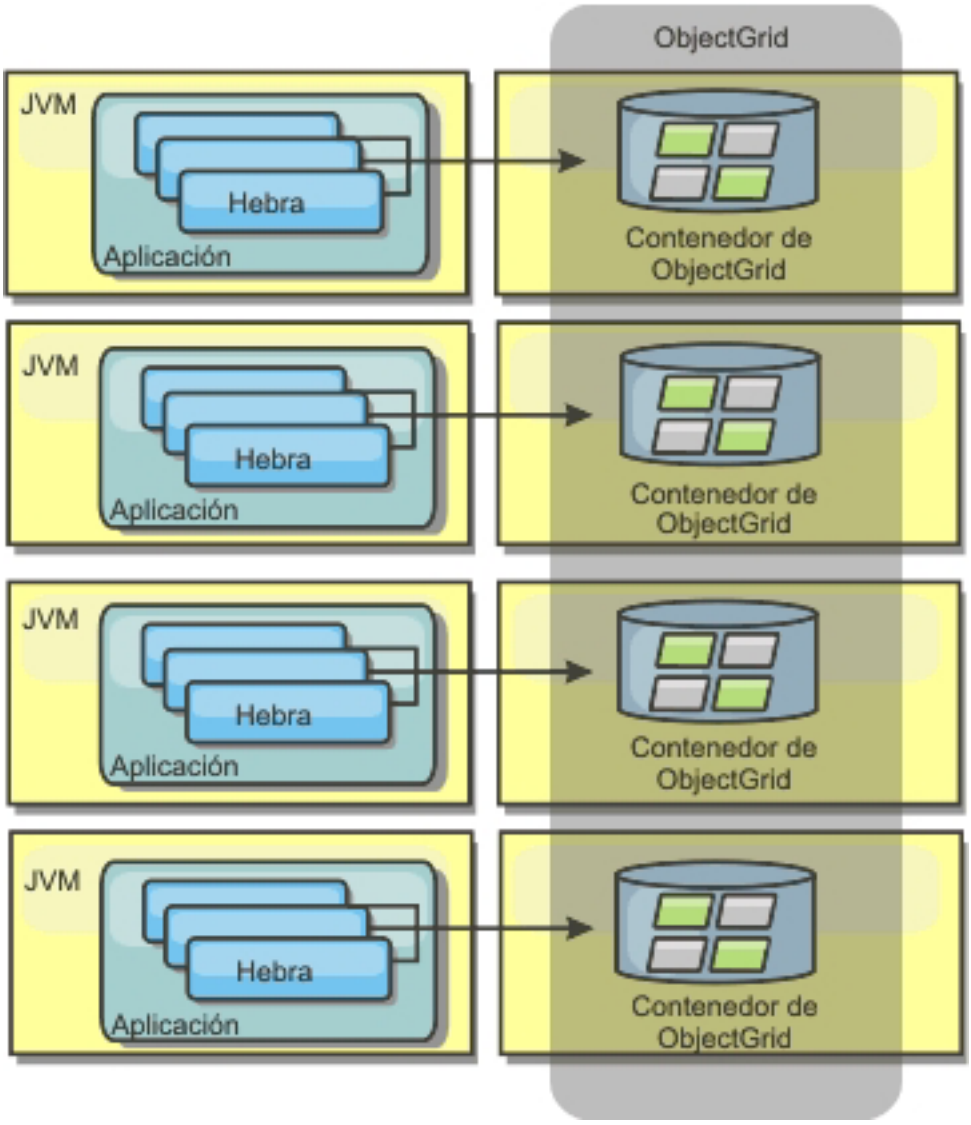


Figura 13. Memoria caché distribuida

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.

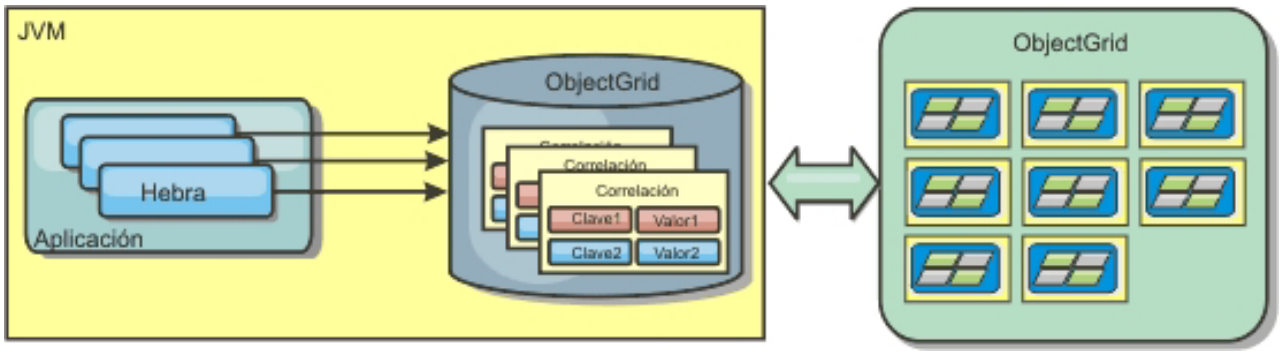


Figura 14. Memoria caché cercana

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.

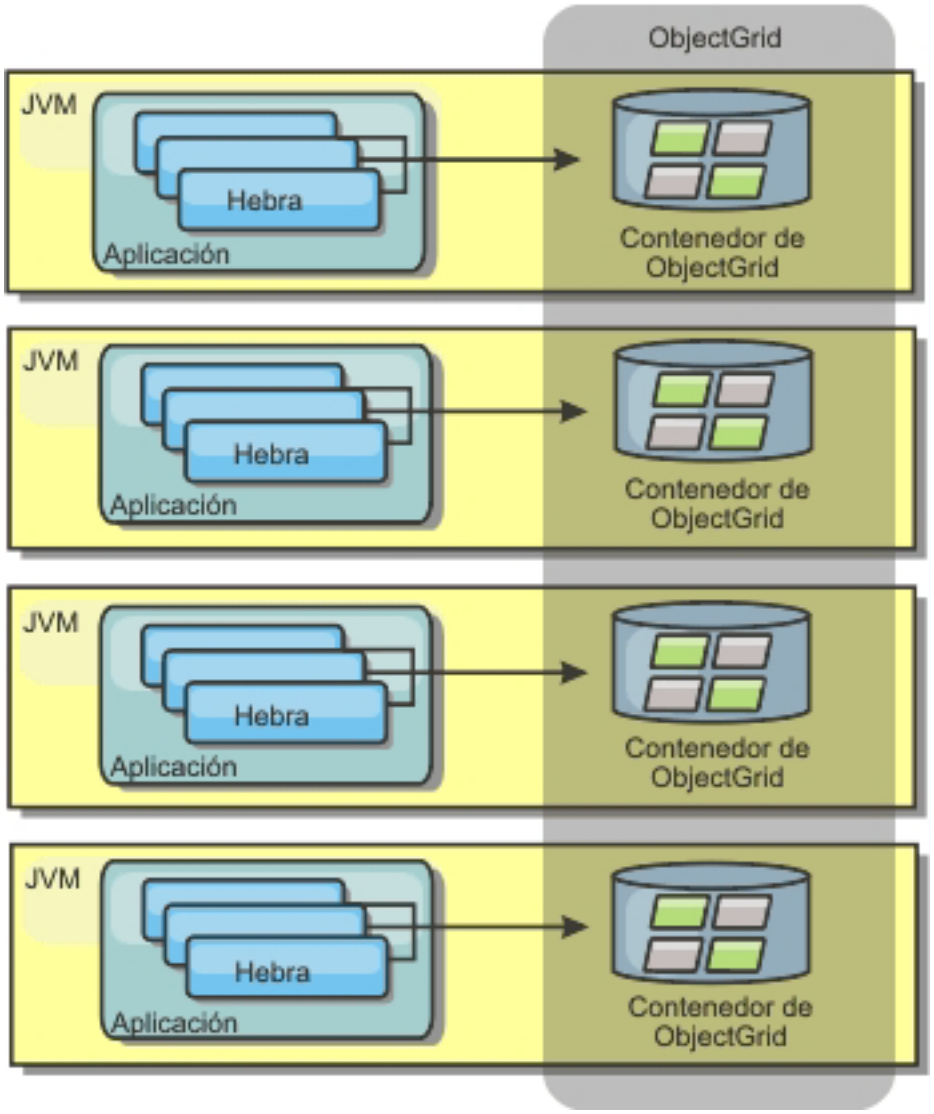


Figura 15. Memoria caché incorporada

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*.

Integración de la base de datos

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.

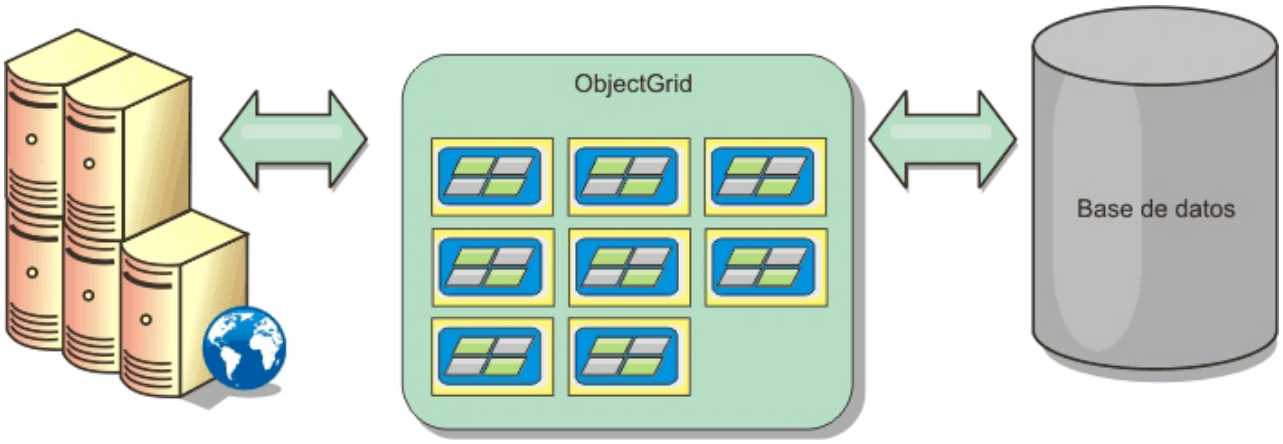


Figura 16. ObjectGrid como un almacenamiento intermedio de base de datos

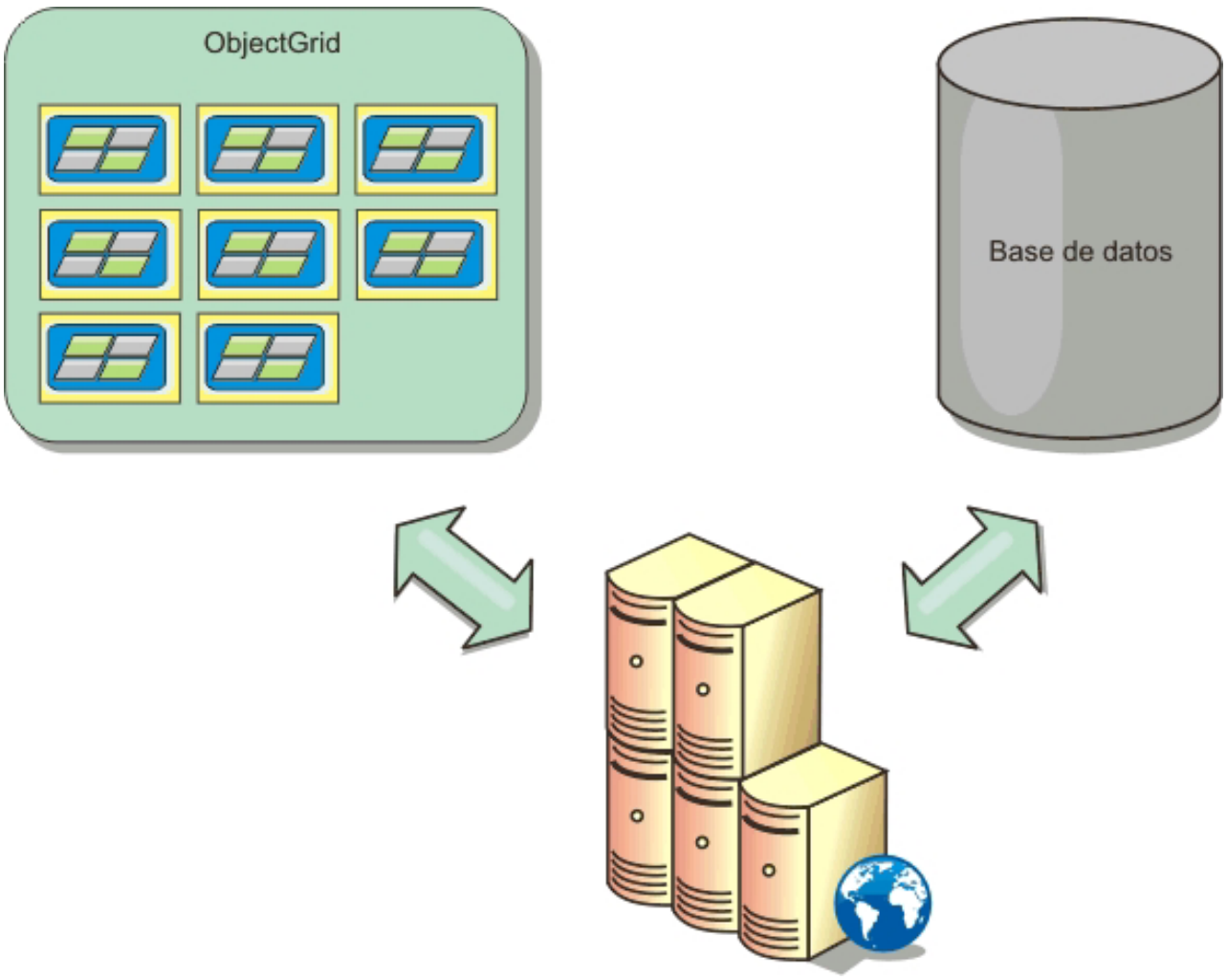


Figura 17. ObjectGrid como una memoria caché secundaria

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

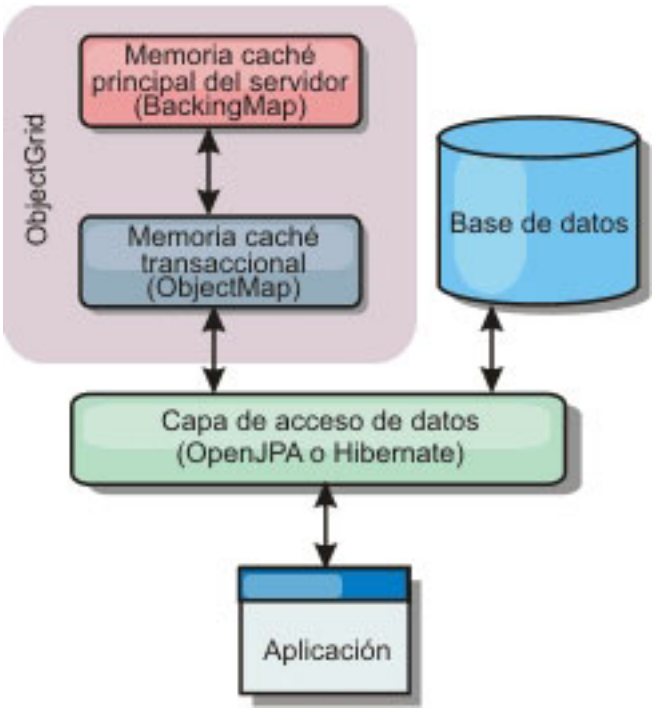


Figura 18. Memoria caché complementaria

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.

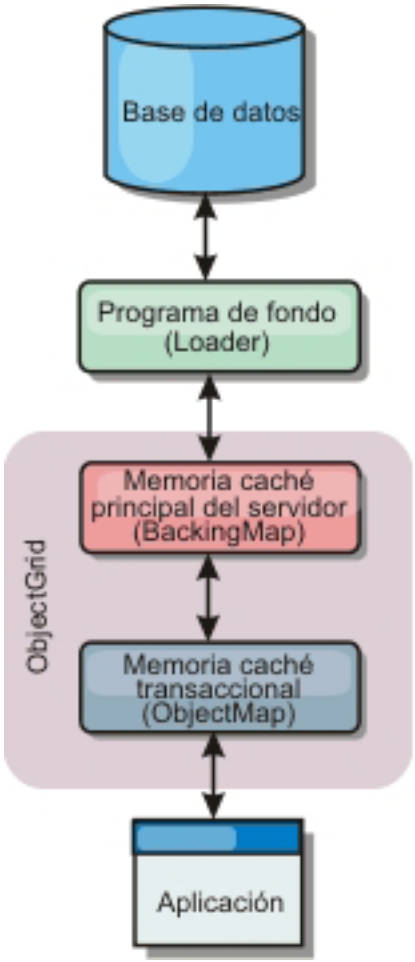


Figura 19. Memoria caché en línea

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.

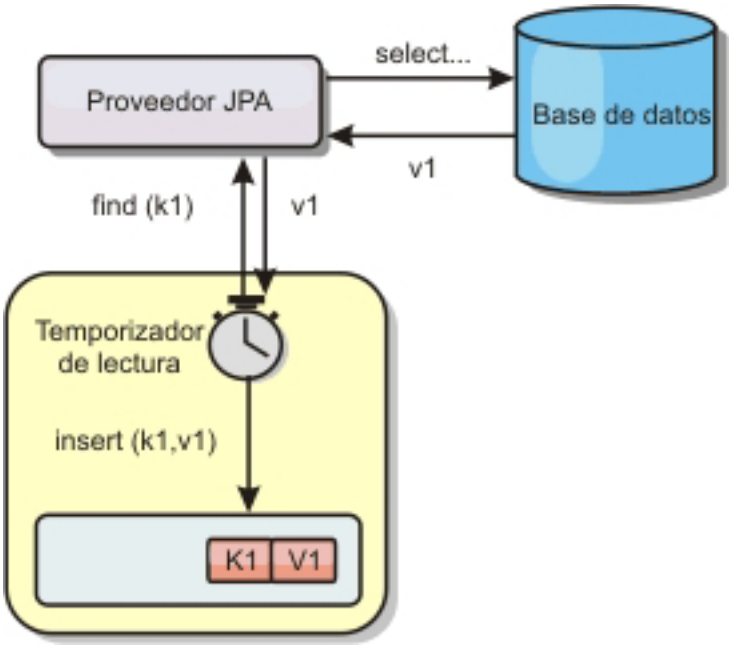


Figura 20. Renovación periódica

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. Consulte el tema “Desalojo” en la página 29 para obtener más detalles.

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.

Renovación periódica

Cuando se utiliza la capacidad de espacio de proceso de base de datos en memoria de WebSphere eXtreme Scale como una memoria caché, las aplicaciones se deben escribir para tolerar los datos obsoletos si la base de datos se puede actualizar de forma independiente de una transacción de eXtreme Scale. El uso de una renovación periódica es un método para que eXtreme Scale pueda mantener la memoria caché actualizada.

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.

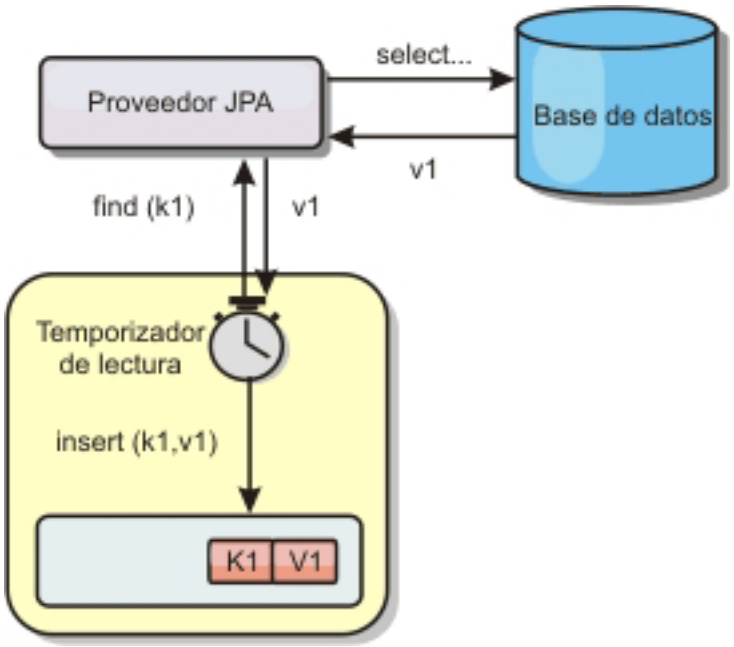


Figura 21. Renovación periódica

Las memorias caché escasas pueden utilizar políticas de desalojo para eliminar automáticamente datos de la memoria caché sin afectar a la base de datos. Las tres políticas incorporadas se incluyen en eXtreme Scale: tiempo de vida, menos usada recientemente y usada menos frecuentemente. 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. Si desea más información sobre los desalojadores, consulte “Desalojo”.

Desalojo

WebSphere eXtreme Scale proporciona un mecanismo predeterminado para desalojar entradas de memoria caché y un plug-in para crear desalojadores

personalizados. Un desalojador controla la pertenencia de las entradas en cada BackingMap. El desalojador predeterminado utiliza una política de desalojo de tiempo de vida (TTL) para cada BackingMap. Si proporciona un mecanismo de desalojador conectable, generalmente utiliza una política de desalojo basada en el número de entradas en lugar del tiempo.

Desalojador de tiempo de vida predeterminado

WebSphere eXtreme Scale proporciona un desalojo de tiempo de vida (TTL) para cada BackingMap. El desalojador TTL mantiene una fecha de caducidad para todas las entradas que se creen. Cuando a una entrada le llega la fecha de caducidad, el desalojador elimina la entrada de BackingMap. Para minimizar el impacto que tiene sobre el rendimiento la eliminación de una entrada, es posible que el desalojador TTL espere a desalojar una entrada después de la fecha de caducidad. El desalojador TTL nunca elimina una entrada antes de que caduque la entrada.

BackingMap tiene atributos que se utilizan para controlar cómo el desalojador de tiempo de vida calcula la hora de caducidad de todas las entradas. Las aplicaciones establecen el atributo ttlType para especificar cómo el desalojador TTL debe calcular el tiempo de caducidad. El atributo ttlType puede establecerse en uno de los siguientes valores:

1. Ninguno: indica que una entrada de la BackingMap nunca caduca. El desalojador TTL no desaloja estas entradas.
2. Hora de creación: indica que la hora en que se ha creado una entrada se utiliza en el cálculo de la hora de caducidad.
3. Hora del último acceso: indica que la hora a la que se ha accedido a una entrada por última vez se utiliza en el cálculo de la hora de caducidad.

Si el atributo ttlType no se establece en una BackingMap, se utiliza el tipo predeterminado Ninguno para que el desalojador TTL no desaloje ninguna entrada. Si el atributo ttlType se establece en la hora de creación o la hora de último acceso, el valor del atributo de tiempo de vida de BackingMap se añade a la hora de creación o a la hora del último acceso para calcular la hora de caducidad. La precisión de tiempo del atributo de correlación de tiempo de vida es en segundos. Un valor de 0 para el atributo de tiempo de vida es un valor especial que se utiliza para indicar que la entrada de correlación puede tener una duración permanente, es decir, la entrada permanece en la correlación hasta que la aplicación elimina o anula explícitamente la entrada de correlación.

Desalojadores opcionales

El desalojador TTL predeterminado utiliza una política de desalojo basada en la hora, y el número de entradas en BackingMap no tiene ningún efecto en la hora de caducidad de una entrada. Puede utilizar un desalojo conectable opcional para desalojar entradas basándose en el número de entradas que existían en lugar de basarse en la hora.

Los siguientes desalojadores conectables opcionales proporcionan algunos algoritmos utilizados habitualmente para decidir qué entradas se van a desalojar cuando una BackingMap aumenta de tamaño por encima de ciertos límites. *

- LRUEvictor es un desalojador que utiliza un algoritmo de menos usada recientemente (LRU) para decidir qué entradas se van a desalojar cuando la BackingMap exceda un número máximo de entradas.

- LFUEvictor es un desalojador que utiliza un algoritmo de usada menos frecuentemente (LFU) para decidir qué entradas se van a desalojar cuando la BackingMap exceda un número máximo de entradas.

BackingMap informa a un desalojador de la creación, modificación o eliminación de entradas en una transacción. BackingMap hace un seguimiento de estas entradas y selecciona cuándo se va a desalojar una o más entradas de la BackingMap.

Una BackingMap no tiene información de configuración para un tamaño máximo. Por el contrario, las propiedades de desalojador se establecen para controlar el comportamiento del desalojador. Tanto LRUEvictor como LFUEvictor tienen una propiedad de tamaño máximo que se utiliza para que el desalojador empiece a desalojar entradas después de que se supere el tamaño máximo. Como el desalojador TTL, es posible que los desalojadores LRU y LFU no desalojen inmediatamente una entrada cuando se alcance el número máximo de entradas para minimizar el impacto en el rendimiento.

Si los algoritmos de desalojo LRU o LFU no son adecuados para una determinada aplicación, puede escribir sus propios desalojadores para crear la estrategia de desalojo.

Desalojo basado en memoria

Importante: El desalojo basado en memoria sólo está soportado en Java Platform, Enterprise Edition versión 5 o posterior.

Todos los desalojadores incorporados dan soporte al desalojo basado en memoria que se puede habilitar en la interfaz BackingMap estableciendo el atributo evictionTriggers de BackingMap en `MEMORY_USAGE_THRESHOLD`. Si desea más información sobre cómo establecer el atributo evictionTriggers en BackingMap, consulte la información sobre la interfaz BackingMap y el archivo XML de descriptor de ObjectGrid en *Guía de administración*.

El desalojo basado en memoria se basa en el umbral de uso del almacenamiento dinámico. Cuando el desalojo basado en memoria está habilitado en BackingMap y BackingMap tiene cualquier desalojo incorporado, el umbral de uso se establece en un porcentaje predeterminado de la memoria total si el umbral no se ha establecido previamente.

Cuando utilice el desalojo basado en memoria, deberá configurar el umbral de recogida de basura en el mismo valor que su uso del almacenamiento dinámico de destino. Por ejemplo, el umbral de desalojo basado en memoria se establece en el 50 por ciento y el umbral de recogida de basura está al 70 por ciento del nivel predeterminado, el uso del almacenamiento dinámico puede llegar hasta el 70 por ciento. Este aumento del uso del almacenamiento dinámico se produce porque el desalojo basado en memoria sólo se desencadena después de un ciclo de recogida de basura.

El algoritmo de desalojo basado en memoria utilizado por WebSphere eXtreme Scale es sensible al comportamiento del algoritmo de recogida de basura que se utiliza. El mejor algoritmo para el desalojo basado en memoria es el recopilador de rendimiento predeterminado de IBM. Los algoritmos de recogida de basura de generación puede provocar un comportamiento no deseado y, por lo tanto, no debe utilizar estos algoritmos con el desalojo basado en memoria.

Para cambiar el porcentaje del umbral de uso, establezca la propiedad memoryThresholdPercentage en los archivos de propiedad de contenedor y servidor para los procesos de servidor de eXtreme Scale.

Durante la ejecución, si el uso de memoria excede el umbral del uso de destino, los desalojadores basados en memoria empiezan a desalojar entradas e intentan mantener el uso de la memoria por debajo del umbral de uso de destino. Sin embargo, no existe ninguna garantía de que la velocidad del desalojo sea lo suficientemente rápida para evitar un posible error de memoria, si el tiempo de ejecución del sistema sigue consumiendo memoria rápidamente.

Procedimientos recomendados para el desalojador predeterminado:

El comportamiento del desalojador de tiempo de vida (TTL) predeterminado puede modificarse mediante el establecimiento de los atributos y las propiedades.

Además de los desalojadores de plug-in descritos en el tema sobre los procedimientos recomendados para optimizar el rendimiento del desalojador de plug-in, se crea un desalojador TTL predeterminado con cada correlación de respaldo. El desalojador predeterminado elimina las entradas en función de un concepto de tiempo de vida. Este comportamiento se define con el atributo ttlType. Existen tres atributos ttlType:

- Ninguno (NONE): especifica que las entradas nunca caducan y, por lo tanto, nunca se eliminan de la correlación.
- Tiempo de creación (CREATION_TIME): especifica que las entradas se desalojan en función de cuándo se crearon.
- Último acceso (LAST_ACCESSED_TIME): especifica que las entradas se desalojan en función de la última vez que se accedió a ellas.

Propiedad TimeToLive

Esta propiedad, junto con la propiedad ttlType, es la más importante desde una perspectiva de rendimiento. Si utiliza el atributo ttlType de tipo CREATION_TIME, el desalojador desaloja una entrada cuando el tiempo de creación es igual al valor del atributo TimeToLive. Si se estableció el valor del atributo TimeToLive en 10 segundos, todos los elementos de la correlación se desalojan después de transcurridos 10 segundos. Es importante establecer con cuidado este valor de CREATION_TIME. El desalojador funciona mejor cuando existen cantidades razonablemente altas de adiciones a la memoria caché que sólo se usan durante un tiempo establecido. Con esta estrategia, todo lo que se crea se eliminará después de transcurrido un tiempo establecido.

A continuación se muestra un ejemplo de un caso en el que es útil el tipo TTL de CREATION_TIME. Suponga que utiliza una aplicación web que obtiene cotizaciones de bolsa. La obtención de las cotizaciones más recientes no es clave. En este caso, las cotizaciones de bolsa se almacenan en la memoria caché en ObjectGrid durante 20 minutos. Transcurridos los 20 minutos, las entradas de la correlación ObjectGrid caducan y se desalojan. Cada 20 minutos aproximadamente, la correlación ObjectGrid utiliza el plug-in Loader para renovar los datos de la correlación con datos renovados de la base de datos. La base de datos se actualiza cada 20 minutos con las cotizaciones de bolsa más recientes. Por lo tanto, para esta aplicación, el uso de un valor TimeToLive de 20 minutos es ideal.

Si utiliza el atributo ttlType de tipo LAST_ACCESSED_TIME, establezca el valor de TimeToLive en un número más bajo que si estuviera utilizando CREATION_TIME,

porque el valor TimeToLive de las entradas se restablece cada vez que se obtiene acceso. En otras palabras, si el atributo TimeToLive es igual a 15 y una entrada ha existido 14 segundos y, acto seguido, se accede a ella, no caduca hasta transcurridos otros 15 segundos. Si establece TimeToLive en un número relativamente alto, puede que muchas entradas nunca se desalojen. No obstante, si establece el valor en un número aproximado a 15 segundos, las entradas se eliminarán si no se accede a ellas con mucha frecuencia.

A continuación se muestra un ejemplo de un caso en el que es útil el tipo TTL de LAST_ACCESSED_TIME. Se utiliza una correlación ObjectGrid para contener datos de sesión de un cliente. Los datos de sesión deben destruirse si el cliente no utiliza los datos de sesión durante un período de tiempo. Por ejemplo, los datos de sesión exceden el tiempo de espera si después de 30 minutos no se produce actividad en el cliente. En este caso, el uso de un tipo de TTL de LAST_ACCESSED_TIME con el atributo TimeToLive establecido en 30 minutos es exactamente lo que se necesita para esta aplicación.

En el ejemplo siguiente se crea una correlación de respaldo, se establece el atributo ttlType del desalojador predeterminado y se establece la propiedad TimeToLive.

```
ObjectGrid objGrid = new ObjectGrid;
BackingMap bMap = objGrid.defineMap("SomeMap");
bMap.setTtlEvictorType(TTLType.LAST_ACCESSED_TIME);
bMap.setTimeToLive(1800);
```

La mayoría de los valores de configuración de los desalojadores deben establecerse antes de inicializar ObjectGrid.

También puede escribir sus propios desalojadores: si desea más información, consulte la información sobre cómo escribir un desalojador personalizado en *Guía de programación*.

Casos de ejemplo de 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.

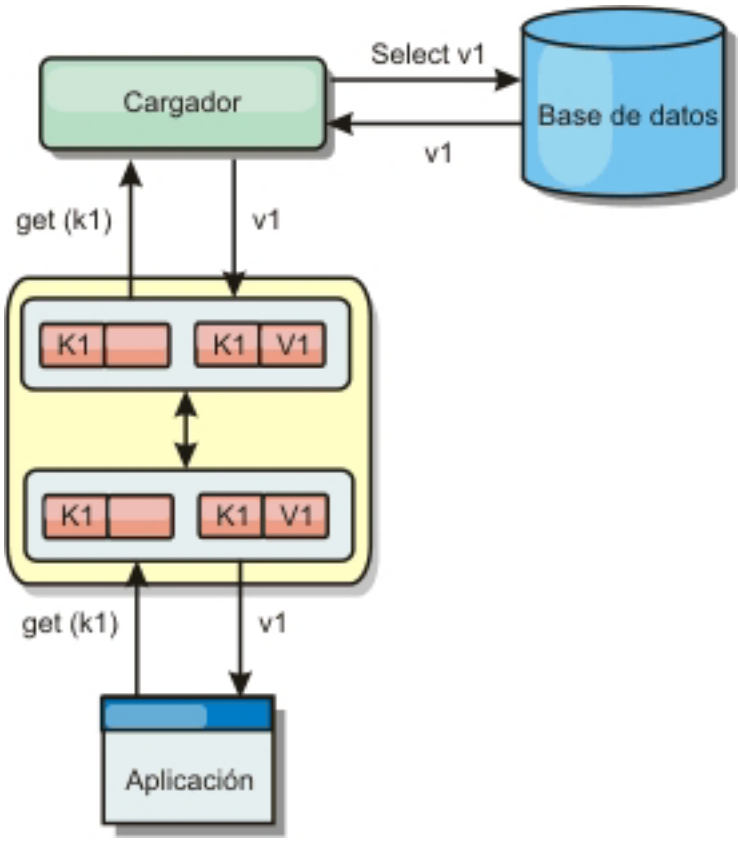


Figura 22. Almacenamiento en memoria caché de lectura directa

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.

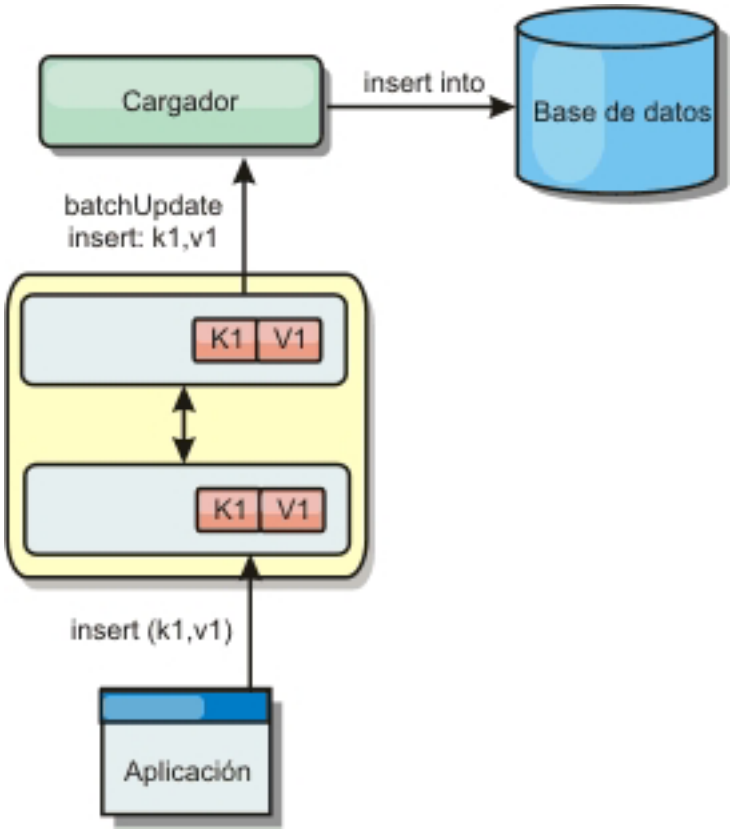


Figura 23. Almacenamiento en memoria caché de grabación directa

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 37.

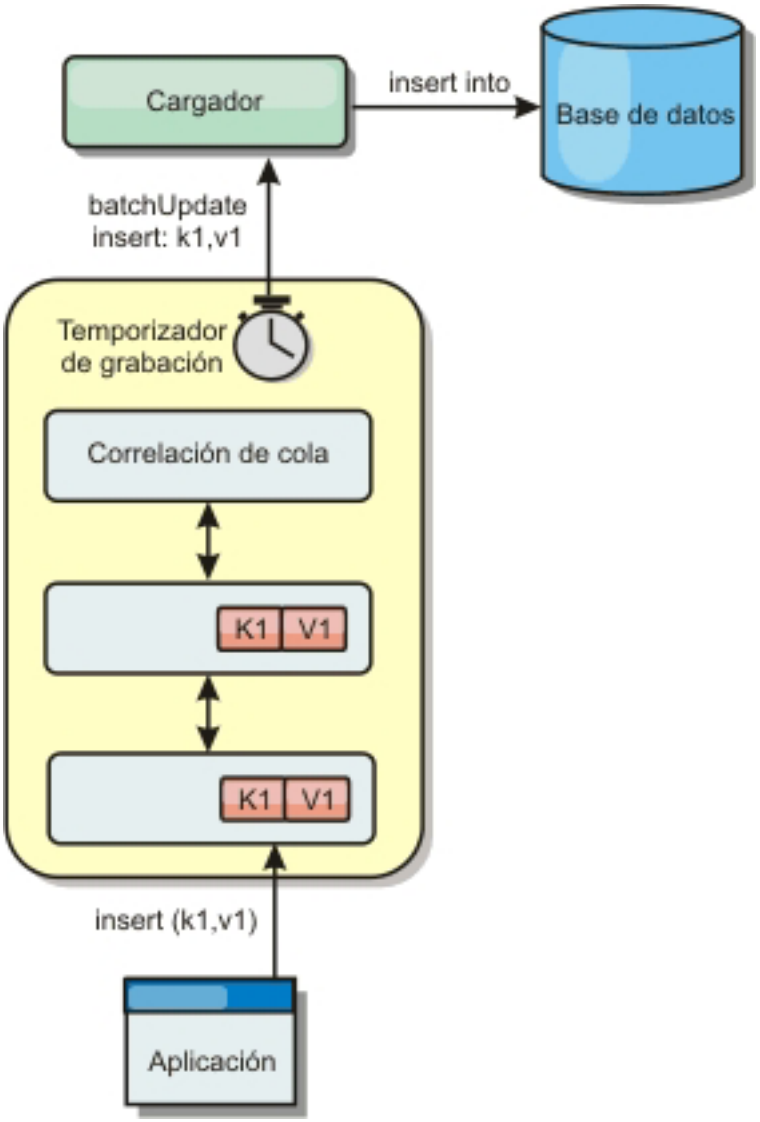


Figura 24. Almacenamiento en memoria caché de grabación anticipada

Consulte "Almacenamiento en memoria caché de grabación anticipada" en la página 37 si desea información adicional.

Cargadores

Un Cargador es un plug-in que funciona como un enlace entre un BackingMap y un programa de fondo como, por ejemplo, una base de datos.

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.

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.

Si desea más información sobre cargadores, consulte la información sobre cargadores en *Guía de administración*.

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 de 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).

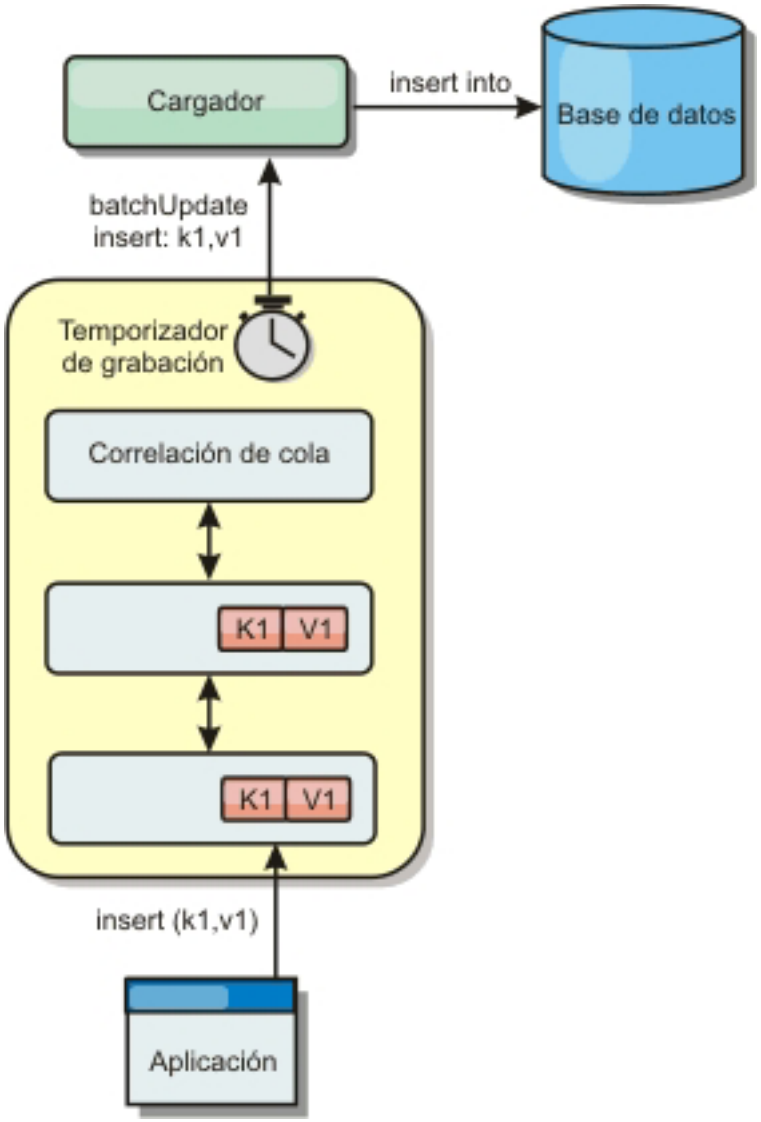


Figura 25. Almacenamiento en memoria caché de grabación anticipada

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.

Precarga de datos y calentamiento

En muchos escenarios, puede sacar partido de la precarga de la cuadrícula con datos.

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.

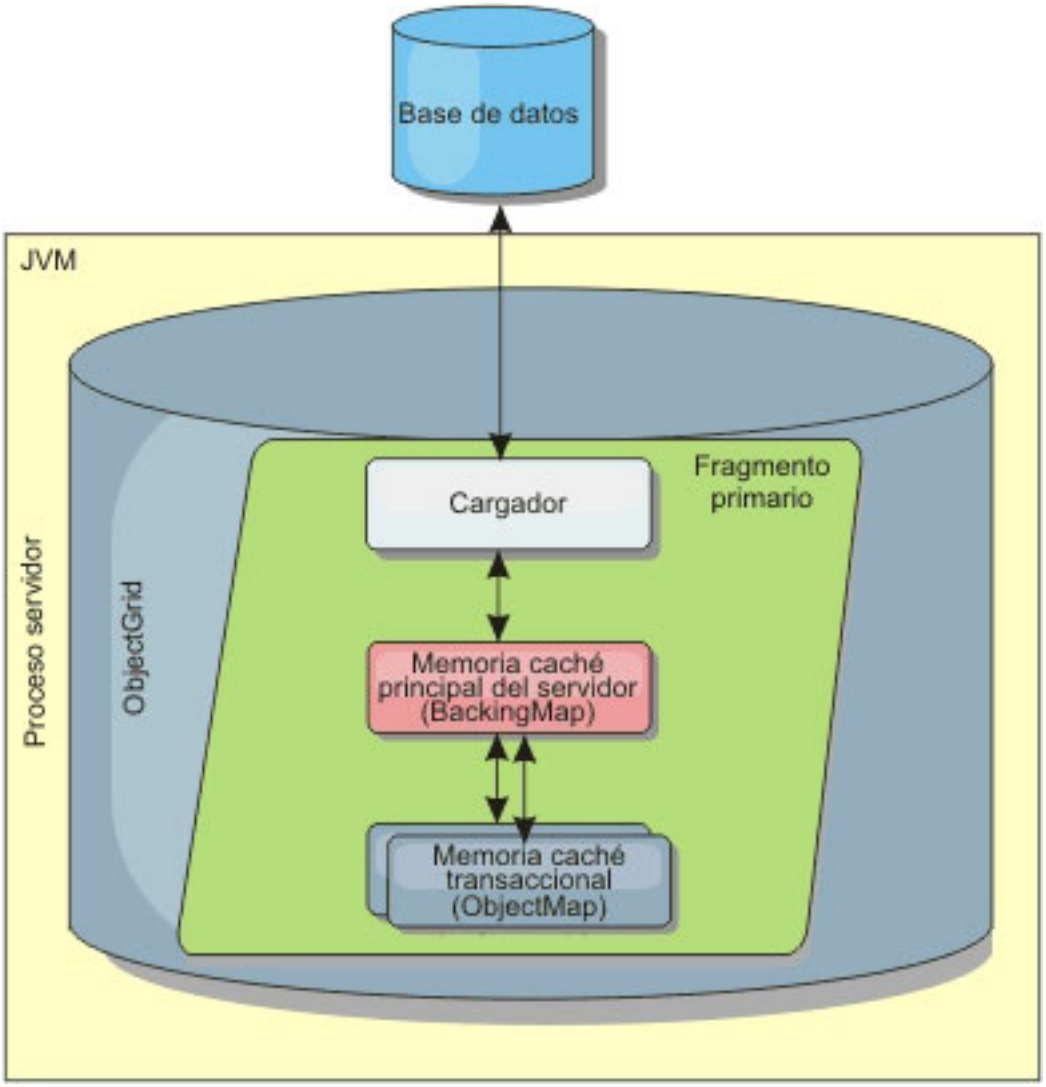


Figura 26. Plug-in Loader

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.

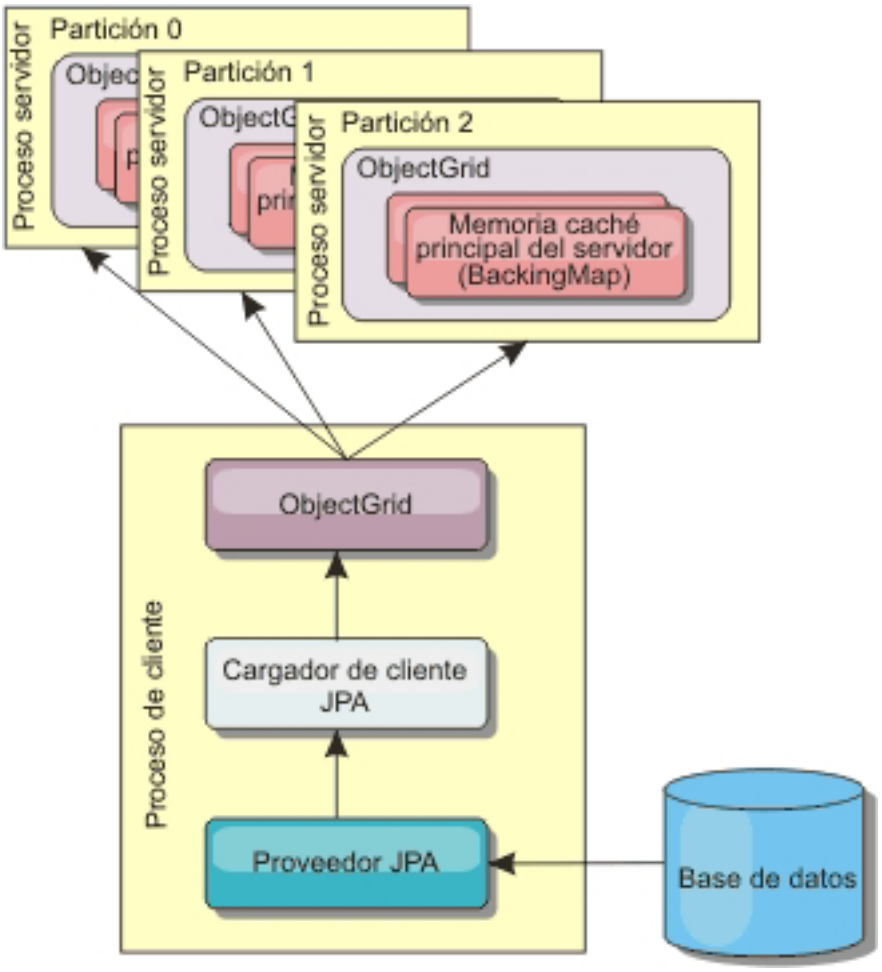


Figura 27. Cargador de clientes

Conceptos de la memoria caché de objetos Java

Básicamente, WebSphere eXtreme Scale se utiliza como una cuadrícula de datos y como memoria caché para los objetos Java. Hay varias API que se pueden utilizar 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 y topología” en la página 9 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, las definiciones de clase deben existir en la classpath, 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 los datos. Los datos, que tienen el formato de las instancias de objeto Java, se convierten 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

Puede asociarse un ObjectTransformer con 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.

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 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.

Capítulo 3. Integración de la memoria caché

WebSphere eXtreme Scale se puede integrar con otros productos relacionados con la memoria caché. JPA se puede utilizar entre WebSphere eXtreme Scale y la base de datos para integrar los cambios como un cargador. También puede utilizar el proveedor de memoria caché dinámica de WebSphere eXtreme Scale para conectar WebSphere eXtreme Scale en el componente de la memoria caché dinámica en WebSphere Application Server. Otra ampliación para WebSphere Application Server es el gestor de sesiones HTTP de WebSphere eXtreme Scale, que puede ayudar a colocar en la memoria caché las sesiones HTTP.

Utilización de eXtreme Scale con 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. Existe un número de implementaciones de código abierto y comerciales.

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.

Visión general de cargadores JPA

Un cargador Java Persistence API (JPA) es una implementación de plug-in de cargador que utiliza JPA para interactuar con la base de datos.

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).

© Copyright IBM Corp. 2009, 2009

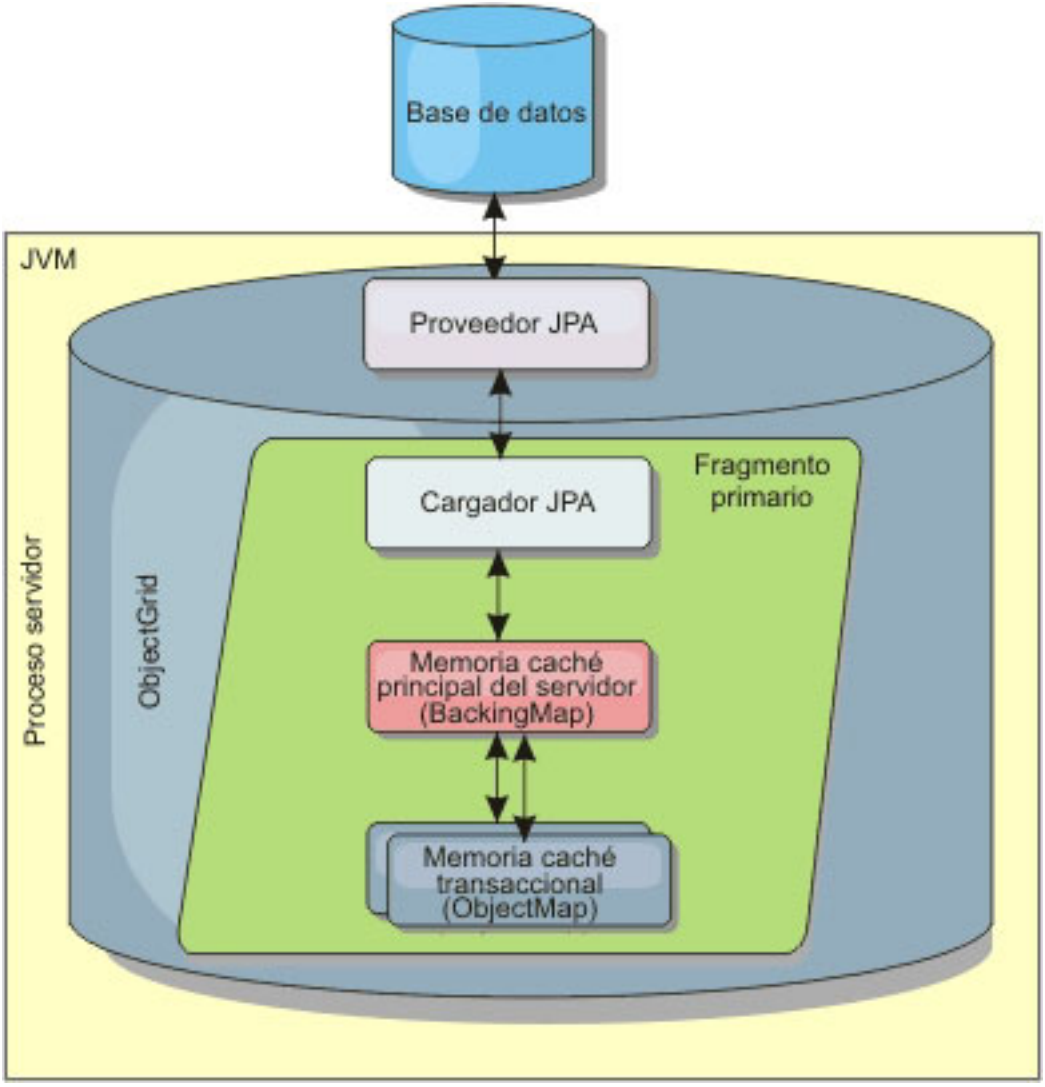


Figura 28. Arquitectura de cargador JPA

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 detalles, consulte Plug-in JPAEntityLoader .

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.

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.

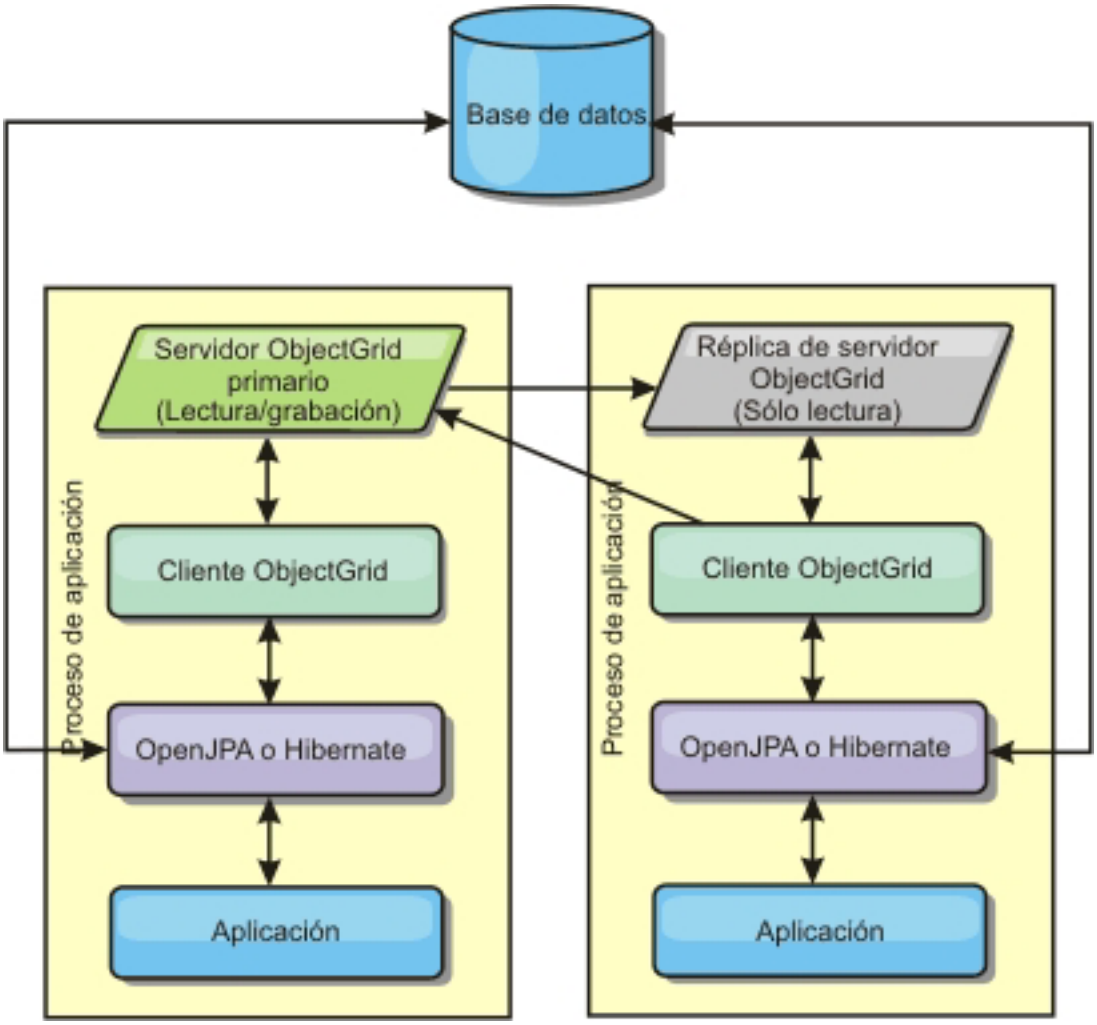


Figura 29. Topología incorporada JPA

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.

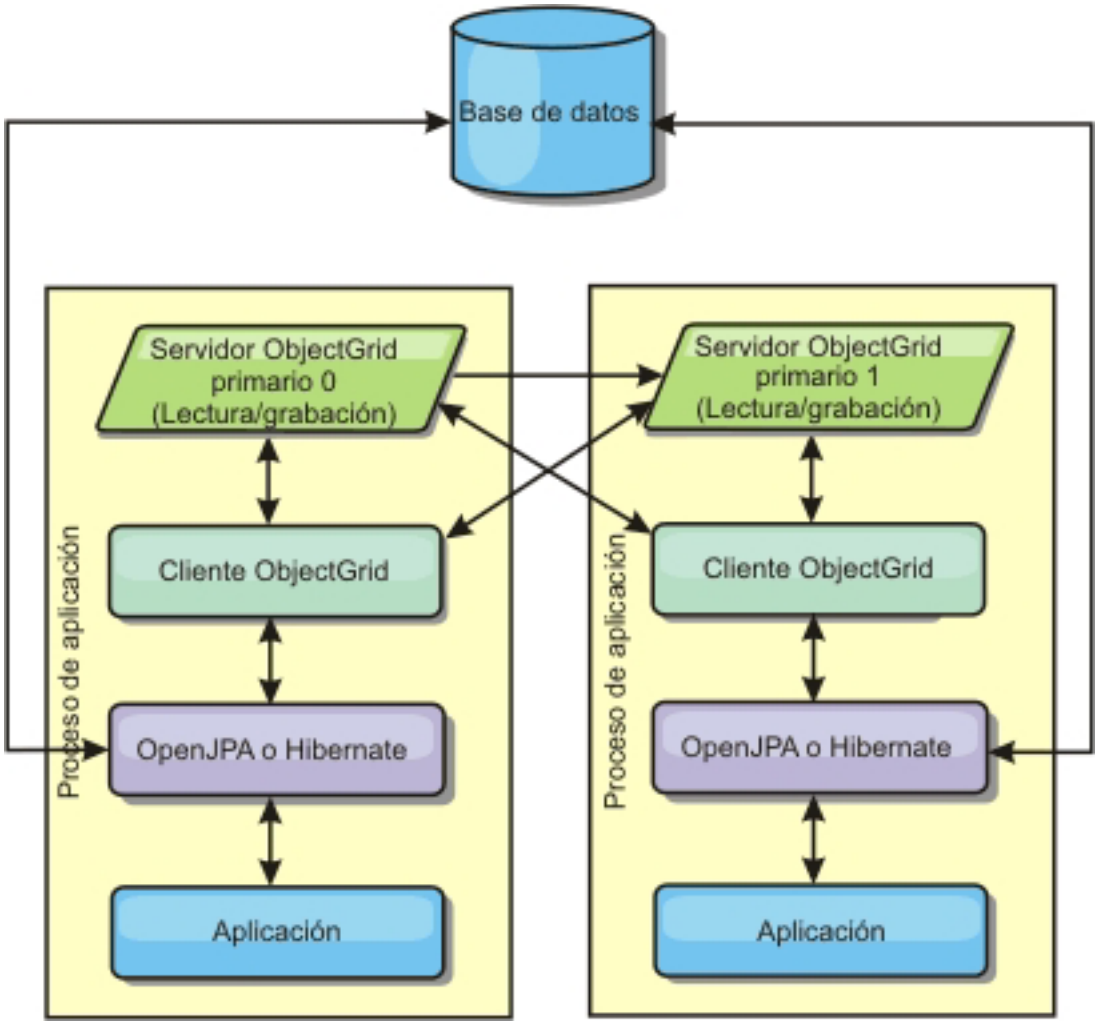


Figura 30. Topología incorporada con particiones JPA

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.

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 configurar eXtreme Scale que se particione y duplique. Una configuración remota es gestionada independientemente de la aplicación y del proveedor JPA.

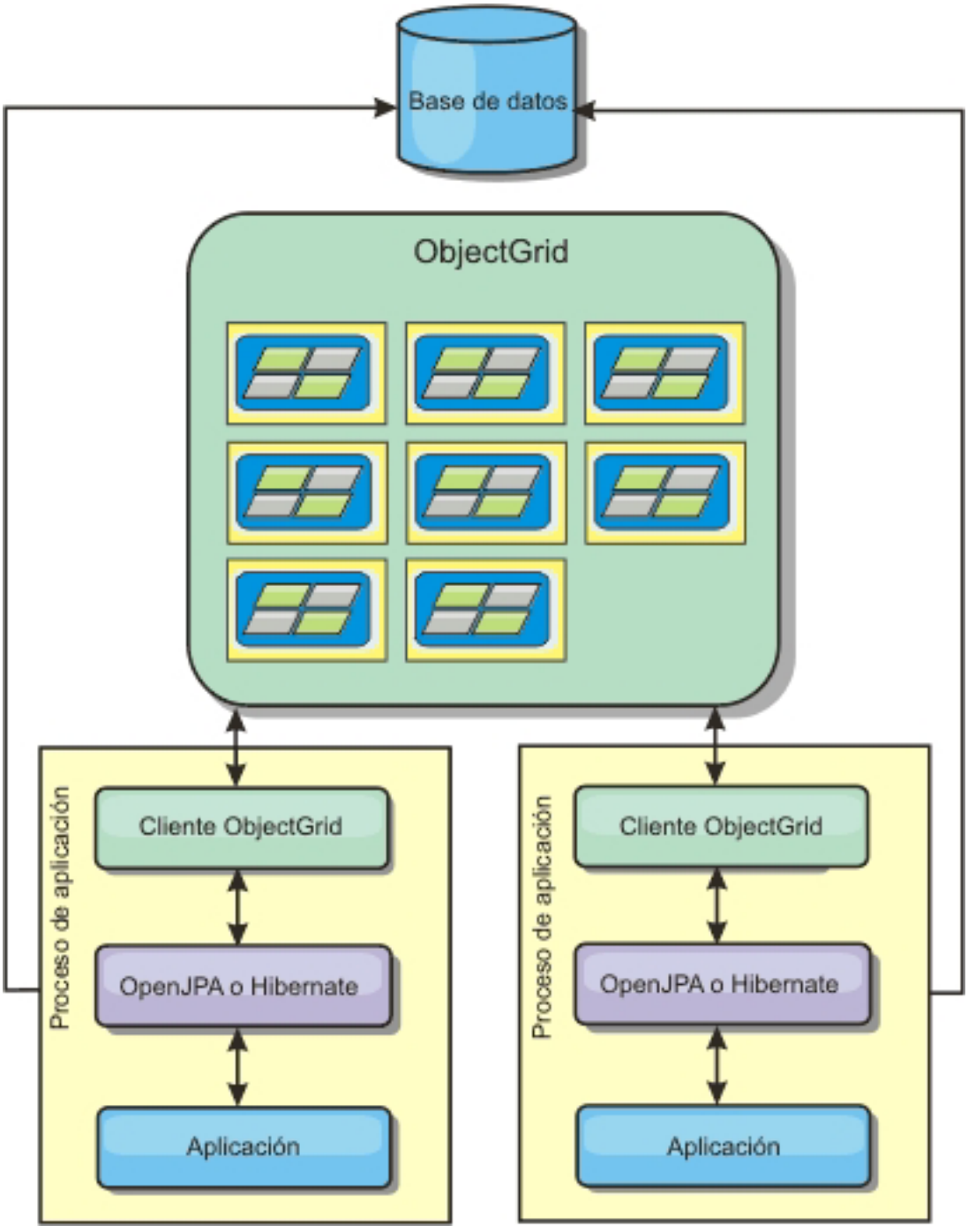


Figura 31. Topología remota JPA

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.

Gestión de sesiones HTTP

En un entorno de WebSphere Application Server versión 5 o posterior, el gestor de sesiones HTTP que se distribuye con WebSphere eXtreme Scale puede alterar temporalmente el gestor de sesiones predeterminado del servidor de aplicaciones.

El gestor de sesiones HTTP de WebSphere eXtreme Scale también puede ejecutarse en WebSphere Application Server versión 6.0.2 o posterior, o en un entorno que no ejecute WebSphere Application Server, como por ejemplo WebSphere Application Server Community Edition o Apache Tomcat.

Características

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 gestión de sesiones para una aplicación asociada. El gestor de sesiones crea sesiones HTTP y gestiona los ciclos de vida de dichas sesiones 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.

Cómo funciona el gestor de sesiones

El gestor de sesiones se inserta en la vía de acceso de la solicitud con el formato de un filtro de servlet. Puede añadir este 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 filtros de forma manual al descriptor de despliegue web de la aplicación. Este filtro recibe la solicitud antes que los archivos JSP o servlet en la aplicación de destino. En este momento, el filtro intercepta los objetos HTTPRequest y HTTPResponse y crea un objeto que los envuelve con su propia implementación.

La implementación de este filtro intercepta todas las llamadas relacionadas con la sesión HTTP que se realizan en los objetos HTTPRequest y HTTPResponse. El gestor de sesiones se encarga de estas llamadas, que no se pasan al gestor de sesiones base en la implementación del servidor Java Platform, Enterprise Edition subyacente. El gestor de sesiones crea y mantiene sesiones y los ciclos de vida de éstas, incluidas las invalidaciones basadas en el tiempo de espera y la activación de escuchas en las invalidaciones de sesiones y otros sucesos de ciclo de vida.

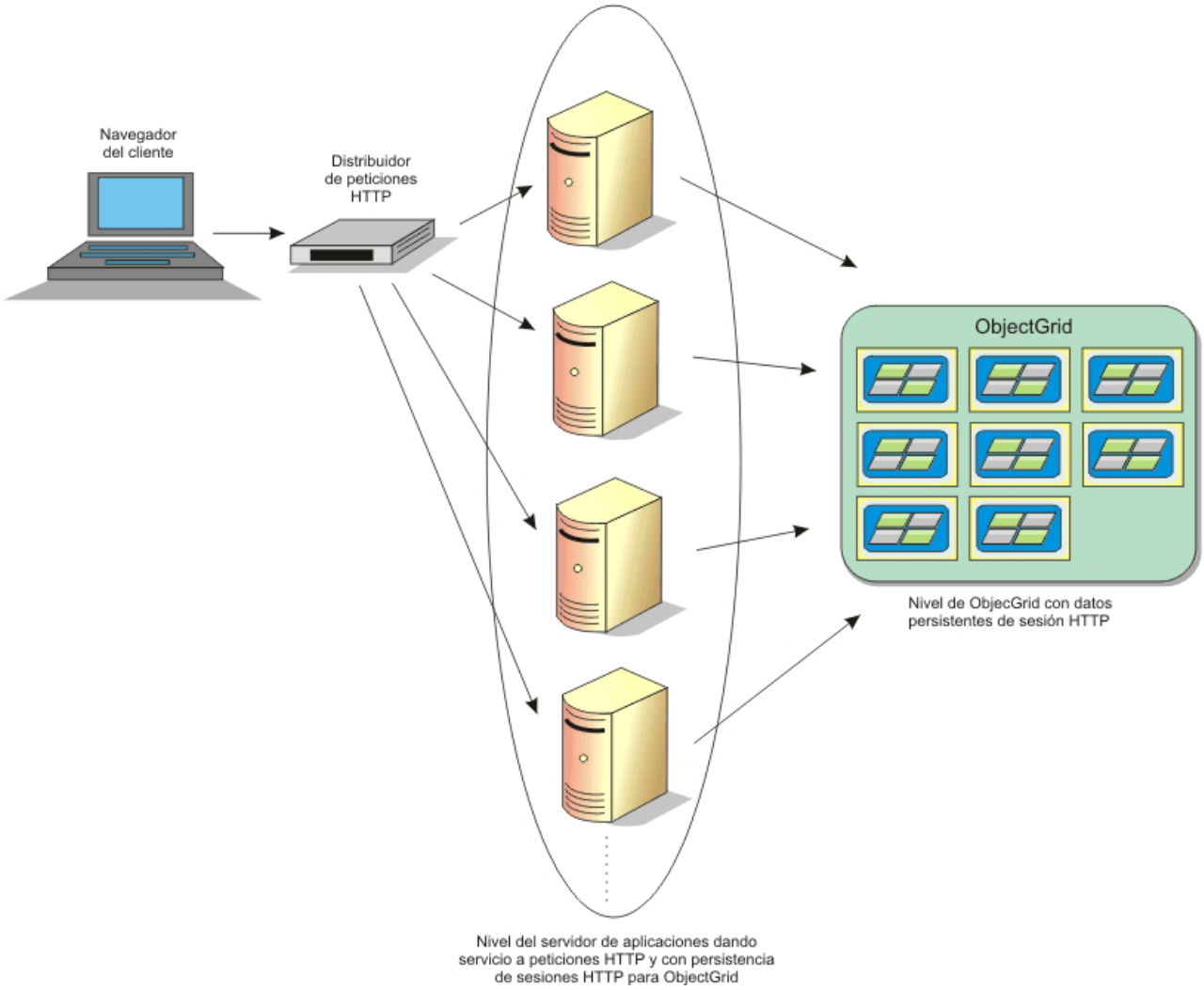


Figura 32. Topología de gestión de sesiones HTTP con una configuración de contenedor remoto

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.
- **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 un eXtreme Scale remoto.

Recuperación de una sesión de eXtreme Scale en una aplicación de gestor de sesiones

```
public void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {

        HttpSession session = req.getSession(true);

        System.out.println("calling getSession");
```

```
    // llamar getAttribute("com.ibm.websphere.objectgrid.session")
// para obtener la instancia de la sesión ObjectGrid
    Session ogSession = (Session)session.getAttribute
  ("com.ibm.websphere.objectgrid.session");

    System.out.println("ogSession = "+ogSession);
}
```

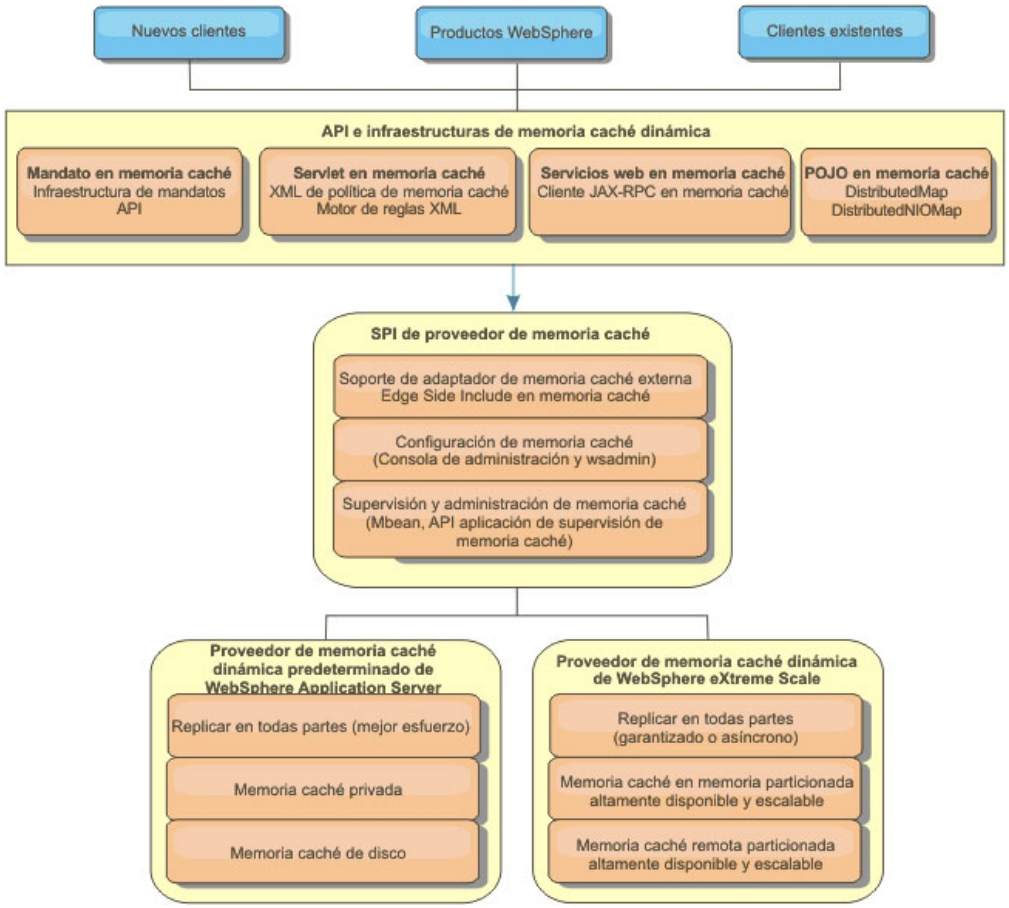
Los cambios realizados en las correlaciones con la sesión que se devuelve de la llamada al método getAttribute se confirman cuando se confirma la sesión subyacente.

Proveedor de la memoria caché dinámica WebSphere eXtreme Scale

La API de memoria caché dinámica está disponible para las aplicaciones Java EE 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

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.



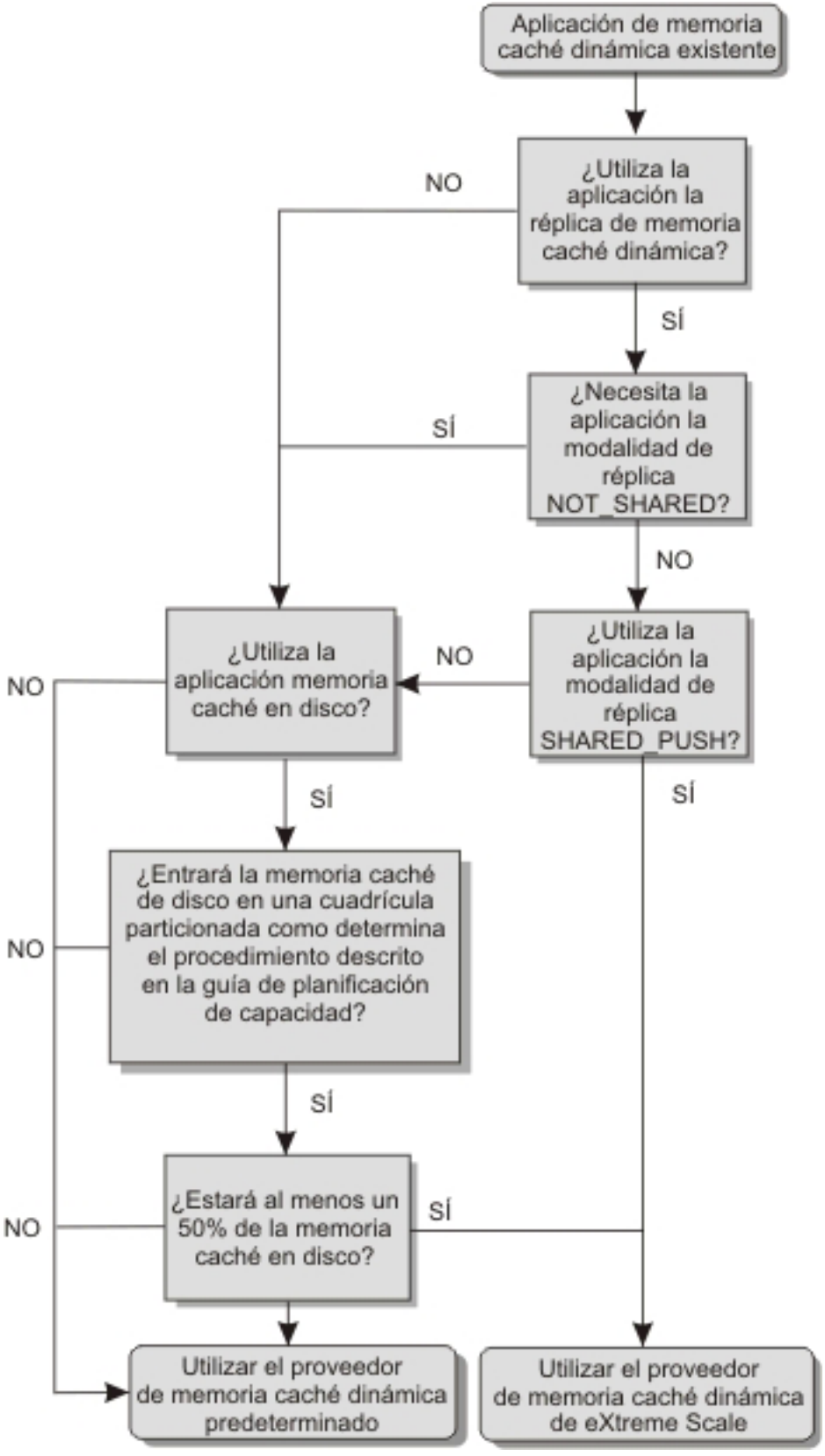
Puede instalar y configurar el proveedor de memoria caché dinámica descrito en "Configuración del proveedor de memoria caché dinámica para WebSphere eXtreme Scale" en la página 72

Decidir cómo sacar partido de WebSphere eXtreme Scale

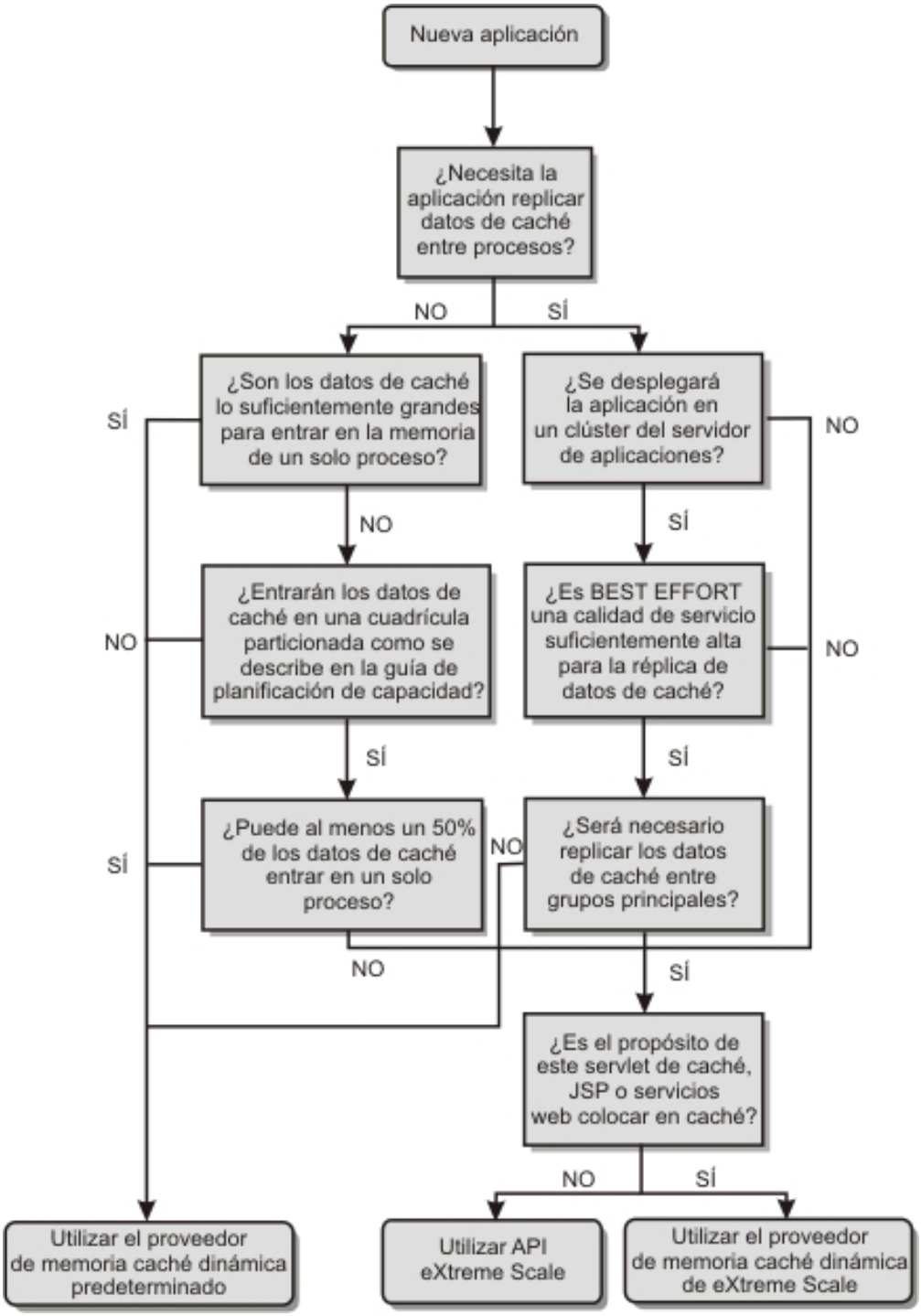
Las características disponibles en WebSphere eXtreme Scale aumentan de forma significativa las capacidades distribuidas de la API de memoria caché dinámica más allá de lo que ofrece el motor de memoria caché dinámica predeterminado y el servicio de duplicación de datos. Con eXtreme Scale, puede crear 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 el proveedor de memoria caché para las nuevas aplicaciones.



Comparación de características

Tabla 3. 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
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 4. 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+	

Tabla 4. Integración de tecnología sin fisuras (continuación)

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
Sincronización de la base de datos utilizando OpenJPA e Hibernate			x

Tabla 5. 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 eXtreme Scale distribuido coloca en la memoria caché el trabajo, consulte ″Configuraciones de despliegue para eXtreme Scale″ en la *Guía de programación*.

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

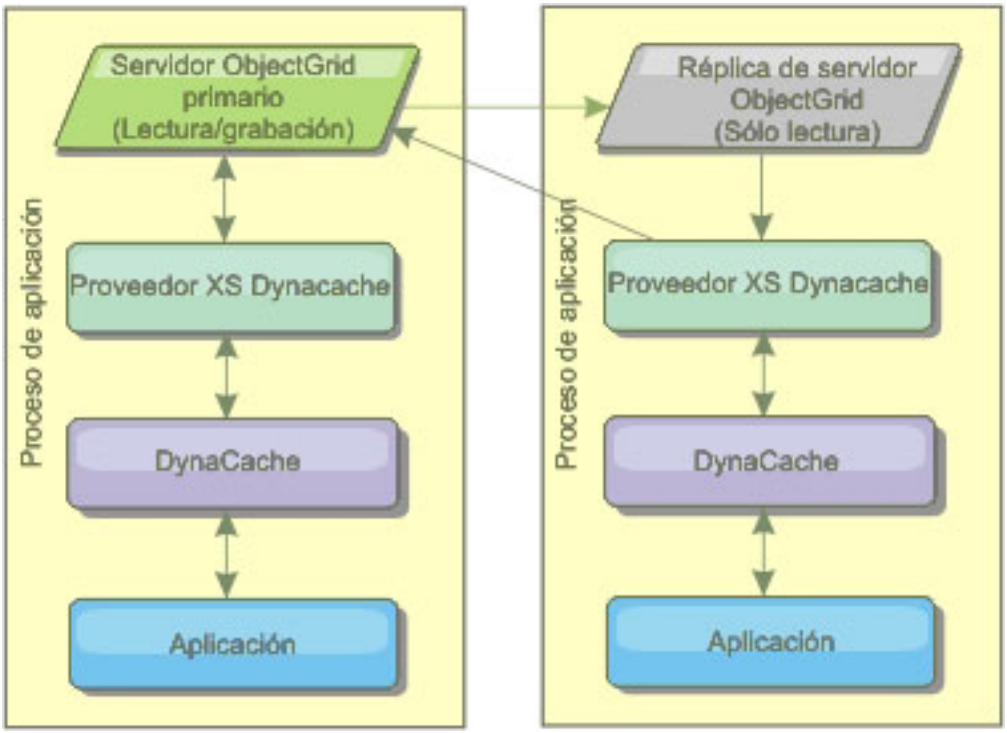
Un servicio de memoria caché dinámica creado con el proveedor eXtreme Scale se puede desplegar en cualquiera de las tres topologías disponibles, lo que le permite adaptar la memoria caché específicamente al rendimiento, los recursos y las necesidades administrativas. Estas topologías son: incorporado,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

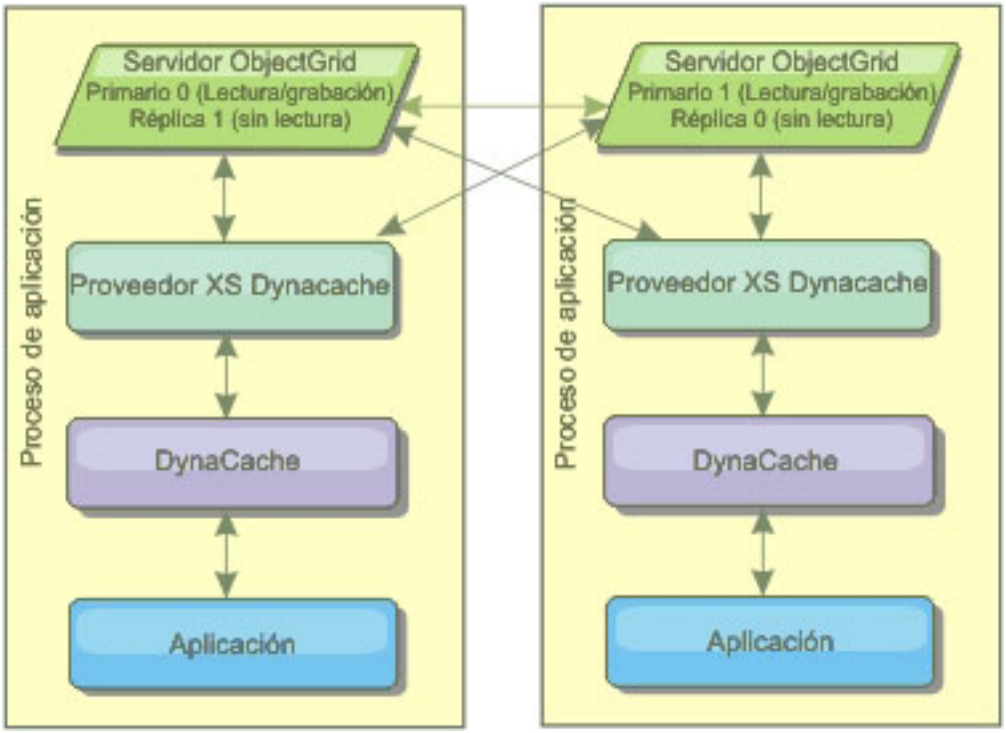
Con la topología incorporada, las entradas de memoria caché nuevas o actualizadas no son visibles de forma inmediata en cada proceso de servidor único. Una entrada de caché no será visible, incluso para el servidor que lo ha generado, hasta que se propague a través de los servicios de duplicación asíncrona 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:

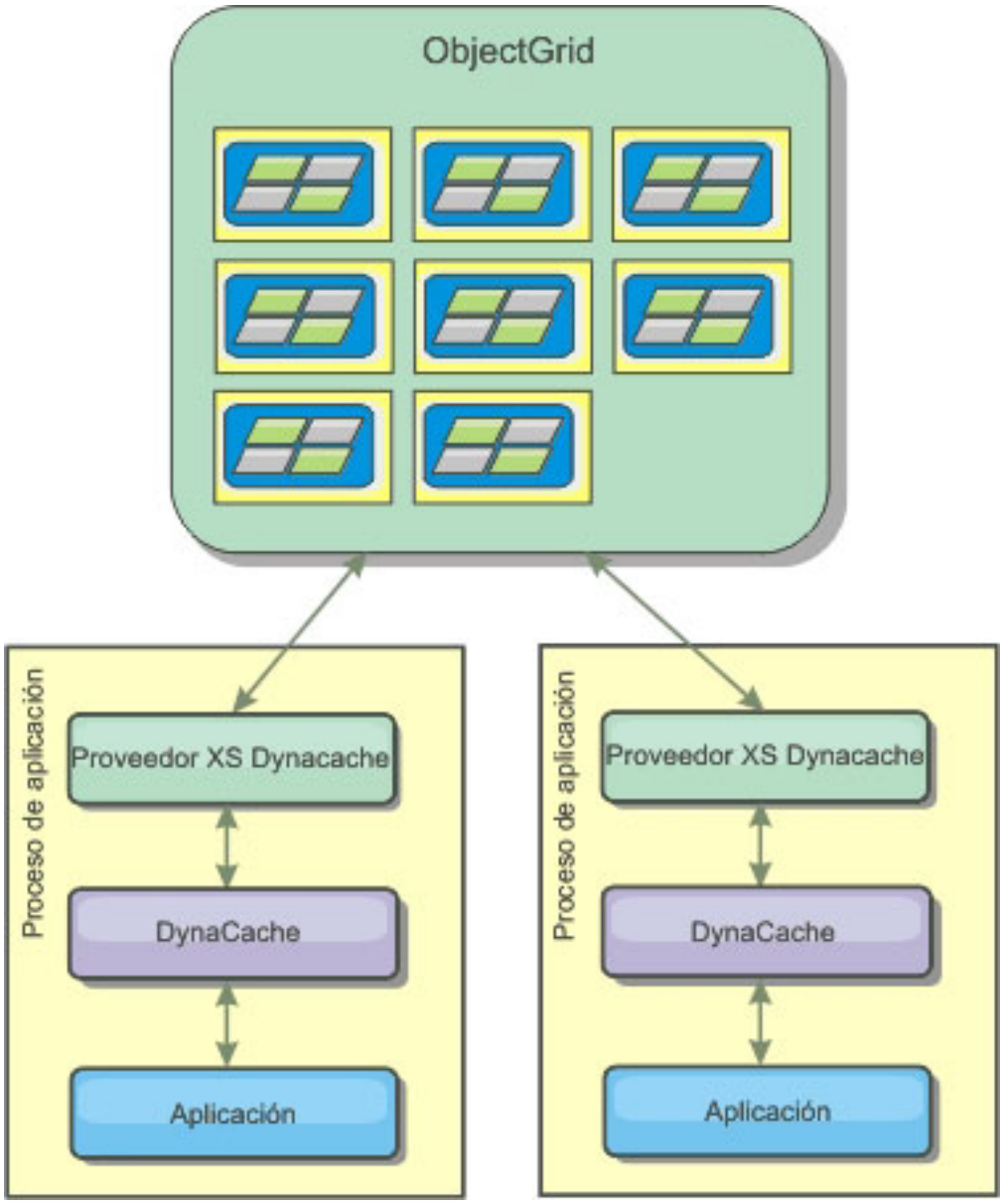


Por ejemplo, suponga que tiene una cuadrícula de procesos de servidor con 256 megabytes de almacenamiento dinámico libre para alojar el servicio de memoria caché dinámica. El proveedor de la memoria caché dinámica predeterminada y el proveedor eXtreme Scale que utiliza la topología incorporada se deben 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 también se puede utilizar para recuperar las instancias de memoria caché dinámica que tienen la **réplica inhabilitada**. Al igual que el proveedor de memoria caché dinámica predeterminada, 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 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 no está soportada actualmente y siempre devolverá -1.
- **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 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 seguridad 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

Configuración del proveedor de memoria caché dinámica para WebSphere eXtreme Scale

La instalación y configuración del proveedor de memoria caché dinámica para eXtreme Scale depende de sus requisitos y del entorno que ha configurado.

Antes de empezar

Instale el proveedor de memoria caché dinámica.

Para utilizar el proveedor de memoria caché dinámica, WebSphere eXtreme Scale debe estar instalado encima de los despliegues del nodo de WebSphere Application Server, incluido el nodo del gestor de despliegue. El proveedor de memoria caché dinámica de eXtreme Scale está soportado en las siguientes versiones de WebSphere Application Server.

- WebSphere Application Server 6.1.0.25 + PK85622 y posterior
- WebSphere Application Server 7.0.0.3 + PK85622 y posterior

Si desea instrucciones de instalación, consulte Instalación de 6.1 o Instalación de 7.0.

Por qué y cuándo se efectúa esta tarea

Siga estos pasos para configurar el proveedor de memoria caché dinámica de eXtreme Scale:

1. Habilite el proveedor de memoria caché dinámica de eXtreme Scale.

   En WebSphere Application Server 7.0, puede configurar el servicio de memoria caché dinámica para utilizar el proveedor de memoria caché dinámica de eXtreme Scale con la consola de administración o con una propiedad personalizada.

   Después de instalar eXtreme Scale, el proveedor de memoria caché dinámica de eXtreme Scale está disponible inmediatamente como una opción ″Proveedor de

memoria caché″ en la consola de administración. Si desea más información, consulte Selección de un proveedor de servicios de memoria caché.

También puede configurar el proveedor de memoria caché dinámica de eXtreme Scale para una instancia de memoria caché estableciendo los siguientes pares de propiedad personalizada y valor en la instancia. Estas propiedades personalizadas son el único modo de habilitar el proveedor eXtreme Scale en WebSphere Application Server 6.1, del modo siguiente.

```
com.ibm.ws.cache.CacheConfig.cacheProviderName =
   "com.ibm.ws.objectgrid.dynacache.CacheProviderImpl"
```

Si desea utilizar el proveedor de memoria caché dinámica de eXtreme Scale para JSP, los servicios web o la colocación en memoria caché de mandato, debe definir la instancia de baseCache para utilizar el proveedor de memoria caché dinámica de eXtreme Scale. Se utilizan las mismas propiedades de configuración para configurar la instancia de baseCache. Asimismo, recuerde que estas propiedades de configuración se deben definir como las propiedades personalizadas de JVM. Esta referencia se aplica a la propiedad de configuración de memoria caché descrita en esta sección excepto para la colocación en memoria caché del servlet. Para utilizar eXtreme Scale con el proveedor de memoria caché dinámica para la colocación en memoria caché de servlet, asegúrese de configurar la habilitación en las propiedades del sistema, en lugar de las propiedades personalizadas.

Si baseCache está configurada para utilizar el proveedor de memoria caché dinámica de eXtreme Scale, todas las demás instancias de memoria caché del servidor utilizarán el proveedor de eXtreme Scale de forma predeterminada. Para que una instancia de memoria caché utilice el proveedor de memoria caché dinámica predeterminado, establezca la propiedad de proveedor de memoria caché del modo siguiente: com.ibm.ws.cache.CacheConfig.cacheProviderName = ″default″. Cualquier propiedad de configuración del proveedor de memoria caché dinámica de eXtreme Scale establecida para baseCache corresponde a la propiedad de configuración predeterminada para todas las instancias de memoria caché respaldadas por eXtreme Scale. Los clientes deben extremar la atención al basarse en los valores predeterminados para estas propiedades.

2. Configure el valor de replicación para la memoria caché.

   Con el proveedor de memoria caché dinámica de eXtreme Scale, es posible tener instancias de memoria caché local e instancias de memoria caché duplicadas. En el caso de las instancias de memoria caché local, no es necesaria ninguna configuración adicional y el resto de esta sección se puede omitir.

   Las memorias cachés duplicadas se pueden crear de dos formas. El primer método es habilitar la réplica de memoria caché a través de la consola de administración. Esta habilitación se puede realizar en cualquier momento en WebSphere Application Server versión 7.0, pero necesitará la creación de un dominio de duplicación DRS en WebSphere Application Server versión6.1.

   El segundo método es utilizar el siguiente par de propiedad personalizada y valor para aplicar la memoria caché para informar de que se trata de una memoria caché duplicada, aunque todavía no se le haya asignado un dominio de duplicación DRS.

   ```
   com.ibm.ws.cache.CacheConfig.enableCacheReplication = "true"
   ```

3. Configure la topología para el servicio de memoria caché dinámica.

   El único parámetro de configuración necesario para el proveedor de memoria caché dinámica de eXtreme Scale es la topología de memoria caché. La siguiente variable se debe establecer como una propiedad personalizada en el servicio de memoria caché dinámica.

`com.ibm.websphere.xs.dynacache.topology`

Lo siguiente son los tres valores posibles para esta propiedad.

- embedded
- embedded_partitioned
- remote

Debe utilizar uno de los valores permitidos.

4. Configure el número de contenedores iniciales para el servicio de memoria caché dinámica.

   Puede maximizar el rendimiento de las memorias caché que están utilizando la topología incorporada con particiones configurando el número de contenedores iniciales. La siguiente variable se debe establecer como una propiedad del sistema en la máquina virtual Java de WebSphere Application Server.

   `com.ibm.websphere.xs.dynacache.num_initial_containers`

   El valor recomendado de esta propiedad de configuración es un entero que es igual o ligeramente inferior que el número total de instancias WebSphere Application Server que acceden a esta instancia de memoria caché distribuida. Por ejemplo, si un servicio de memoria caché dinámica está compartido entre los miembros de una cuadrícula, la variable se debe establecer en el número de miembros de cuadricula.

5. Configure la cuadrícula de servicio de catálogo de eXtreme Scale.

   Al utilizar eXtreme Scale como el proveedor de memoria caché dinámica para una instancia de memoria caché distribuida, debe configurar una cuadrícula de servicio de catálogo de eXtreme Scale.

   Una única cuadrícula de servicio de catálogo puede prestar servicio a varios proveedores de servicios de memoria caché dinámica respaldados por eXtreme Scale.

   Un servicio de catálogo se puede ejecutar dentro o fuera de los procesos WebSphere Application Server.

   Cuando se está ejecutando una cuadrícula de servicio de catálogo, debe establecer la propiedad personalizada **catalog.services.cluster** para los puntos finales de servicio de catálogo.

6. Configure los objetos de clave personalizados.

   Cuando se utilicen los objetos personalizados como claves, los objetos deben implementar la interfaz Serializable o Externalizable. Cuando se utilizan las topologías particionadas incorporadas o las topologías incorporadas, debe colocar los objetos en la vía de acceso de biblioteca compartida de WebSphere, simplemente como si estuvieran siendo utilizadas con el proveedor de la memoria caché dinámica predeterminada. Consulte Utilización de las interfaces DistributedMap y DistributedObjectCache para la memoria caché dinámica en el centro de información de WebSphere Application Server Network Deployment si desea más detalles.

   Si utiliza la topología remota, debe colocar los objetos de clave personalizados en la CLASSPATH para los contenedores autónomos de eXtreme Scale.

7. Configure los servidores de contenedor eXtreme Scale.

   Los datos almacenados en la memoria caché se almacenan en contenedores WebSphere eXtreme Scale. Los contenedores se pueden ejecutar dentro o fuera de los procesos WebSphere Application Server. El proveedor de eXtreme Scale crea automáticamente contenedores dentro del proceso WebSphere cuando se utilizan topologías incorporadas o topologías particionadas incorporadas para una instancia de memoria caché. No es necesaria realizar una configuración adicional para estas topologías.

Si se utiliza la topología remota, debe iniciar los contenedores autónomos de eXtreme Scale, antes de que se inicien las instancias de WebSphere Application Server que acceden a la instancia de memoria caché. Asegúrese de que todos los contenedores de servicio de memoria caché dinámica específico señalan a los mismos puntos finales de servicio de catálogo.

Los archivos de configuración XML para los contenedores autónomos del proveedor de memoria caché dinámica de eXtreme Scale se encuentran en el directorio `<raíz_instalación>/customLibraries/ObjectGrid/dynacache/etc` para las instalaciones encima de WebSphere Application Server, o el directorio `<raíz_instalación>/ObjectGrid/dynacache/etc` para las instalaciones autónomas. Los archivos se denominan `dynacache-remote-objectgrid.xml` y `dynacache-remote-definition.xml`. Realice copias de estos archivos para editar y utilizar cuando se inicien los contenedores autónomos para el proveedor de la memoria caché dinámica de eXtreme Scale. El parámetro **numIntitialContainers** del archivo **`dynacache-remote-deployment.xml`** se deben establecer de acuerdo con el número de procesos de contenedor que se ejecutan.

Nota: El conjunto de procesos de contenedor debe tener suficiente memoria libre para dar servicio a todas las instancias de memoria caché dinámica configuradas para utilizar la topología remota. Cualquier proceso WebSphere Application Server que comparta los mismos o valores equivalentes para **catalog.services.cluster** debe utilizar el mismo conjunto de contenedores autónomos. El número de contenedores y el número de máquinas en las que residen se deben redimensionar de forma consecuente. Consulte "Planificación de capacidad y alta disponibilidad" si desea detalles adicionales.

En el siguiente código, se muestra una entrada de línea de mandato de ejemplo de UNIX® que lanza un contenedor autónomo para el proveedor de la memoria caché dinámica de eXtreme Scale:

`startOgServer.sh` *contenedor1* `-objectGridFile` *../dynacache/etc/dynacache-remote-objectgrid.xml* `-deploymentPolicyFile` *../dynacache/etc/dynacache-remote-deployment.xml* `-catalogServiceEndpoints` *MiServidor1.empresa.com:2809*

Planificación de capacidad y alta disponibilidad

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 del rendimiento con el proveedor de memoria caché dinámica, consulte "Ajuste del proveedor de la memoria caché dinámica" en la página 78.

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.

Ajuste del proveedor de la memoria caché dinámica

El proveedor de la memoria caché dinámica WebSphere eXtreme Scale soporta los siguientes parámetros de configuración para el ajuste de rendimiento.

Por qué y cuándo se efectúa esta tarea

- **com.ibm.websphere.xs.dynacache.ignore_value_in_change_event**: Cuando registre un receptor de suceso de cambio con el proveedor de memoria caché dinámica y genere una instancia ChangeEvent, existe una sobrecarga asociada a la deserialización de la entrada de la memoria caché para que el valor se pueda

colocar dentro de ChangeEvent. Si se establece este parámetro opcional de la instancia de memoria caché en `true` se omite la deserialización de la entrada de memoria caché al generar ChangeEvents. El valor devuelto será nulo, en el caso de una operación remove, o una matriz de bytes que contiene el formato serializado del objeto. Las instancias InvalidationEvent llevan una penalización de rendimiento similar, que puede evitar estableciendo com.ibm.ws.cache.CacheConfig.ignoreValueInInvalidationEvent en `true`.

- **com.ibm.websphere.xs.dynacache.disable_recursive_invalidate**: la invalidación de la memoria caché dinámica es recursiva de forma predeterminada. Con el proveedor de memoria caché dinámica WebSphere eXtreme Scale, la sobrecarga adicional se asocia a realizar una invalidación recursiva en topologías incorporada con particiones y remotas. Si una memoria caché no necesita la invalidación recursiva, establezca este parámetro en `true` para impedir la sobrecarga.
- **com.ibm.websphere.xs.dynacache.enable_compression**: el proveedor de memoria caché dinámica eXtreme Scale puede comprimir las entradas de memoria caché en la memoria para aumentar la densidad de la memoria caché. Esto presenta una sobrecarga adicional para las operaciones de lectura y escritura, pero pueda ahorrar una cantidad de memoria significativa de aplicaciones, como la colocación en memoria caché de servlet.

Capítulo 4. Escalabilidad

WebSphere eXtreme Scale se puede escalar a través del uso de datos particiones y puede escalar miles de contenedores 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.

Particionamiento

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.

© Copyright IBM Corp. 2009, 2009

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.

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.

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.

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. 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 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

Existen dos estrategias de colocación disponibles en WebSphere eXtreme Scale, partición fija y por contenedor. La estrategia de colocación afecta en el modo cómo se colocan las particiones y cómo se distribuyen los datos.

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 colocación 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.

Interfaz PartitionableKey

Cuando una configuración de eXtreme Scale utiliza la estrategia de colocación de partición fija, depende del método hash de la clave en una partición para insertar, obtener, actualizar o eliminar el valor. Se llama al método hashCode en la clave y debe estar bien definido, si se crea una clave personalizada. Sin embargo, otra opción es utilizar la interfaz PartitionableKey. Con la interfaz PartitionableKey, puede utilizar un objeto que no sea la clave para realizar el método hash en una partición.

Puede utilizar la interfaz PartitionableKey en situaciones en las que existen varias correlaciones y los datos que confirma están relacionados y, por lo tanto, deben colocarse en la misma partición. WebSphere eXtreme Scale no soporta el compromiso de dos fases, así que varias transacciones de correlaciones no se deben comprometer, si se dividen en varias particiones. Si PartitionableKey realiza un método hash en la misma partición para las claves en distintas correlaciones del mismo conjunto de correlaciones, se pueden comprometer de forma conjunta.

También puede utilizar la interfaz PartitionableKey cuando se deban colocar grupos de claves en la misma partición, pero no, necesariamente, durante una única transacción. Si se debe realizar el método hash en claves de una ubicación, departamento, tipo de dominio o algún otro tipo de identificador, las claves secundarias se pueden asignar a un PartitionableKey padre.

Por ejemplo, los empleados debe realizar el método hash en la misma partición que su departamento. Cada clave de empleado debería tener un objeto PartitionableKey que pertenezca a la correlación de departamento. Tanto el empleado como el departamento deberán realizar un método hash en la misma partición.

La interfaz PartitionableKey proporciona un método, llamado ibmGetPartition. El objeto devuelto de este método debe implementar el método hashCode. Se utilizará el resultado devuelto del uso del método hashCode alternativo para direccionar la clave a una partición.

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 5. Disponibilidad

Con una gran disponibilidad, WebSphere eXtreme Scale proporciona redundancia y detección de anomalías.

WebSphere eXtreme Scale organiza automáticamente las cuadrículas de máquinas 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. Consulte el apartado "Arquitectura y topología" en la página 9 para obtener más información.

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

© Copyright IBM Corp. 2009, 2009

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 122 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 la cuadrícula del 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, pero el servicio de ubicación y administración se ejecutan en cualquier lugar. 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 82.

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.

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 más que bloques 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 6. 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 7. 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 8. 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 del lado del 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.

Arquitectura de réplica

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. Para realizar la réplica, para cada partición existen un fragmento primario y 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. Una réplica asíncrona sigue conservando el orden de las transacciones enviadas desde el fragmento primario.

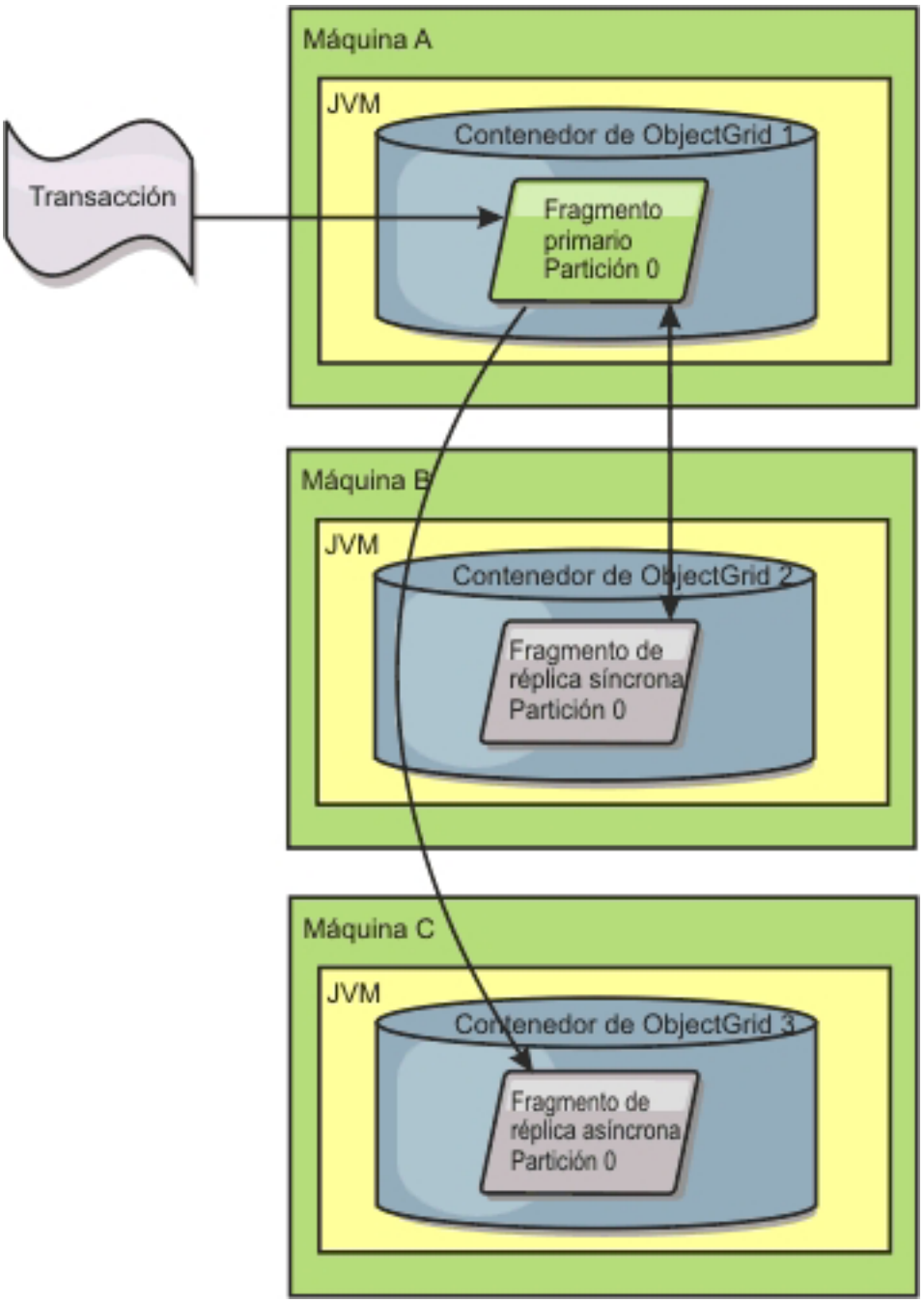


Figura 33. Vía de acceso de comunicación entre un fragmento primario y fragmentos de réplica

El coste de memoria de la réplica

Para poner la réplica en línea, debe crearse un punto de comprobación para copiarse encima de los datos existentes del fragmento primario. Sin embargo, el principal coste de memoria en el fragmento primario es la memoria para cada entrada que se modifica una vez que se ha producido el punto de comprobación. Si los datos cambian mucho después del punto de comprobación, esta operación cuesta una cantidad de memoria que es proporcional al número de entradas modificadas. Una vez que el punto de comprobación se ha copiado en la réplica, los cambios se vuelven a fusionar en el punto de comprobación. Las entradas

modificadas no se liberan hasta que no se han enviado a todas las réplicas necesarias.

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 un número de particiones. Cada partición 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 equilibra los tipos de fragmentos que se distribuyen 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.

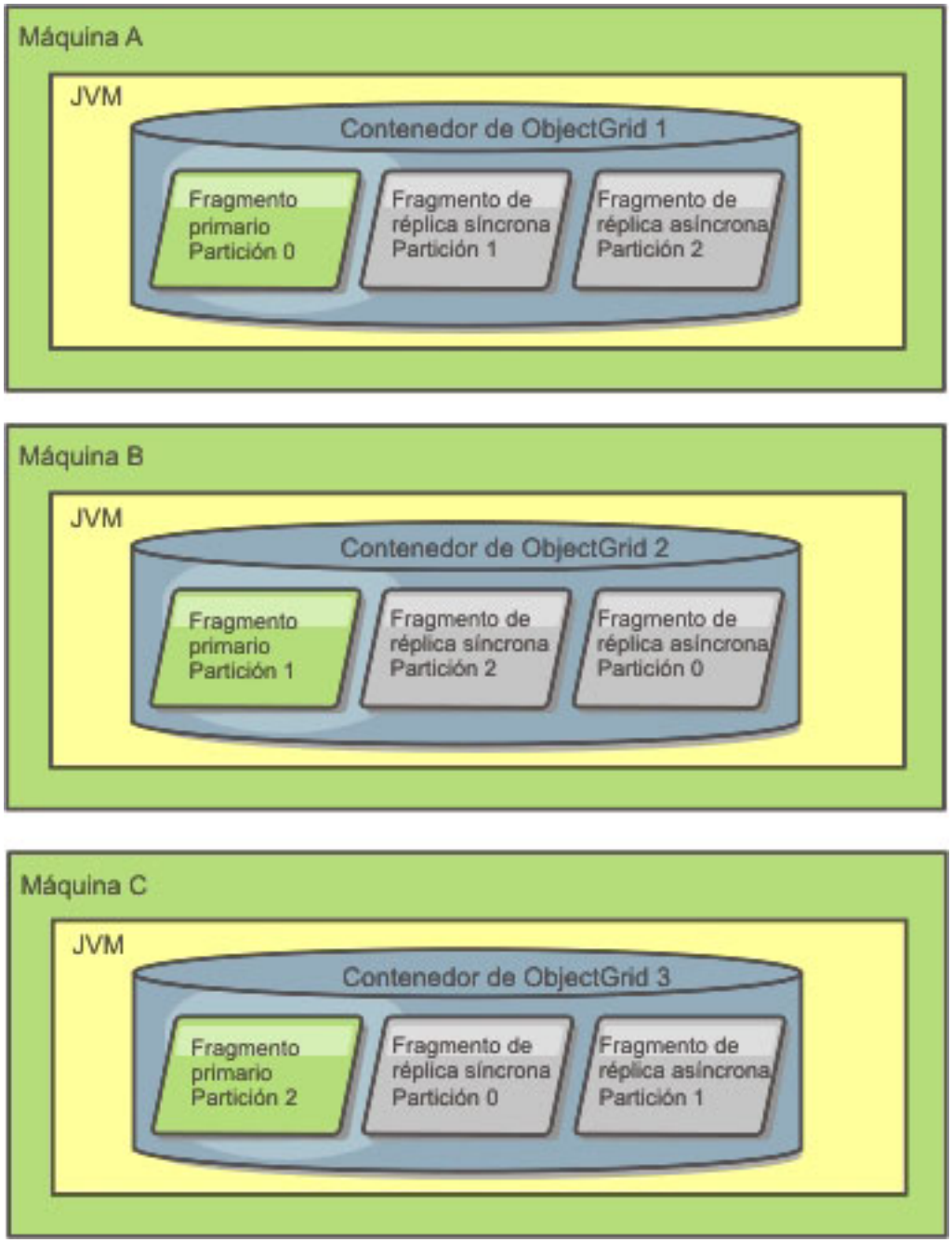


Figura 34. La colocación de un conjunto de correlaciones de ObjectGrid con una política de despliegue de 3 particiones con un valor minSyncReplicas de 1, un valor maxSyncReplicas de 1 y un valor maxAsyncReplicas de 1

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 104 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.
```

Cuando se inicia por primera vez el fragmento réplica, 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.

Para obtener una copia de los datos en el fragmento primario, los fragmentos réplica síncronos y asíncronos realizan la misma secuencia de arranque. Durante el registro del fragmento réplica, el fragmento primario crea un punto de control de los datos actuales. Los datos del fragmento primario están en un estado copiar al grabar. Los datos actuales que van al fragmento réplica nunca se modifican, pero se produce una operación copiar al grabar siempre que una transacción nueva actualiza los registros del fragmento primario. El fragmento primario puede

continuar con el proceso sin modificar los datos que van al fragmento réplica. El fragmento primario conserva una lista de los cambios realizados desde el punto de control.

Los datos del punto de control se envían a la réplica. Una vez que el punto de control ha llegado a la réplica, se libera la memoria del punto de control. Los cambios realizados desde que se creó el punto de control se fusionan. La lista de los cambios también se envía al fragmento réplica. A medida que los cambios se envían, se libera la memoria de los cambios.

Después de que la réplica complete la fase de punto de control, pasa a la modalidad de igual y empieza a recibir datos en cuanto el fragmento primario los recibe. En este punto, el fragmento réplica empieza a comportarse como un fragmento réplica síncrono o asíncrono.

Cuando un fragmento de réplica alcanza la modalidad de igual, imprime un mensaje en el archivo `SystemOut.log` para la réplica.

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

El tiempo se refiere a la cantidad de tiempo que tardó el fragmento réplica en obtener todos los datos iniciales del fragmento primario, incluidos los datos del punto de control y los cambios adicionales que se realizan durante la copia del punto de control. El valor de tiempo puede mostrarse como cero o muy bajo si el fragmento primario no tiene ningún dato que deba replicar.

Fragmento réplica síncrono

Si la réplica nueva es un fragmento réplica síncrono, el fragmento primario inicia a continuación una solicitud o respuesta si una transacción realiza una confirmación en el fragmento primario. El fragmento primario espera hasta que el fragmento réplica responda que ha recibido los datos. Los datos del fragmento réplica síncrono permanecen al mismo nivel que los del fragmento primario.

Fragmento réplica asíncrono

Si la réplica nueva es un fragmento réplica asíncrono, el fragmento primario envía los datos a la réplica, pero no espera una respuesta. La réplica asíncrona ordena y aplica los datos enviados desde el fragmento primario. No se garantiza que los datos del fragmento réplica asíncrono estén al mismo nivel que los del fragmento primario.

Modalidad de igual para todos los fragmentos réplica

Cuando está en modalidad de igual, después de que todos los fragmentos réplica hayan recibido un cambio de transacción, se libera la memoria de la transacción en el fragmento primario. Los fragmentos réplica sólo reciben datos del fragmento primario durante transacciones en las que se insertan, actualizan y eliminan datos. No se contacta a los fragmentos réplica para leer los datos del fragmento primario.

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. 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. Si por detrás hubiera un fragmento réplica asíncrono, recibiría una copia nueva de los datos y el registro.

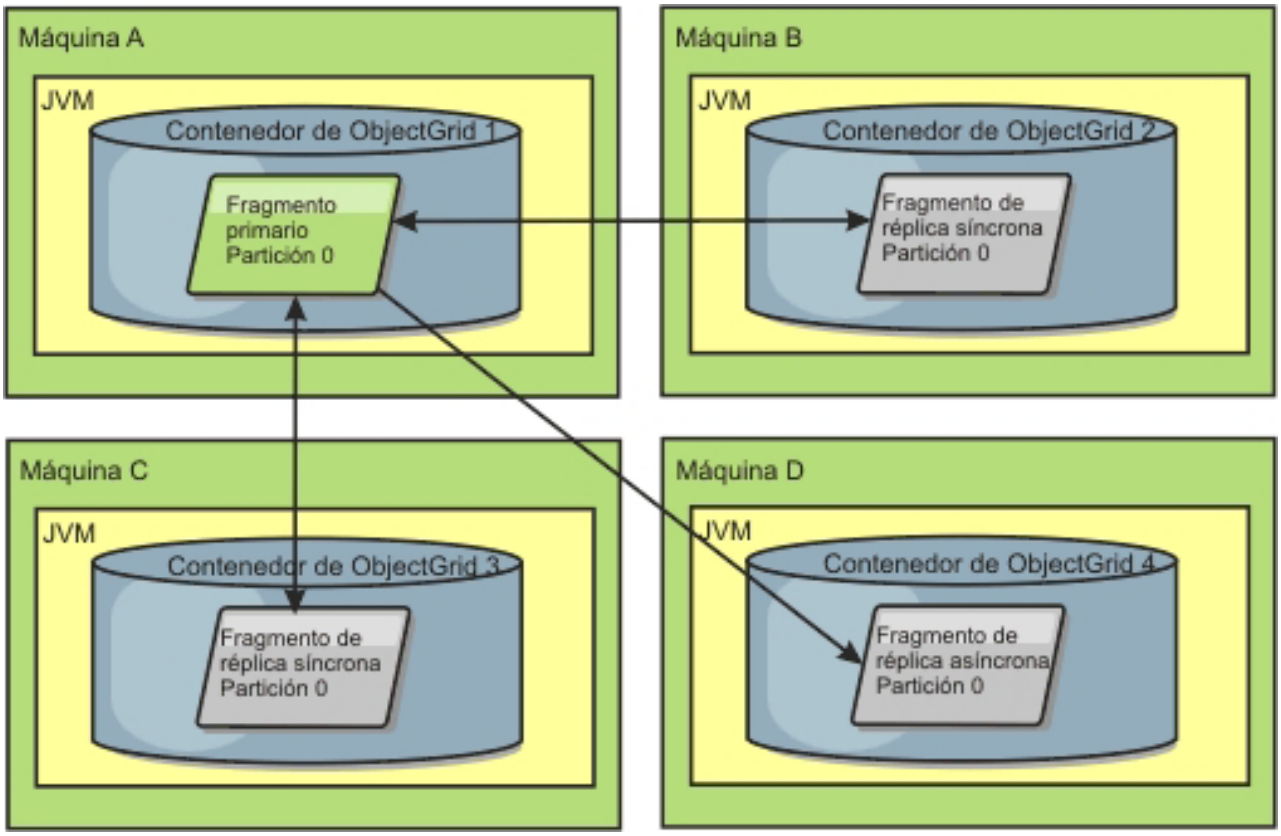


Figura 35. Colocación de ejemplo de un conjunto de correlaciones ObjectGrid para la partición partition0. La política del despliegue tiene un valor minSyncReplicas de 1, un valor maxSyncReplicas de 2 y un valor maxAsyncReplicas de 1.

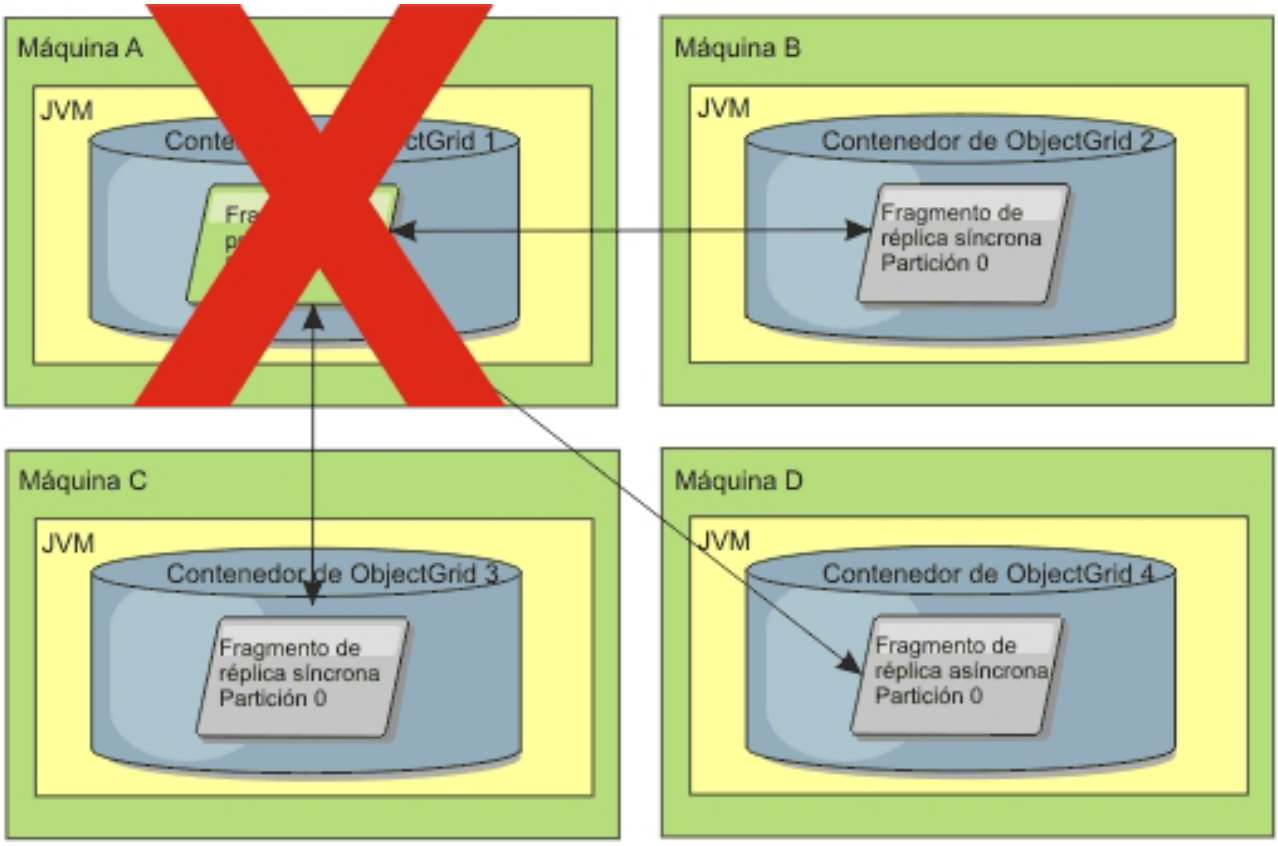


Figura 36. El contenedor del fragmento primario falla.

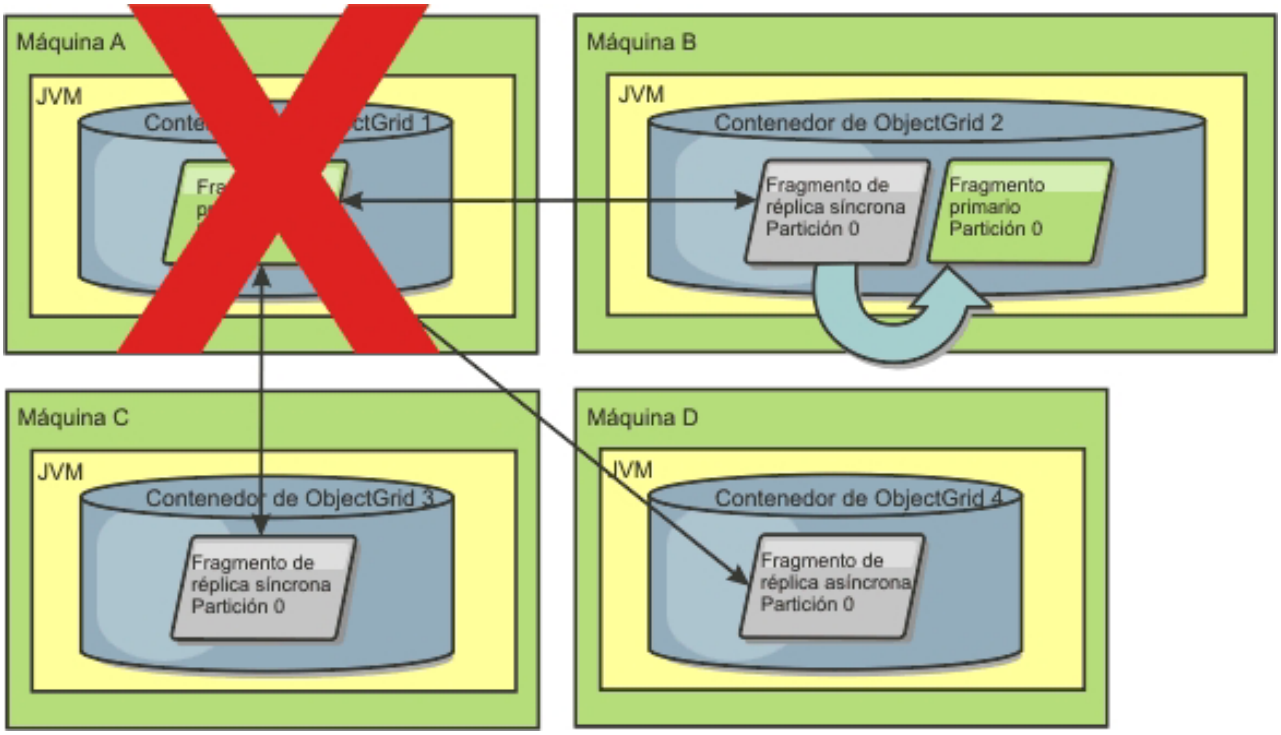


Figura 37. El fragmento de réplica síncrona en el contenedor 2 de ObjectGrid pasa a ser el fragmento primario.

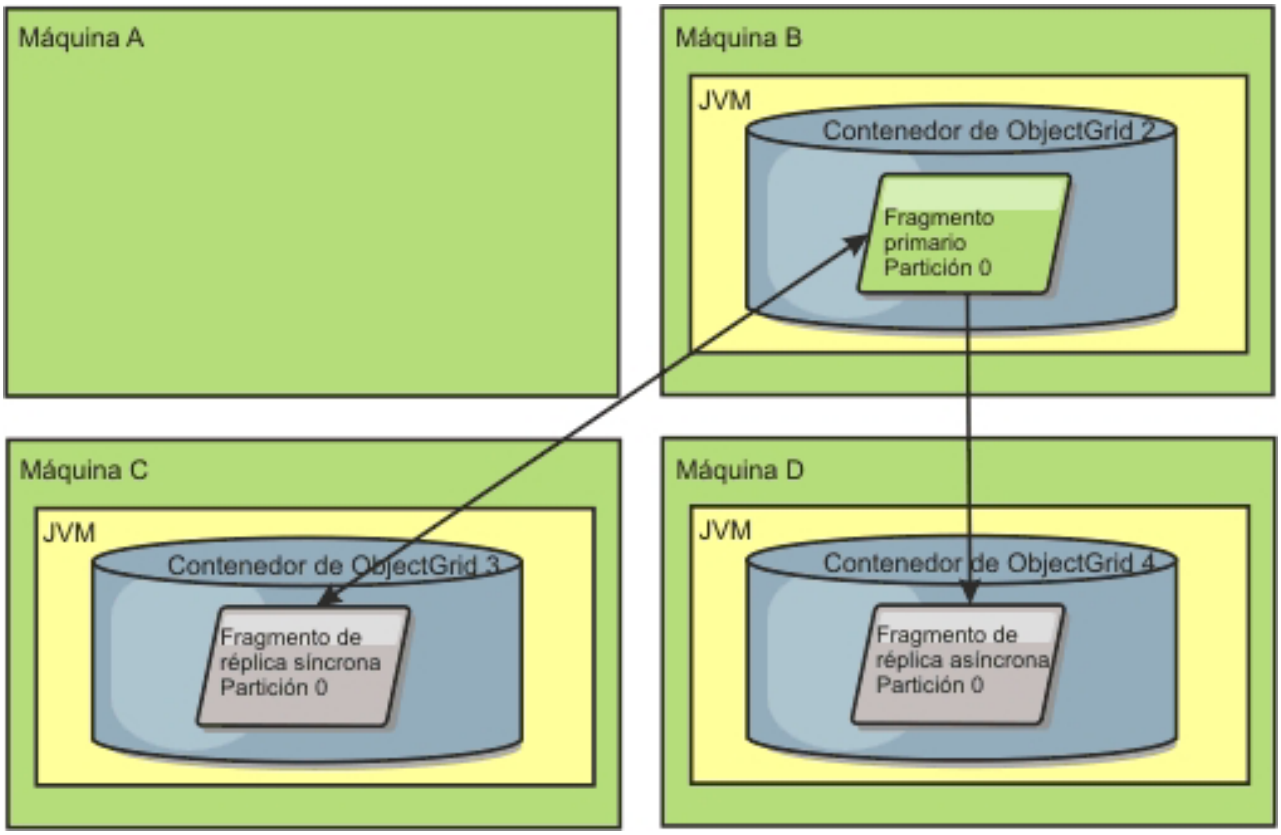


Figura 38. La máquina B contiene el fragmento primario. En función de cómo se establece la modalidad de reparación automática y la disponibilidad de los contenedores, un nuevo fragmento de réplica síncrona se podría colocar o no en una máquina.

Recuperación de un fragmento réplica

El fragmento primario controla al fragmento réplica. 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” en la página 111.

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.

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.

Utilización de zonas para la colocación de réplicas

El soporte de zonas permite realizar configuraciones sofisticadas para la colocación de fragmentos en centros de datos. Con esta prestación, las cuadrículas de miles de particiones se pueden gestionar fácilmente utilizando un puñado de reglas de colocación opcionales. Un centro de datos puede estar en varias plantas de un edificio, distintos edificios, o incluso en distintas ciudades u otras distinciones, tal como se haya configurado con las reglas de zonas.

Flexibilidad de zonas

Puede colocar fragmentos en zonas. Esta función le permite tener más control sobre cómo eXtreme Scale coloca los fragmentos en una cuadrícula. Máquinas virtuales Java que aloja un servidor eXtreme Scale se puede marcar con un identificador de zona. El archivo de despliegue ahora puede incluir una o más reglas de zona y estas reglas de zona están asociadas con un tipo de fragmento. La mejor forma de explicarlo es con algunos ejemplos seguidos de más detalles.

El control de zonas de colocación de cómo eXtreme Scale asigna primarios y réplicas para configurar topologías avanzadas.

Una máquina virtual Java puede tener valores servidores eXtreme Scale activos. Un contenedor puede alojar varios fragmentos de un solo eXtreme Scale.

Esta prestación es útil para asegurarse de que los fragmentos primarios y los fragmentos réplicas se colocan en distintas ubicaciones o zonas para obtener una mejor alta disponibilidad. Normalmente, eXtreme Scale no coloca un fragmento primario y de réplica en la máquinas virtuales Java con la misma dirección IP. Esta regla simple normalmente impide que dos servidores eXtreme Scale se coloquen en el mismo sistema físico. No obstante, quizás necesite un mecanismo más flexible. Por ejemplo, es posible que esté utilizando dos chasis blade y desee que los primarios se *extiendan* por los dos chasis y la réplica de cada primario pueda colocarse en el otro chasis desde el primario.

Primarios *extendidos* significa que los primarios se colocan en cada zona y la réplica de cada primario se coloca en la zona opuesta. Por ejemplo, el primario 0 estaría en zoneA, y la réplica sinc 0 estaría en zoneB. El primario 1 estaría en zoneB, y la réplica sinc 1 estaría zoneA.

En este caso el nombre del chasis sería el nombre de zona. De forma alternativa, puede especificar nombres para zonas según sus plantas de un edificio y utilizar las zonas para asegurarse de que los primarios y las réplicas para los mismos datos estén en distintas plantas. También es posible edificios y centros de datos. Se han realizado comprobaciones en centros de datos utilizando zonas como mecanismo para asegurarse de que los datos se dupliquen de forma correcta entre los centros de datos. Si utiliza el gestor de sesiones HTTP para eXtreme Scale, también puede utilizar zonas. Con esta característica puede desplegar una sola aplicación web por tres centros de datos y asegurarse de que las sesiones HTTP para los usuarios se dupliquen a lo largo de los centros de datos para que las sesiones se puedan recuperar incluso si falla todo un centro de datos.

WebSphere eXtreme Scale reconoce la necesidad de gestionar una cuadrícula de gran tamaño por varios centros de datos. Si es necesario, puede asegurarse de que los fragmentos primarios y de copia de seguridad para la misma partición estén ubicados en distintos centros de datos. Puede colocar todos los primarios en el centro de datos 1 y todas las réplicas en el centro de datos 2, o puede aplicar un sistema de redondeo en los primarios y las réplicas entre los dos centros de datos. Las reglas son flexibles por lo que hay muchos escenarios posibles. eXtreme Scale también puede gestionar miles de servidores, que junto con la colocación totalmente automática con reconocimiento de centro de datos, hace que esas cuadrículas de gran tamaño sean asequibles desde un punto de vista administrativo. Los administradores pueden especificar lo que desean de forma simple y eficiente.

Como administrador, utiliza zonas de colocación para controlar donde se colocan los fragmentos primarios y de réplica, lo que permite configurar topologías avanzadas de alta disponibilidad y alto rendimiento. Puede definir una zona para cualquier agrupación lógica de procesos de eXtreme Scale, tal como se ha indicado anteriormente: estas zonas pueden corresponder a ubicaciones de estaciones de trabajo físicas, como un centro de datos, una planta de un centro de datos o un chasis de blade. Puede extender datos a través de zonas, que proporciona una mayor disponibilidad, o puede partir los primarios y las réplicas en distintas zonas cuando es necesario una parada activa.

Asociación de un servidor eXtreme Scale a una zona que no utiliza WebSphere Extended Deployment

Si se utiliza eXtreme Scale con Java Standard Edition o un servidor de aplicaciones que no se basa en WebSphere Extended Deployment versión 6.1, se puede asociar una JVM que es un contenedor de fragmento con una zona si utiliza las siguientes técnicas.

Aplicaciones que utilizan el script startOgServer

El script startOgServer se utiliza para iniciar una aplicación de eXtreme Scale cuando no se ha incrustado en un servidor existente. El parámetro **-zone** se utiliza para especificar la zona para utilizar todos los contenedores dentro del servidor.

Especificación de la zona al iniciar un contenedor utilizando las API

Asociación de nodos de WebSphere Extended Deployment con zonas

Si utiliza eXtreme Scale con aplicaciones de WebSphere Extended Deployment Java Platform, Enterprise Edition, puede hacer uso de la característica del grupo de

nodos de WebSphere Extended Deployment para colocar de forma automática un miembro de clúster en zonas específicas de máquinas virtuales Java. Una JVM puede ser miembro de una sola zona. Los servidores que se ejecutan en un nodo que forma parte de dicho grupo de nodos se incluyen en la zona especificada por el nombre del grupo de nodos. Debe asegurarse de que estos miembros del clúster no sean miembros de dos o más de esos grupos de nodos. UnaJVM de miembro de clúster comprueba la pertenencia a la zona sólo durante el inicio. Si se añade un nuevo grupo de nodos o se cambia la pertenencia sólo afecta a las Máquinas virtuales Java recién iniciadas o reiniciadas.

Reglas de zonas

Una partición deeXtreme Scale tiene un fragmento primario y cero o más fragmentos de réplica. Para este ejemplo, considere el siguiente convenio de denominación para estos fragmentos. `P` es el fragmento primario, `S` es una réplica síncrona y `A` es una réplica asíncrona. Una regla de zonas tiene tres componentes:

- Un nombre de regla
- Una lista de zonas
- Un distintivo inclusivo o exclusivo

Una regla de zonas especifica el conjunto de zonas posible en el que se puede colocar el fragmento. El distintivo inclusivo indica que tras colocar un fragmento en una zona de la lista, los demás fragmentos también se colocan en esa zona. Un valor exclusivo indica que cada fragmento correspondiente a una partición se coloca en una zona distinta en la lista de zonas. Por ejemplo, el uso de un valor exclusivo significa que si hay tres fragmentos (primario y dos réplicas síncronas), la lista de zonas debe tener tres zonas.

Cada fragmento puede asociarse a una regla de zonas. Una regla de zonas puede compartirse entre dos fragmentos. Cuando una regla se comparte, el distintivo inclusivo o exclusivo se extiende a través de fragmentos de todos los tipos que comparten una sola regla.

Ejemplos

A continuación se proporciona un conjunto de ejemplos que muestran distintos casos de ejemplo y la configuración de despliegue para implementar los casos de ejemplo.

Escritura en bandas de primarios y réplicas a través de zonas

Dispone de tres chasis blade y desea que los primarios se distribuyan a lo largo de los tres, con una sola réplica síncrona colocada en un chasis distinto al primario. Defina cada chasis como una zona con los nombres de chasis `ALPHA`, `BETA` y `GAMMA`. A continuación de proporciona un XML de despliegue de ejemplo:

```
<?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="library">
<mapSet name="ms1" numberOfPartitions="37" minSyncReplicas="1"
maxSyncReplicas="1" maxAsyncReplicas="0">
<map ref="book" />
   <zoneMetadata>
<shardMapping shard="P" zoneRuleRef="stripeZone"/>
<shardMapping shard="S" zoneRuleRef="stripeZone"/>
<zoneRule name ="stripeZone" exclusivePlacement="true" >
<zone name="ALPHA" />
<zone name="BETA" />
<zone name="GAMMA" />
    </zoneRule>
```

```
   </zoneMetadata>
  </mapSet>
 </objectgridDeployment>
</deploymentPolicy>
```

Un XML de despliegue contiene una cuadrícula denominada library con una cola correlación (map) denominada book. Utiliza cuatro particiones con una sola réplica síncrona. La cláusula de metadatos de zona muestra la definición de una sola regla de zonas y la asociación de reglas de zonas con fragmentos. Los fragmentos primario y síncrono están asociados a la regla de zonas ″stripeZone″. La regla de zonas tiene todas las tres zonas en ellas y utiliza la colocación exclusiva. Esta regla indica que si el primario de la partición 0 se coloca en `ALPHA`, la réplica de la partición 0 se colocará en `BETA` o `GAMMA`. De forma parecida, los primarios para otras particiones se colocan en otras zonas y las réplicas se colocarán.

Réplica asíncrona en una zona distinta a la réplica primaria y síncrona

En este ejemplo, existen dos edificios con una alta conexión de latencia entre ellos. Desea que no se pierdan datos de alta alta disponibilidad para todos los casos de ejemplo. No obstante, el impacto en el rendimiento de la réplica síncrona entre edificios le lleva a un compromiso. Desea un primario con una réplica síncrona en un edificio y una réplica asíncrona en el otro edificio. Normalmente, las anomalías son cuelgues de la JVM o anomalías en el sistema en lugar de problemas a gran escala. Con esta topología, puede superar anomalías normales sin pérdida de datos. La pérdida de un edificio es tan rato que alguna pérdida de datos es aceptable en ese caso. Puede crear dos zonas, una para cada edificio. A continuación se muestra el archivo XML de despliegue:

```
<?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="library">
<mapSet name="ms1" numberOfPartitions="13" minSyncReplicas="1"
maxSyncReplicas="1" maxAsyncReplicas="1">
<map ref="book" />
   <zoneMetadata>
<shardMapping shard="P" zoneRuleRef="primarySync"/>
<shardMapping shard="S" zoneRuleRef="primarySync"/>
<shardMapping shard="A" zoneRuleRef="aysnc"/>
<zoneRule name ="primarySync" exclusivePlacement="false" >
<zone name="BldA" />
<zone name="BldB" />
    </zoneRule>
<zoneRule name="aysnc" exclusivePlacement="true">
<zone name="BldA" />
<zone name="BldB" />
    </zoneRule>
   </zoneMetadata>
  </mapSet>
 </objectgridDeployment>
</deploymentPolicy>
```

La réplica primaria y síncrona comparten la regla de zonas primaySync con un valor de distintivo exclusivo de false. Por lo tanto, después de colocar el primario o síncrono en una zona, el otro también se coloca en la misma zona. La réplica asíncrona utiliza una segunda regla de zonas con las mismas zonas que la regla de zonas primarySync pero utiliza el atributo **exclusivePlacement** establecido en true. Este atributo indica que un fragmento no se puede colocar en una zona con otro fragmento de la misma partición. Como resultado, la réplica asíncrona no se coloca en la misma zona que el primario o las réplicas síncronas.

Colocar todos los primarios en una zona y todas las réplicas en otra zona

Aquí, todos los primarios están en una zona específica y todas las réplicas en una zona distinta. Tendremos un primario y una sola réplica asíncrona. Todas las réplicas estarán en la zona A y los primarios en B.

```
<?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="library">
<mapSet name="ms1" numberOfPartitions="13" minSyncReplicas="0"
maxSyncReplicas="0" maxAsyncReplicas="1">
<map ref="book" />
    <zoneMetadata>
<shardMapping shard="P" zoneRuleRef="primaryRule"/>
<shardMapping shard="A" zoneRuleRef="replicaRule"/>
<zoneRule name ="primaryRule">
<zone name="A" />
     </zoneRule>
<zoneRule name="replicaRule">
<zone name="B" />
       </zoneRule>
      </zoneMetadata>
     </mapSet>
   </objectgridDeployment>
 </deploymentPolicy>
```

Aquí, puede ver dos reglas, una para los primarios (P) y otra para la réplica (A).

Zonas en redes de área amplia (WAN)

Es posible que desee desplegar un solo eXtreme Scale en varios edificios o centros de datos con interconexiones de red más lentas. Las conexiones de red más lentas llevan a un ancho de banda más bajo y a conexiones de latencia más alta. La posibilidad de particiones de red también aumenta en esta modalidad debido a la congestión de la red y otros factores. eXtreme Scale aborda este entorno duro de las siguientes formas:

Pulsaciones limitadas entre zonas

Las Máquinas virtuales Java que están agrupadas en en grupos principales se envían pulsaciones entre sí. Cuando el servicio de catálogo organiza las máquinas virtuales Java en grupos, estos grupos no abarcan zonas. Un líder dentro de este grupo pasa información de pertenencia al servicio de catálogo. El servicio de catálogo verifica todas las anomalías notificadas antes de realizar alguna acción. Lo lleva a cabo intentando conectarse a las Máquinas virtuales Java sospechosas. Si el servicio de catálogo ve una detección de anomalía falsa no realizará ninguna acción ya que la partición del grupo principal se arreglará en un corto periodo de tiempo.

El servicio de catálogo también enviará pulsaciones a líderes de grupo principal de forma periódica a velocidad baja para manejar el caso de aislamiento de grupo principal.

Servicio de catálogo como circuito de arbitraje de la cuadrícula

El servicio de catálogo es un circuito de arbitraje para una cuadrícula de eXtreme Scale. Es fundamental que actúe con una sola voz para que la cuadrícula se ejecute. El servicio de catálogo se ejecuta en un conjunto fijo de máquinas virtuales Java que duplican datos de un primario elegido en las demás máquinas virtuales Java de ese conjunto. El servicio de catálogo debe distribuirse entre zonas físicas o centros de datos para disminuir la probabilidad de que quede aislado de la cuadrícula y pueda superar casos de ejemplos de anomalía anticipados.

El servicio de catálogo se comunica con las máquinas virtuales Java de contenedor de la cuadrícula utilizando operaciones idempotentes o recuperables. Lo lleva a cabo comunicándose mediante IIOP. Todos los cambios de estado en el servicio de

catálogo se duplican de forma síncrona entre los miembros actuales que alojan el servicio de catálogo. Esta réplica sólo se lleva a cabo satisfactoriamente si la mayoría de las máquinas virtuales Java acepta el cambio. Esto significa que si el servicio de catálogo se particiona, sólo la partición mayoritaria puede confirmar cambios. El primario de servicio sólo enviará mandatos a lasmáquinas virtuales Java de contenedor si se confirma el cambio de estado que crea ese mandato. Esto significa que una partición minoritaria no puede anticipar su estado o emitir mandatos a contenedores.

Un servicio de catálogo particionado no detiene el funcionamiento de la cuadrícula. La cuadrícula seguirá aceptando solicitudes de cliente y operaciones de ejecución. Si no hay ninguna partición de servicio de catálogo mayoritaria, las anomalías de la cuadrícula no se recuperarán hasta que el servicio de catálogo recupere la mayoría. Si la recuperación se demora, con el tiempo a medida que se produzcan anomalías determinadas particiones pasarán a estar fuera de línea hasta que el servicio de catálogo recobre la mayoría.

Las Máquinas virtuales Java del líder del grupo principal notifican los cambios de pertenencia al servicio de catálogo. Si el servicio de catálogo se particiona, el servicio devolverá una tabla de direccionamiento actualizada para el servicio de catálogo. Dicha tabla de direccionamiento de una partición minoritaria no incluirá la ubicación de un primario. El líder necesitará iterar por todas las Máquinas virtuales Java del servicio de catálogo posibles para intentar localizar la partición primaria. Será necesario hacerlo periódicamente mientras se espera a que se resuelva la partición. Después de recibir una tabla de direccionamiento con un primario, las acciones de recuperación pendientes se redireccionarán mediante el primario del servicio de catálogo mayoritario.

Si el grupo principal no puede conectarse con un primario de servicio principal durante un periodo de tiempo, significa que está desconectado físicamente del resto de la cuadrícula (posiblemente con una partición de servicio de catálogo minoritario) o el servicio de catálogo está estancado en particiones mayoritarias. Es imposible distinguir una cosa de la otra. Si hay una partición de servicio de catálogo mayoritaria, es posible que se recupere de la pérdida aparente del grupo principal desconectado. Esto puede causar que haya dos primarios para la misma partición, el primario antiguo existente y el nuevo primario en el resto de la red. La partición de servicio de catálogo mayoritaria no tiene ninguna forma de ″matar″ los primarios antiguos ya que está en un estado de red desconectado con los primarios antiguos. Cuando el servicio de catálogo se recupera y el grupo principal desconectado descubre los nuevos primarios, el servicio de catálogo observará que hay dos primarios. Indicará a los grupos principales desconectados previamente que supriman todos los fragmentos y, a continuación, se producirá el equilibrio.

Si las particiones de servicio de catálogo están en dos particiones menores o en una sola partición menor superviviente, el cliente necesitará implicarse en la recuperación. Es necesario un mandato JMX (Java Management Extensions) para especificar que está permitido que una sola partición minoritaria tome medidas. Debe asegurarse de que las otras particiones minoritarias estén detenidas.

Direccionamiento a zonas según preferencias

Con el direccionamiento a zonas según preferencia, eXtreme Scale puede direccionar las transacciones a las zonas basándose en las preferencias que especifique.

WebSphere eXtreme Scale le permite ejercer un control significativo sobre dónde se colocan los fragmentos de ObjectGrid. Consulte “Utilización de zonas para la

colocación de réplicas" en la página 112 si desea información sobre algunos escenarios básicos y cómo configurar la política de despliegue de forma consecuente.

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 colocación 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 partición 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 colocación 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”. Consulte el tema “Asociación de los nodos WebSphere Extended Deployment con las zonas” en “Utilización de zonas para la colocación de réplicas” en la página 112.

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.

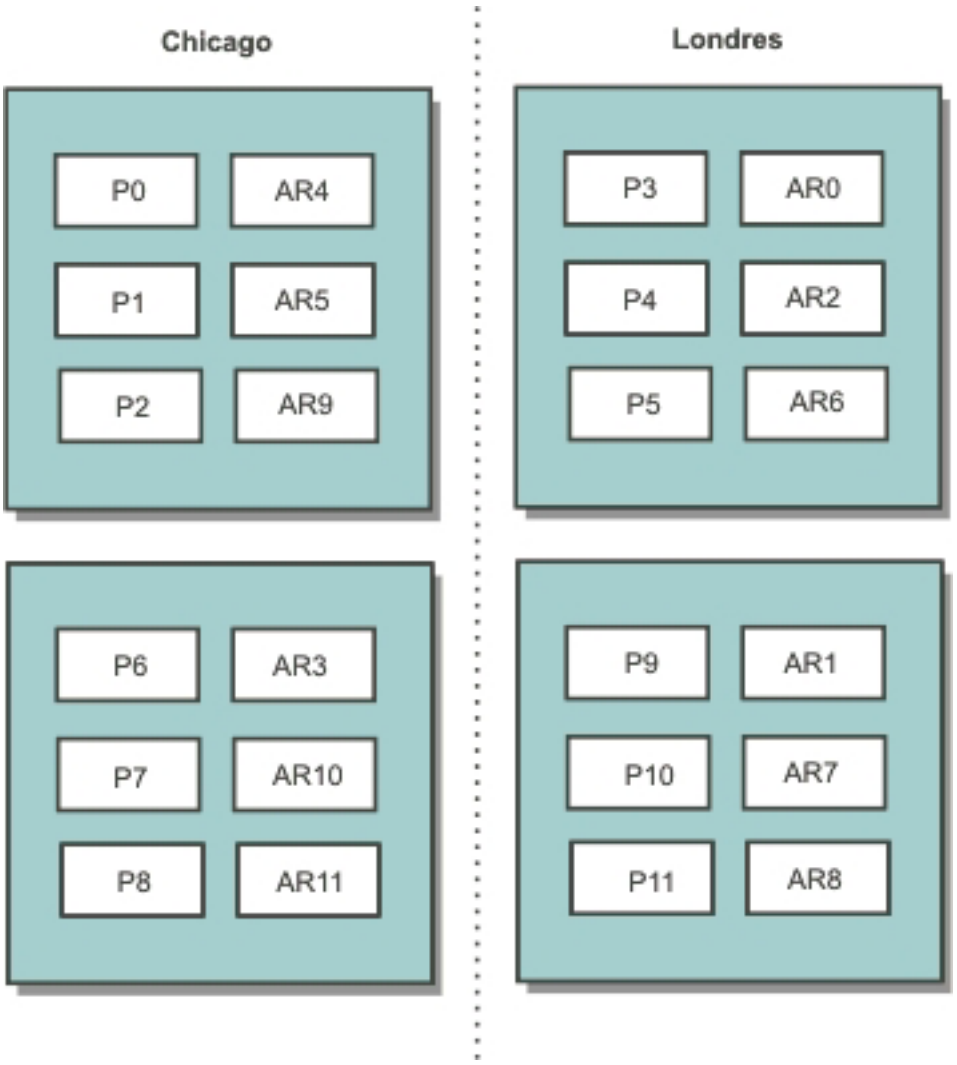


Figura 39. Primarios y réplicas en las zonas

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.

Tipos de detección de migración tras error

WebSphere eXtreme Scale utiliza dos métodos para la detección de la migración tras error.

Pulsaciones

1. Los sockets se mantienen abiertos entre máquinas 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, 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 manejan utilizando las pulsaciones.

Las pulsaciones se envían de forma periódica entre pares de procesos: cuando se pierden un número fijo de pulsaciones, se supone 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. 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 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 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.

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. Consulte el apartado “Tipos de detección de migración tras error” en la página 122 para obtener detalles sobre cómo disminuir el tiempo que se tarda en detectar una anomalía grave.

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 la cuadrícula del 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, pero el servicio de ubicación y administración se ejecutan en cualquier lugar. 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.

Servicio de catálogo de alta disponibilidad

Un 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 una cuadrícula para alta disponibilidad.

Componentes del 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 de la cuadrícula del servidor 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 de la cuadrícula del servidor 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. La cuadrícula del servidor 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 la cuadrícula del servidor 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 de la cuadrícula 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 de varios. eXtreme Scale se compone de varios de estos grupos, y este agrupamiento es un habilitador de escalabilidad clave. Cada grupo es un grupo de máquinas 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. Se ejecuta como uno de los N servicios elegidos en la cuadrícula, de forma 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 una cuadrícula de catálogo, los miembros de la cuadrícula se enlazan juntos. Planifique con atención la topología de la cuadrícula del catálogo, porque no podrá modificar la configuración del catálogo durante la ejecución. Extienda la cuadrícula tanto como sea posible para evitar errores.

Inicio de una cuadrícula de servidor de catálogo

Conexión de contenedores eXtreme Scale incorporados en WebSphere Application Server con una cuadrícula de catálogo autónomo

Puede configurar contenedores eXtreme Scale que están incorporados en un entorno WebSphere Application Server para conectarse a una cuadrícula de catálogo autónomo. Utilice la misma propiedad para conectar el servidor de aplicaciones con la cuadrícula de catálogo. No obstante, la propiedad no gestiona el ciclo de vida del catálogo con los servidores. En lugar de esto, la propiedad permite a los contenedores localizar la cuadrícula de catálogo remoto. Para establecer la propiedad, consulte .

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.

Quórums del servidor de catálogos

El quórum es el número mínimo de servidores de catálogos necesario para realizar la colocación de operaciones para la cuadrícula. El número mínimo es el conjunto completo de servidores de catálogos, a menos que el quórum haya sido modificado.

Términos importantes

Lo que aparece a continuación es una lista de términos relacionados con las consideraciones de quórum para 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. Por ejemplo, los servidores de un centro de datos pueden estar todos marcados en una zona.
- Pulso: un pulso es el acto de realizar un ping a una máquina virtual Java (JVM) para determinar su estado.

Topología

Esta sección explica cómo funciona IBM 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 fluyan todos los cortafuegos en todo el tráfico 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 realizará de forma agresiva pulsaciones en las JVM de una sola zona y si se pierde una pulsación se generará un suceso de migración tras error que durará tanto como el servicio de catálogo tenga quórum.

Cuadrícula de servicio de catálogos y servidores de contenedor

Una cuadrícula es una colección de JVM similares. Un 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.

La cuadrícula de servicio de catálogo utiliza un mecanismo de quórum completo. Esto significa que todos los miembros de la cuadrícula 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 incluirá las JVM de la misma zona. Las JVM de distintas zonas nunca estarán en el mismo grupo principal.

Un grupo principal intentará 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, ya sea pasando el distintivo habilitado `-quorum` en el mandato startOgServer, o añadiendo 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 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/startOgServer 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 en contra de 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 de la cuadrícula del servicio de catálogo, seguirá intentándolo.

Pulsaciones de la cuadrícula del servicio de catálogo

El 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 contener como anómalo 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 será visible en las consultas de xsadmin. Habrá mensajes en los registros de la JVM del contenedor que indican que esto ha sucedido. Estas JVM se deben reiniciar manualmente.

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. La cuadrícula 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 responderá a los sucesos de contenedor mientras que el servicio de catálogo tenga quórum.

Si el servicio de catálogo pierde quórum, esperará a que se restablezca el quórum. Mientras que el servicio de catálogo no tenga quórum, ignorará los sucesos de los servidores de contenedor. Los servidores de contenedor reintentarán 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 volverá a unir en la cuadrícula del servicio de catálogo y se restablecerá 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 a la cuadrícula 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 a la cuadrícula 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 la cuadrícula 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.

SSL inhabilitado

Está totalmente desaconsejado especificar un tiempo de espera de transacción en el archivo XML de descriptor de ObjectGrid. El cliente intentará una sola transacción hasta que se exceda este tiempo de espera. Si no se ha especificado un tiempo de espera, el cliente lo intentará de forma indefinida, que no se recomienda, ya que la hebra del cliente se bloqueará para una cantidad de tiempo indeterminada. El tiempo de transacción se deberá establecer a varios del tiempo máximo de transacción esperado.

También se recomienda encarecidamente establecer el tiempo de espera de petición ORB en el archivo orb.properties. El cliente se bloqueará en un socket de caída de tensión para este tiempo máximo. Si el tiempo transcurrido desde que se emitió la petición sigue siendo inferior que el tiempo de espera de la transacción, WebSphere eXtreme Scale volverá a intentar la petición. El cliente lo volverá a

intentar hasta que el tiempo transcurrido total excede el tiempo de espera de la transacción. De esta forma, el tiempo máximo de bloque de cliente es el tiempo de espera de la transacción además del tiempo de espera de petición de ORB.

El ORB se bloqueará hasta que la pila TCP cierre el socket de la caída de tensión, si no se ha especificado el tiempo de espera de petición ORB. Este tiempo depende del ajuste de la pila TCP y podrían ser horas en función de TCP FIN_WAIT y otros parámetros relacionados.

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
  ```

Uso de JMS para distribuir los cambios de transacciones

Utilice JMS (Java Message Service) para los cambios distribuidos 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 máquinas 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

Guía de administración

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*.

Capítulo 6. Seguridad

WebSphere eXtreme Scale es un sistema de almacenamiento en memoria caché distribuido. Si desea asegurar el acceso para proteger los datos, puede habilitar la seguridad e integrarla con proveedores de seguridad 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

© Copyright IBM Corp. 2009, 2009

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 7. Proceso de transacciones

WebSphere eXtreme Scale utiliza las transacciones como su mecanismo para la interacción con datos.

Sesiones y transacciones

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.

Ventajas de las transacciones

Mediante el uso de transacciones, puede:

- Retrotraer cambios si se produce una excepción o si la lógica empresarial necesita deshacer los cambios de estado.
- 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.

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.

Visión general de la transacción

Utilice transacciones por las siguientes razones:

- Para evitar que una hebra sufra cambios simultáneos.
- Para aplicar varios cambios como una unidad atómica durante la confirmación.
- Para implementar un ciclo de vida para bloqueos en cambios.

© Copyright IBM Corp. 2009, 2009

- Para actuar como unidad de réplica.

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.

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.

Atributo CopyMode

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.

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.

Modalidad de bloqueo y transacciones

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" />
```

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.

Estrategias de bloqueo

Las estrategias de bloqueo pueden ser de tipo pesimista, optimista o sin bloqueo. Para elegir la estrategia de bloqueo, debe conocer el porcentaje de cada tipo de operaciones que realizará, si utilizará un cargador, etc.

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 144.

Capítulo 8. Guías de aprendizaje

Puede utilizar las guías de aprendizaje para empezar con una funciones WebSphere eXtreme Scale determinadas.

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.

Por qué y cuándo se efectúa esta tarea

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:

© Copyright IBM Corp. 2009, 2009

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.

1. Cree la entidad de cliente, que se utiliza para almacenar detalles de cliente independientemente del objeto Order. A continuación se muestra un ejemplo de una entidad de cliente:

   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 149. 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 EntityManager 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.

Por qué y cuándo se efectúa esta tarea

Figura 40. 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.

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 un 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.

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.

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.

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.

Por qué y cuándo se efectúa 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.

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 160
   - Cómo crear un índice que la consulta pueda utilizar
3. "Guía de aprendizaje de ObjectQuery - Paso 3" en la página 160
   - 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 162
   - 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.

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 160 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 158 antes de continuar con este paso de la guía de aprendizaje.

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 utilizará automáticamente el índice cuando esté allí. Esto 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.

Por qué y cuándo se efectúa 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.

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 160 antes de continuar con el paso actual.

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 de Java SE - Página principal

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.

Por qué y cuándo se efectúa 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

1. "Guía de aprendizaje de seguridad Java SE - Paso 1" en la página 165
   - 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 168
   - 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 175
   - Uso de la política de autorización JAAS
4. "Guía de aprendizaje de seguridad de Java SE - Paso 4" en la página 178
   - 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.

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 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,
// sea para uso interno propio del cliente o para su redistribución por parte del cliente, como parte de 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 de 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 165.

Por qué y cuándo se efectúa 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.

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,
// sea para uso interno propio del cliente o para su redistribución por parte del cliente, como parte de 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 168 antes de llevar a cabo esta tarea.

Por qué y cuándo se efectúa 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 ofrecer distintos privilegios 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 permiso representadas por distintas clases de permisos. Este tema muestra MapPermission. Para obtener una categoría completa de permisos, vea la consulta de autorización de cliente.

En eXtreme Scale, la clase com.ibm.websphere.objectgrid.security.MapPermission representa permisos para los recursos de eXtreme Scale, específicamente los métodos de 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 distintos permisos de correlación para diferentes métodos. Si los permisos requeridos no se conceden al cliente, se produce una excepción AccessControlException.

Esta guía de aprendizaje muestra cómo utilizar la autorización JAAS para otorgar accesos a correlaciones de autorizaciones para usuarios distintos.

1. **Habilitar autorización de eXtreme Scale** Para habilitar la autorización en ObjectGrid es necesario establecer securityEnabled en true para ese determinado ObjectGrid en el archivo XML. La seguridad en ObjectGrid significa autorización. Utilice los siguientes mandatos para crear un nuevo XML de ObjectGrid con la seguridad habilitada.
   a. cd objectgridRoot/bin
   b. cp SimpleApp.xml SecureSimpleApp.xml

   A continuación, añada securityEnabled="true" en el nivel de ObjectGrid tal como se muestra en el siguiente 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" securityEnabled="true">
            <backingMap name="customer" readOnly="false" copyKey="true"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

2. **Definir política de autorización** Recuerde que en la sección de autenticación previa al cliente del paso anterior hemos creado tres usuarios en el almacén de claves: cashier, manager y administrator. En este ejemplo, mostraremos que 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 og_auth.policy 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 "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 tendrá 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 cashier tendrá 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.

   Puede iniciar un servidor de contenedor seguro utilizando el siguiente script:

c. Vuelva al directorio bin.

d.

   - 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 `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.

Ahora utilizamos el mismo mandato que en el paso anterior de la guía de aprendizaje:

a. Vaya al directorio bin como se ha indicado anteriormente.

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", utilizamos el usuario "cashier" para iniciar la aplicación cliente.

c. Vaya al directorio bin una vez más.

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
```

Esto sucede 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*.

Paso siguiente

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 175 antes de llevar a cabo esta tarea.

Por qué y cuándo se efectúa 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.

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 175.

   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 168.

     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`.

Capítulo 9. Glosario

Este glosario incluye los términos y las definiciones para WebSphere eXtreme Scale.

En este glosario se utilizan las siguientes referencias cruzadas:

1. Véase remite al lector de un término a una sinónimo preferido, o de un acrónimo o abreviatura a la forma completa definida.
2. Véase también remite al lector a un término relacionado o en contraste.

Para consultar los glosarios de otros productos IBM, vaya a www.ibm.com/software/globalization/terminology.

Acuerdo de nivel de servicio (SLA) . Un contrato entre un cliente y un proveedor de servicios que especifica las expectativas para el nivel de servicio con respecto a la disponibilidad, el rendimiento y otros objetivos mensurables.

administrador. La persona responsable de las tareas administrativas tales como la autorización de accesos y la gestión de contenidos. Los administradores pueden también otorgar los niveles de autoridad a los usuarios.

administrador de seguridad. La persona que controla el acceso a los datos de la empresa y a las funciones de programa.

afinidad de sesiones. Un método de configurar las aplicaciones de modo que un cliente siempre está conectado al mismo servidor. Estas configuraciones inhabilitan la gestión de cargas de trabajo después de la conexión inicial obligando a una solicitud de cliente a ir siempre al mismo servidor.

agente. Programa que realiza una acción en nombre de un usuario u otro programa sin la intervención del usuario o en una planificación regular, e informa de los resultados al usuario o programa.

agente de nodo. Un agente administrativo que gestiona todos los servidores de aplicaciones de un nodo y que representa el nodo de la célula de gestión.

alias de autenticación. Un alias que autoriza el acceso a los adaptadores de recursos y los orígenes de datos. Un alias de autenticación contiene datos de autenticación, como ID de usuario y contraseña.

almacén de certificados de colecciones. Una colección de certificados intermedios o listas de revocación de certificados, CRL, que utilizan una vía de acceso a certificados para crear una cadena de certificados para su validación.

almacén de datos persistente. Almacenamiento no volátil para datos de sucesos, como un sistema de base de datos, que se mantiene a través de los límites de sesión y que continúa existiendo después de la ejecución del programa o proceso que la ha creado.

alta disponibilidad (HA) . Que pertenece a un sistema en clúster que se ha vuelto a configurar cuando se producen anomalías de nodo o daemon, de forma que las cargas de trabajo se pueden redistribuir a los nodos restantes del clúster.

ámbito.

1. Una especificación del límite dentro del que se pueden utilizar los recursos del sistema.

2. En los servicios web, una propiedad que identifica el ciclo de vida del objeto que presta servicio a la solicitud de invocación.

analista de sistemas. Un especialista responsable de traducir requisitos de negocio en definiciones y soluciones del sistema.

APAR. Véase informe autorizado de análisis de programa.

API. Véase interfaz de programación de aplicaciones.

© Copyright IBM Corp. 2009, 2009

API de JavaMail. Una infraestructura independiente de la plataforma y del protocolo que permite crear aplicaciones cliente de correo basadas en Java.

API de Java para XML (JAX) . Un conjunto de API basadas en Java para manejar distintas operaciones que impliquen datos definidos mediante XML (lenguaje de códigos ampliable.)

aplicación. Uno o más programas de sistema o componentes de software que proporcionan una función de soporte directo de un proceso o procesos de negocio específicos.

aplicación cliente. Una aplicación, que se ejecuta en una estación de trabajo y está enlazada con un cliente, que da acceso a la aplicación para poner servicios en cola en un servidor.

aplicación Java EE. Cualquier unidad desplegable de funcionalidad Java EE. Esta unidad puede ser un único módulo o un grupo de módulos empaquetados en un archivo EAR con un descriptor de despliegue de aplicación Java EE. (Sun)

archivador empresarial (EAR) . Tipo especializado de archivo JAR, definido por el estándar Java EE, utilizado para desplegar aplicaciones Java EE en servidores de aplicaciones Java EE. Un archivo EAR contiene componentes EJB, un descriptor de despliegue y archivos WAR (archivador web) para aplicaciones web individuales. Véase también archivador web.

archivador Java. Formato de archivo comprimido para almacenar todos los recursos necesarios para instalar y ejecutar un programa Java en un solo archivo. Véase también archivador web, archivador empresarial.

archivo de almacén de confianza . Archivo de base de datos de claves que contiene las claves públicas de una entidad de confianza.

archivo de clase. Un archivo fuente Java compilado.

archivo de definición de build. Archivo XML que identifica los componentes y características de un paquete de instalación personalizado (CIP).

archivo de exportación.

1. Un archivo creado durante el proceso de desarrollo para las operaciones de entrada que contienen los valores de configuración para el proceso entrante.

2. El archivo que contiene los datos que se han exportado.

archivo delimitado por comas. Un archivo cuyos registros contienen campos que están separados por una coma.

archivo JAR. Un archivo Java de archivado. Véase también archivador web, archivador empresarial.

archivo JAR EJB. Archivador Java que contiene un módulo EJB. (Sun)

archivo Java. Archivo fuente editable (con la extensión .java) que se puede compilar en bytecode (archivo .class).

archivo JSP. Archivo HTML de script con una extensión .jsp que permite la inclusión de contenido dinámico en páginas web. Un archivo JSP se puede solicitar directamente como un URL, llamado por un servlet o desde una página HTML.

área del editor. En Eclipse área de la ventana del entorno de trabajo donde se abren los archivos para editarlos.

arreglo temporal. Un arreglo certificado que está generalmente disponible para todos los clientes entre paquetes de arreglos planificados regularmente o entre releases. Véase también fixpack.

Arreglo temporal de programa (PTF) . Para los productos de System i, System p y System z, un arreglo probado por IBM que se pone a disposición de todos los clientes. Véase también fixpack.

asíncrono. Relativo a sucesos que no están sincronizados en el tiempo o que no se producen a intervalos de tiempo regulares o predecibles.

atributo global. En XML, un atributo declarado como hijo del elemento de esquema en lugar de como parte de una definición de tipo complejo. Es posible hacer referencia a los atributos globales en uno o varios modelos de contenido utilizando el atributo ref.

autenticación. Un servicio de seguridad que proporciona la prueba de que un usuario de un sistema es realmente quien dice ser. Los mecanismos habituales para implementar este servicio son contraseñas y firmas digitales. La autenticación es distinta de la autorización; la autenticación no se ocupa de garantizar ni de denegar el acceso a los recursos del sistema.

autónomo. Independiente de cualquier otro dispositivo, programa o sistema. En un entorno de red, una máquina autónoma accede localmente a todos los recursos necesarios.

autorización. El proceso para otorgar a un usuario, sistema o proceso, un acceso completo o restringido a un objeto, recurso o función.

base de datos local. Base de datos que está ubicada en la estación de trabajo en uso.

bean. Definición o instancia de un componente JavaBeans. Véase también JavaBeans, enterprise bean.

bean de entidad. En programación EJB, un enterprise bean que representa los datos persistentes mantenidos en una base de datos. Cada bean de entidad transporta su propia identidad. (Sun)

bean de mandato. Un proxy que puede invocar una sola operación utilizando un método execute().

Bean gestionado (MBean) . En la especificación de JMX (Java Management Extensions), los objetos Java que implementan recursos y su instrumentación.

Bean Scripting Framework. Una arquitectura para incorporar funciones de lenguaje de scripts en las aplicaciones Java.

biblioteca.

1. Colección de elementos de modelo, que incluyen elementos de negocio, procesos, tareas, recursos y organizaciones.

2. Proyecto que se utiliza para el desarrollo, gestión de versiones y organización de los recursos compartidos. Sólo se puede crear y almacenar un subconjunto de tipos de artefactos en una biblioteca como, por ejemplo, objetos empresariales e interfaces.

bifurcación. Un elemento de proceso que hace copias de sus entradas y las reenvía en paralelo por varias vías de proceso distintas.

bloqueo. Procedimiento que impide que un proceso de aplicación perciba los cambios no confirmados que realiza otro proceso de aplicación, y que además impide que un proceso de aplicación actualice los datos a los que está accediendo otro proceso. Un bloqueo garantiza la integridad de los datos al impedir el acceso simultáneo de varios usuarios a datos incoherentes.

bloqueo actualizable. Bloqueo que identifica el intento de actualizar una entrada de la memoria caché al utilizar un bloqueo pesimista.

bloqueo compartido. Bloqueo que limita a los procesos de aplicaciones de ejecución simultánea a operaciones de solo lectura de los datos de la base de datos.

bloqueo exclusivo. Bloqueo que impide que los procesos de aplicaciones de ejecución simultánea accedan a los datos de la base de datos. Véase también bloqueo compartido.

bloqueo pesimista. Estrategia de bloqueo según la cual un bloqueo tiene lugar entre el tiempo durante el que se selecciona una fila y el tiempo durante el cual se realiza una operación de actualización o supresión en dicha fila.

BMP. Véase persistencia gestionada por bean.

BMT. Véase transacción gestionada por bean.

bucle. Una secuencia de instrucciones realizada repetidamente.

bucle do while. Bucle que repite la misma secuencia de actividades siempre que se cumpla una condición. A diferencia de un bucle while, un bucle do while comprueba la condición al final del bucle. Eso significa que la secuencia de actividades siempre se ejecuta al menos una vez.

bucle for. Bucle que repite la misma secuencia de actividades el número de veces especificado.

bucle while. Bucle que repite la misma secuencia de actividades siempre que se cumpla una condición. El bucle while comprueba su condición al principio de cada bucle. Si la condición es falsa desde el principio, la secuencia de actividades que contiene el bucle nunca se ejecuta.

bus de servicio empresarial (ESB). Infraestructura de conectividad flexible para integrar aplicaciones y servicios; ofrece un enfoque flexible y gestionable para la implementación de SOA (Service-Oriented Architecture).

bytecode. Código independiente del sistema generado por el constructor Java y ejecutado por el intérprete Java. (Sun)

calidad de servicio (QoS) . Conjunto de características de comunicación que requiere una aplicación. La calidad de servicio (QoS) define una prioridad de transmisión específica, un nivel de fiabilidad de ruta y un nivel de seguridad.

canal de contenedores web. Un tipo de canal en una cadena de transporte que crea un puente en la cadena de transporte entre un canal de entrada HTTP y un servlet o motor JSP (JSP).

canal SSL. Un tipo de canal de una cadena de transporte que asocia un repertorio de configuración SSL (Secure Sockets Layer) a la cadena de transporte.

canal TCP. Un tipo de canal de una cadena de transporte que proporciona a las aplicaciones cliente conexiones persistentes en una LAN (red de área local).

cargador. Componente que lee datos y los graba en un almacén persistente.

cargador de clases. Componente de una máquina virtual Java (JVM) responsable de buscar y cargar archivos de clase. Un cargador de clases afecta el empaquetamiento de aplicaciones y el comportamiento de aplicaciones empaquetadas desplegadas en servidores de aplicaciones.

carpeta. Un contenedor utilizado para organizar objetos.

catálogo. Un contenedor que, dependiendo del tipo de contenedor, guarda procesos, datos, recursos, organizaciones o informes en el árbol de proyecto.

categoría. Un contenedor utilizado en un diagrama de estructura para agrupar los elementos basados en un atributo o calidad compartido.

célula.

1. Grupo de procesos gestionados que están federados en el mismo gestor de despliegue y puede incluir grupos principales de alta disponibilidad.

2. Uno (o más) procesos que aloja componentes de ejecución. Cada uno tiene uno o más grupos principales a los que se ha asignado un nombre.

certificado de firmante. La entrada del certificado de confianza que generalmente está en un archivo de almacén de confianza.

certificado digital. Un documento electrónico que se utiliza para identificar a un individuo, servidor, empresa u otro tipo de entidad y para asociar una clave pública a la entidad. Los certificados digitales los emiten las autoridades certificadoras y tienen la firma digital de éstas.

chasis. El bastidor de metal en el que se montan distintos componentes electrónicos.

ciclo de vida. Pasada completa por las cuatro fases del desarrollo de software: inicio, elaboración, construcción y transición.

CIP. Véase paquete de instalación personalizado.

clase. En programación o diseño orientado a objetos, un modelo o plantilla que se puede utilizar para crear objetos con una definición común y propiedades, operaciones y comportamientos comunes. Un objeto es una instancia de una clase.

clase de bean. En Enterprise JavaBeans (EJB), clase Java que implementa una clase javax.ejb.EntityBean o una clase javax.ejb.SessionBean.

clase Java. Clase escrita en lenguaje Java.

clasificador. Un atributo especializado que se utiliza para agrupar y asignar código de color a los elementos de proceso

clave.

1. Valor matemático criptográfico que se utiliza para firmar, verificar, cifrar o descifrar digitalmente los mensajes.

2. Información que caracteriza e identifica de forma exclusiva la entidad real de cuyo seguimiento se encarga un contexto de supervisión.

clave primaria.

1. Un objeto que identifica de forma exclusiva un bean de entidad de un tipo concreto.

2. En una base de datos relacional, una clave que identifica de forma exclusiva una fila de una tabla de base de datos.

cliente. Un programa de software o un sistema que solicita servicios a un servidor. Véase también host.

cliente de aplicaciones ligero. Un tiempo de ejecución de aplicación Java, ligero y descargable, que puede interactuar con enterprise beans.

cliente ligero. Cliente que tiene instalado poco software o ninguno pero que tiene acceso a software gestionado y entregado por los servidores de red que tiene conectados. Un cliente ligero es una alternativa a un cliente de funciones completas como por ejemplo una estación de trabajo.

cliente/servidor. Relativo al modelo de interacción en el proceso de datos distribuido en el que un programa de un sistema envía una solicitud a un programa de otro sistema y espera una respuesta. El programa solicitante se denomina cliente, y el programa que responde se denomina servidor.

Cloudscape. Sistema ORDBMS (Object-Relational Database Management System), completamente Java, que puede intercalarse.

clúster. Un grupo de servidores de aplicaciones que colaboran para el equilibro de cargas de trabajo y la migración tras error.

clúster de proxy. Un grupo de servidores proxy que distribuye las solicitudes HTTP a través del clúster.

clúster de servidores. Un grupo de servidores que generalmente están en máquinas físicas diferentes y que tienen configuradas las mismas aplicaciones pero que funcionan como un servidor lógico individual.

clúster dinámico. Un clúster de servidores que utiliza pesos para equilibrar dinámicamente las cargas de trabajo de los miembros del clúster, basándose en la información de rendimiento recogida de los miembros del clúster.

código abierto. Relativo al software cuyo código fuente está disponible públicamente para utilizar o modificar. Normalmente el software de código abierto se desarrolla como colaboración pública y se pone a disposición pública, aunque su uso y redistribución pueden estar sujetos a restricciones de licencia. Linux es un ejemplo muy conocido de software de código abierto.

código de despliegue. Código adicional que permite que el código de implementación de beans escrito por un desarrollador de aplicaciones trabaje en un entorno de tiempo de ejecución EJB determinado. El código de despliegue puede generarse con herramientas que proporcione el proveedor del servidor de aplicaciones.

componente.

1. Objeto o programa reutilizable que realiza una función específica y se utiliza con otros componentes y aplicaciones.

2. En Eclipse, uno o varios plug-ins que funcionan conjuntamente para proporcionar un conjunto discreto de funciones.

componente web. Un servlet, archivo JavaServer Pages (JSP) o un archivo HTML (HyperText Markup Language). Uno o más componentes web componen un módulo web.

consulta.

1. Solicitud de información de una base de datos de acuerdo con condiciones específicas: por ejemplo, una solicitud de una lista de todos los clientes de una tabla de clientes cuyos balances son superiores a 1000 dólares.

2. Solicitud reutilizable de información sobre uno o más elementos del modelo.

consulta EJB. En lenguaje de consulta EJB, serie que contiene una cláusula SELECT opcional que especifica los objetos EJB que se deben devolver, una cláusula FROM que indica las colecciones de beans, una cláusula WHERE opcional que contiene predicados de búsqueda en las colecciones, una cláusula ORDER BY opcional que especifica la ordenación de la colección de resultados, y parámetros de entrada que corresponden a los argumentos del método buscador.

consulta SQL. Componente de determinadas sentencias SQL que especifica una tabla de resultados.

contenedor EJB. Un contenedor que implementa el contrato del componente EJB de la arquitectura Java EE. Este contrato especifica un entorno de tiempo de ejecución para los enterprise beans que incluye seguridad, simultaneidad, gestión de ciclo de vida, transacción, despliegue y otros servicios. (Sun)

contenedor web. Un contenedor que implementa el contrato del componente web de la arquitectura Java EE. (Sun)

contexto de EJB. En enterprise beans, objeto que permite que un enterprise bean invoque servicios proporcionados por el contenedor y obtenga información sobre el proceso que efectúa la llamada de un método invocado por el cliente. (Sun)

contienda de hebras. Una condición en la que una hebra está esperando un bloqueo o un objeto mantenido por otra hebra.

convertidor. En programación Enterprise JavaBeans (EJB), una clase que convierte una representación de base de datos en un tipo de objeto y viceversa.

correlación.

1. Una estructura de datos que correlaciona las claves con los valores.

2. Un archivo que define la transformación entre orígenes y destinos.

3. En el entorno de desarrollo EJB, la especificación de cómo los campos de persistencia gestionada por contenedor de un enterprise bean se corresponden a columnas en una tabla de base de datos relacional u otro almacenamiento persistente.

corriente de registro cronológico de errores. Un flujo continuado de información sobre errores que se transmite utilizando un formato definido previamente.

cortafuegos. Una configuración de red, normalmente tanto hardware como software, que impide la entrada y salida del tráfico no autorizado de una red segura.

crear una instancia. Representar una abstracción con una instancia concreta.

create, método. En enterprise beans, método definido en la interfaz inicial e invocado por el cliente para crear un enterprise bean. (Sun)

credencial. En la infraestructura JAAS (Java Authentication and Authorization Service), clase de asunto que posee atributos relacionados con la seguridad. Estos atributos pueden contener información utilizada para autenticar el sujeto en nuevos servicios.

cuadrícula de datos. Sistema que sirve para acceder a terabytes o petrabytes de datos.

cuadrícula de eXtreme Scale. Patrón que se usa para interactuar con eXtreme Scale cuando todos los datos y clientes están en un proceso.

cualificador. Un elemento simple que aporta a otro elemento simple o compuesto genérico un significado específico. Los cualificadores se utilizan en la correlación de apariciones únicas o múltiples. Un cualificador también se puede utilizar para denotar el espacio de nombres utilizado para interpretar la segunda parte del nombre, al que se hace referencia normalmente como el ID.

cuello de botella . Un punto del sistema en el que la contención de un recurso está afectando al rendimiento.

daemon. Programa que se ejecuta de manera desatendida para realizar funciones de forma continua o periódica, como el control de la red.

datos de hora de construcción. Los objetos que no son utilizados por el conversor, como los estándares EDI, los tipos de documento de datos orientados a registro, y los mapas.

DB2. Familia de programas bajo licencia de IBM para la gestión de bases de datos relacionales.

definición de documento DTD. Una descripción o diseño de un documento XML basado en un DTD XML.

definición de tipo de documento (DTD) . Reglas que especifican la estructura de una clase concreta de documentos SGML o XML. La DTD define la estructura con elementos, atributos y notaciones y establece restricciones para la utilización de cada elemento, atributo y notación en la clase particular de los documentos.

derivación. En programación orientada a objetos, el refinamiento o ampliación de una clase a partir de otra.

desalojador. Componente que controla la pertenencia de las entradas en cada instancia de BackingMap. Las memorias caché dispersas pueden utilizar desalojadores para eliminar automáticamente los datos de la memoria caché sin que ello afecte a la base de datos.

desarrollo ascendente. En los servicios web, el proceso de desarrollar un servicio a partir de un artefacto existente como por ejemplo un bean Java o un enterprise bean en vez de hacerlo a partir de un archivo WSDL (lenguaje de descripción de servicios web).

descriptor de despliegue. Archivo XML que describe cómo desplegar un módulo o aplicación especificando opciones de configuración y contenedor. Por ejemplo, un descriptor de despliegue de EJB pasa información a un contenedor EJB acerca de cómo gestionar y controlar un enterprise bean.

descubrimiento automático. El descubrimiento de artefactos de servicio en un sistema de archivos, registro externo u otra fuente.

deserialización. Un método para convertir una variable serializada en datos de objeto.

desplegable. Consulte desplegable.

desplegar. Para colocar archivos o instalar software en un entorno operativo. En Java Platform, Enterprise Edition (Java EE), esto supone crear un descriptor de despliegue adecuado al tipo de aplicación que se va a desplegar.

destino. Un punto de salida que se utiliza para entregar documentos a un sistema de programa de fondo o a un socio comercial.

destino de instalación. Sistema en el que se instalan los paquetes de instalación seleccionados.

directorio de despliegue. El directorio en el que se encuentran la configuración de servidor publicado y la aplicación web en la máquina en la que se instala el servidor de aplicaciones.

directorio LDAP. Tipo de repositorio que almacena información sobre personas, organizaciones y otros recursos, y al que se accede mediante el protocolo LDAP. Las entradas del repositorio se organizan en una estructura jerárquica, y en algunos casos dicha estructura refleja la estructura o geografía de una organización.

disparar. En la programación orientada a objetos, causar una transición de estado.

disponibilidad.

1. La condición que permite a los usuarios acceder y utilizar sus aplicaciones y datos.

2. Los periodos de tiempo durante los que se puede acceder a un recurso. Por ejemplo, una empresa puede estar disponible los días laborables de 9 AM a 5 PM y los sábados de 9 AM a 3 PM.

distribuidor de IP. Dispositivo ubicado entre las solicitudes entrantes de los usuarios y los nodos del servidor de aplicaciones que redirecciona las solicitudes a través de los nodos.

DMZ. Véase zona desmilitarizada.

DNS. Véase Sistema de nombres de dominio.

dominio. Un objeto, icono o contenedor que contiene otros objetos que representan los recursos de un dominio. Se puede utilizar el objeto de dominio para manejar estos recursos.

DTD. Véase definición de tipo de documento.

EAR. Véase archivador empresarial.

Eclipse. Iniciativa de código abierto que proporciona a los ISV y a otros desarrolladores de herramientas una plataforma estándar para elaborar herramientas de desarrollo de aplicaciones compatibles con conectores.

edición. Generación de despliegues sucesivos de un determinado conjunto de artefactos con versión.

EJB. Véase Enterprise JavaBeans.

elemento de componente. Una entidad de un componente en la que puede establecerse un punto de interrupción, como por ejemplo una actividad o un fragmento de código Java en un proceso de negocio o un primitivo de mediación o un nodo en un flujo de mediación.

elemento en espera. Una hebra que espera una conexión.

elemento global. En XML, un elemento declarado como hijo del elemento de esquema en lugar de como parte de una definición de tipo complejo. Es posible hacer referencia a los elementos globales en uno o varios modelos de contenido utilizando el atributo ref.

en desuso. Relativo a una entidad como, por ejemplo, un elemento o característica de programación a la que se da soporte pero que ya no se recomienda y puede pasar a ser obsoleta.

enlace de ámbito de célula. Ámbito de enlace en el que el enlace no es específico de ningún nodo o servidor, ni está asociado con ellos. Este tipo de enlace de nombres se crea en el contexto de raíz de persistencia de una célula.

enlace de datos JMS. Un enlace de datos que proporciona una correlación entre el formato utilizado por un mensaje JMS externo y la representación SDO (Service Data Object) utilizada por un módulo SCA (Service Component Architecture).

enlace de protocolo. Un enlace que permite al bus de servicio empresarial procesar mensajes independientemente del protocolo de comunicaciones.

en sentido ascendente. Dícese de la dirección de flujo, que va desde el inicio del proceso (arriba) hacia el final del proceso (abajo).

en sentido descendente. Dícese de la dirección de flujo, que va del primer nodo del proceso (arriba) al último nodo del proceso (abajo).

enterprise bean. Componente que implementa una tarea o una entidad de negocio y reside en un contenedor EJB. Los beans de entidad, beans de sesión y beans controlados por mensajes son enterprise beans. (Sun) Véase también bean.

Enterprise JavaBeans (EJB). Una arquitectura de componentes definida por Sun Microsystems para el desarrollo y el despliegue de aplicaciones de nivel empresarial, distribuidas y orientadas a objetos (Java EE).

entidad.

1. Clase Java sencilla que representa una fila en una tabla de base de datos o una entrada en una correlación.

2. En lenguajes de marcación como, por ejemplo, XML, colección de caracteres a la que se puede hacer referencia como una unidad, por ejemplo, para incorporar texto que se repite a menudo o caracteres especiales en un documento.

entorno. Un colección denominada de los recursos lógicos y físicos utilizados para soportar el rendimiento de una función.

entorno de despliegue. Una colección de clústeres, servidores y middleware configurados que colaboran para proporcionar un entorno para alojar módulos de software. Por ejemplo, un entorno de despliegue puede incluir un host para destinaciones de mensajes, un procesador u ordenador de sucesos de negocio y programas administrativos.

entorno de tiempo de ejecución Java. Un subconjunto de un Java developer kit que contiene los programas ejecutables básicos y los archivos que constituyen la plataforma Java estándar. JRE incluye la máquina virtual Java (JVM), las clases básicas y los archivos de soporte.

equilibrio de carga. Supervisión de los servidores de aplicaciones y gestión de la carga de trabajo en los servidores. Si un servidor sobrepasa su carga de trabajo, las solicitudes se redirigen a otro servidor con más capacidad.

error. Discrepancia entre un valor o condición calculada, observada o medida y el valor o condición cierta, especificada o teóricamente correcta.

ESB. Véase bus de servicio empresarial.

escalabilidad. La capacidad de ampliación de un sistema a medida que se van añadiendo procesadores, capacidad de memoria o capacidad de almacenamiento.

escucha. Un programa que detecta solicitudes entrantes e inicia el canal asociado.

escucha de punto final. El punto o la dirección en que un bus de integración de servicios recibe los mensajes entrantes de un servicio web.

espacio de nombres. Un contenedor lógico en el que todos los nombres son exclusivos. El identificador exclusivo para un artefacto se compone del espacio de nombres y el nombre local del artefacto.

espacio de trabajo.

1. Un directorio en el disco que contiene todos los archivos de proyecto, así como información como las preferencias.

1. Repositorio temporal de información de configuración que utilizan los clientes administrativos.

3. En Eclipse, la colección de proyectos y otros recursos que el usuario está desarrollando actualmente en el entorno de trabajo. Los metadatos relativos a estos recursos residen en un directorio del sistema de archivos; los recursos pueden residir en el mismo directorio.

esqueleto. Andamiaje de una clase de implementación.

esquema URL. Un formato que contiene la referencia de otro objeto.

estático. Una palabra clave del lenguaje de programación Java que se utiliza para definir una variable como una variable de clase.

excepción. Una condición o suceso que no puede ser manejada por un proceso normal.

exportación. Una interfaz expuesta de un módulo SCA (Service Component Architecture) que ofrece un servicio de negocio al mundo exterior. Una exportación tiene un enlace que define cómo pueden acceder al servicio los solicitantes de servicio, por ejemplo, como servicio web.

expresión. Operando SQL o XQuery o una colección de operadores y operandos SQL o XQuery que proporcionan un único valor.

eXtreme Scale distribuido. Patrón de uso para interactuar con eXtreme Scale cuando existen servidores y clientes en varios procesos.

fábrica. En programación orientada a objetos, una clase que se utiliza para crear instancias de otra clase. Las fábricas se utilizan para aislar la creación de objetos de una clase en particular en un lugar, de modo que se puedan proporcionar nuevas funciones sin cambios drásticos de código.

fábrica EJB. Bean de acceso que simplifica la creación o la búsqueda de una instancia de enterprise bean.

fase de despliegue . Véase fase de despliegue.

fase de despliegue. Una fase que incluye una combinación de crear el entorno de host para las aplicaciones y el despliegue de éstas aplicaciones. Esto incluye resolver las dependencias de recursos de las aplicaciones, las condiciones operativas, los requisitos de capacidad y las restricciones de integridad y acceso.

fixpack. Una colección acumulativa de arreglos que se pone a disposición entre paquetes de renovación planificados, renovaciones de fábrica o releases. Se tiene previsto autorizar a los clientes para desplazarse a un nivel de mantenimiento específico. Véase también arreglo temporal.

formato binario. Representación de un valor decimal en el cual todos los campos deben tener entre 2 ó 4 bytes de longitud. El signo (+ o -) se encuentra en el bit del extremo izquierdo del campo, y el valor numérico se encuentra en

los bits restantes del campo. Los números positivos tienen un 0 en el bit de signo y tienen el formato verdadero. Los números negativos tienen un 1 en el bit de signo y tienen dos formatos complementarios.

fragmento. Instancia de una partición. Un fragmento puede ser primario o una réplica.

fuga de memoria. Efecto de un programa que mantiene referencias a objetos que ya no son necesarios y que, por lo tanto, se deben reclamar.

general. Relativo a la visualización de un grupo de objetos desde un nivel alto o abstracto.

gestión de carga de trabajo. La optimización de la distribución de las solicitudes de trabajo entrantes a los servidores de aplicaciones, los enterprise beans, los servlets y otros objetos que pueden procesar la solicitud de manera más eficaz.

gestor autónomo. Conjunto de componentes de software o hardware, configurados por políticas, que gestionan el comportamiento de otros componentes de software o hardware como lo haría una persona. Un gestor autónomo incluye un bucle de control que consta de componentes de supervisión, análisis, planificación y ejecución.

Gestor de carga de trabajo (WLM). Un componente de z/OS que proporciona la capacidad de ejecutar diversas cargas de trabajo a la vez dentro de una imagen de z/OS o a lo largo de diversas imágenes.

gestor de despliegue. Un servidor que gestiona operaciones para un grupo lógico o una célula de otros servidores.

GIOP. Véase protocolo Inter-ORB general.

global.

1. Que pertenece a un elemento que está disponible para cualquier proceso del espacio de trabajo. Un elemento global aparece en el árbol de proyecto y se puede utilizar en varios procesos. Las tareas, procesos, repositorios y servicios pueden ser globales (cualquier proceso del proyecto puede hacer referencia a ellos) o locales (específicos de un solo proceso).

2. Que pertenece a la información disponible para más de un programa o subrutina.

grupo.

1. Colección de usuarios que pueden compartir autorizaciones de acceso sobre los recursos protegidos.

2. Un conjunto de documentos relacionados dentro de un intercambio. Un intercambio puede contener de cero a muchos grupos.

3. En algunos lugares, dos o más personas agrupadas por pertenecer a un lugar.

grupo HA. Una colección de uno o más miembros utilizada para proporcionar alta disponibilidad para un proceso.

HA. Véase alta disponibilidad.

hebra. Una corriente de instrucciones del sistema que controla un proceso. En algunos sistemas operativos, una hebra es la unidad de operación más pequeña en un proceso. Algunas hebras pueden ejecutarse simultáneamente llevando a cabo distintos trabajos.

herencia. Técnica de programación orientada a objetos en la que las clases existentes se usan como base para crear otras clases. Mediante herencia, los elementos más específicos incorporan la estructura y el comportamiento de los elementos más generales.

herencia EJB. Tipo de herencia en la que un enterprise bean hereda propiedades, métodos y atributos del descriptor de control a nivel de método de otro enterprise bean que reside en el mismo grupo.

High Availability Manager. Una infraestructura en la que se determina la pertenencia de los miembros al grupo principal y se comunica el estado entre los miembros del grupo principal.

host.

1. Sistema que está conectado a una red y proporciona un punto de acceso a esa red. El host puede ser un cliente, un servidor, o ambos simultáneamente.

2. En los perfiles de rendimiento, una máquina que es propietaria de procesos de los que se están creando perfiles. Véase también servidor.

host virtual. Una configuración que permite que un host parezca varios hosts. Los recursos asociados con un host virtual no pueden compartir datos con recursos asociados con otro host virtual, incluso si los hosts virtuales comparten la misma máquina física.

HTTPS.

1. Véase HTTP sobre SSL.

2. Véase protocolo seguro de transferencia de hipertexto.

HTTP sobre SSL (HTTPS) . Un protocolo web para transacciones seguras que cifra y descifra solicitudes de página de usuario y páginas devueltas por el servidor web.

IDE. Siglas de Integrated Development Environment. Véase Entorno de Desarrollo Integrado.

identificador de instancia global. Identificador exclusivo globalmente que lo genera la aplicación o el emisor y que se utiliza como clave primaria para la identificación del suceso.

Identificador universal de recursos (URI) .

1. Una serie compacta de caracteres para identificar un recurso abstracto o físico.

2. Una dirección exclusiva que se utiliza para identificar el contenido en la web como, por ejemplo, una página de texto, un vídeo o un clip de sonido, una imagen estática o animada o un programa. El formato más común del URI es la dirección de página web, que es una forma o subconjunto concreto de URI denominado localizador universal de recursos (URL). Un URI describe habitualmente cómo acceder al recurso, el sistema que contiene el recurso y el nombre del recurso (un nombre de archivo) en el sistema.

IIOP. Véase protocolo Inter-ORB de Internet.

importación.

1. Un artefacto de desarrollo que importa un servicio que es externo a un módulo.

2. El punto en el que un módulo SCA accede a un servicio externo (un servicio fuera del módulo SCA) como si fuera local. Una importación define las interacciones entre el módulo SCA y el proveedor de servicios. Una importación tiene un enlace y una o más interfaces.

índice. Conjunto de punteros que están ordenados lógicamente según los valores de una clave. Los índices proporcionan acceso rápido a los datos e imponen la exclusividad de los valores clave de las filas de la tabla.

Information Center. Recopilación de información que proporciona soporte a los usuarios de uno o más productos, puede iniciarse independientemente desde el producto e incluye una lista de temas para la navegación y un motor de búsqueda.

informe autorizado de análisis de programa (APAR) . Una solicitud de corrección de un defecto en un release soportado de un programa proporcionado por IBM.

instalación silenciosa. Instalación que no envía mensajes a la consola sino que almacena los mensajes y errores en archivos de registro. Una instalación silenciosa puede utilizar archivos de respuesta para la entrada de datos.

instancia. Una aparición específica de un objeto que pertenece a una clase.

instancia de componente. Un componente de ejecución que puede ejecutarse en paralelo con otras instancias del mismo componente.

Integrated Development Environment (IDE) . Conjunto de herramientas de desarrollo de software, como los editores del fuente, los compiladores y los depuradores, a las que se puede acceder desde una sola interfaz de usuario.

interfaz. Una colección de operaciones que se utilizan para especificar un servicio de una clase o componente.

interfaz de programación de aplicaciones (API) . Una interfaz que permite que un programa de aplicación escrito en un lenguaje de nivel alto pueda utilizar funciones o datos específicos del sistema operativo o de otro programa.

Interfaz Profiler de la máquina virtual Java (JVMPI). Una herramienta de perfiles que da soporte a la recopilación de información como, por ejemplo, datos sobre la recopilación de errores, y la API de JVM (máquina virtual Java) que ejecuta el servidor de aplicaciones.

Intermediario de solicitud de objetos (ORB) . En programación orientada a objetos, el software que sirve de intermediario al habilitar objetos de forma transparente para el intercambio de solicitudes y respuestas.

invocación. La activación de un programa o procedimiento.

IP. Siglas de Internet Protocol. Véase protocolo Internet.

JAAS. Véase Java Authentication and Authorization Service.

JAF. Véase JavaBeans Activation Framework.

Java. Lenguaje de programación orientado a objetos para código de interpretación portable que soporta la interacción entre objetos remotos. Java fue desarrollado y especificado por Sun Microsystems, Incorporated.

Java Authentication and Authorization Service (JAAS). En la tecnología Java EE, una API estándar para realizar operaciones basadas en seguridad. A través de JAAS, los servicios pueden autenticar y autorizar usuarios al tiempo que permiten que las aplicaciones sigan siendo independientes de las tecnologías subyacentes.

JavaBeans. Según la definición de Sun Microsystems para Java, modelo de componente portable, independiente de la plataforma y reutilizable. Véase también bean.

JavaBeans Activation Framework (JAF). Una ampliación estándar de la plataforma Java que determina los tipos de datos arbitrarios y las operaciones disponibles y que puede crear una instancia de bean para ejecutar los servicios pertinentes.

Java Database Connectivity (JDBC). Estándar del sector para la conectividad independiente de la base de datos entre la plataforma Java y una amplia gama de bases de datos. La interfaz JDBC proporciona una interfaz a nivel de llamada para el acceso a bases de datos basadas en SQL y basadas en XQuery.

Javadoc.

1. Una herramienta que analiza las declaraciones y los comentarios de documentación en un conjunto de archivos de origen y genera un conjunto de páginas HTML que describen clases, clases internas, interfaces, constructores, métodos y campos. (Sun)

2. Pertenece a la herramienta que analiza las declaraciones y los comentarios de documentación en un conjunto de archivos de origen y genera un conjunto de páginas HTML que describen las clases, clases internas, interfaces, constructores, métodos y campos.

Java EE. Véase Java Platform, Enterprise Edition.

Java EE Connector Architecture (JCA). Arquitectura estándar para conectar la plataforma Java EE a sistemas de información de empresa heterogéneos (EIS).

Java Management Extensions (JMX). Un medio de realizar la gestión mediante tecnología Java. JMX es una ampliación abierta y universal del lenguaje de programación Java para la gestión y se puede desplegar en todos los sectores en los que ésta sea necesaria.

Java Message Service (JMS). Interfaz de programas de aplicación que proporciona funciones de lenguaje Java para manejar mensajes.

Java Naming and Directory Interface (JNDI). Ampliación de la plataforma Java que proporciona una interfaz estándar para los servicios de denominación y directorio heterogéneos.

Java Platform, Enterprise Edition (Java EE). Un entorno para desarrollar y desplegar aplicaciones empresariales, definido por Sun Microsystems Inc. La plataforma Java EE consta de un conjunto de servicios, interfaces de programación de aplicaciones (API) y protocolos que proporcionan la funcionalidad para desarrollar aplicaciones basadas en la web de varios niveles. (Sun)

Java Platform, Standard Edition (Java SE). Plataforma central de la tecnología Java. (Sun)

JavaScript. Un lenguaje de creación de scripts web que se utiliza en navegadores y servidores web. (Sun)

JavaScript Object Notation. Un formato de intercambio de datos ligero que se basa en la notación literal de objetos de JavaScript. JSON es independiente del lenguaje de programación, sin embargo utiliza convenciones de lenguajes entre los que se incluyen C, C++, C#, Java, JavaScript, Perl, Python.

Java SE. Véase Java Platform, Standard Edition.

Java Secure Socket Extension (JSSE). Un paquete Java que permite las comunicaciones seguras a través de Internet. Implementa una versión Java de los protocolos SSL (Secure Sockets Layer) y TLS (Transport Layer Security) y da soporte al cifrado de datos, la autenticación del servidor, la integridad de mensajes y, opcionalmente, la autenticación de clientes.

Java SE Development Kit (JDK). El nombre del kit de desarrollo de software que Sun Microsystems proporciona para la plataforma Java.

JavaServer Pages (JSP). Tecnología de scripts del lado del servidor que permite que el código Java se intercale dinámicamente en las páginas web (archivos HTML) y se ejecute en el momento de servir la página, con objeto de devolver contenido dinámico a un cliente.

Java Specification Request (JSR). Una especificación propuesta formalmente para la plataforma Java.

JAX. Siglas de Java API for XML. Véase API de Java para XML.

JCA . Véase Java EE Connector Architecture.

JDBC. Véase Java Database Connectivity.

JDK. Véase Java SE Development Kit.

jerarquía de clases. Las relaciones entre las clases que comparten una única herencia.

JMS. Véase Java Message Service.

JMX. Véase Java Management Extensions.

JNDI. Véase Java Naming and Directory Interface.

JSP. Véase JavaServer Pages.

JSR. Siglas de Java Specification Request. Véase petición de especificación Java (JSR).

JSSE. Véase Java Secure Socket Extension.

JVM. Siglas de Java Virtual Machine. Véase máquina virtual Java (JVM).

JVMPI. Véase Java virtual machine Profiler Interface.

Jython. Una implementación del lenguaje de programación Python integrado en la plataforma Java.

kit de desarrollo de software (SDK) . Un conjunto de herramientas, API, documentación para ayudar en el desarrollo de software de un lenguaje de sistemas específico o de un entorno operativo determinado.

LDAP. Véase protocolo LDAP (Lightweight Directory Access Protocol).

lectura no permitida. Una solicitud de lectura que no implica ningún mecanismo de bloqueo. Esto significa que los datos que se pueden leer se podrían retrotraer más adelante y generar una incoherencia entre lo que se ha leído y lo que está en la base de datos.

lenguaje de códigos ampliable (XML) . Metalenguaje estándar para definir lenguajes de marcación basado en SGML (Standard Generalized Markup Language).

Lenguaje de consulta estructurado (SQL) . Un lenguaje estándar para definir y manipular los datos en una base de datos relacional.

lenguaje de mandatos Java. Un lenguaje de scripts para el entorno Java que se utiliza para crear contenidos web y controlar las aplicaciones Java.

Lightweight Directory Access Protocol (LDAP). Un protocolo abierto que utiliza TCP/IP para proporcionar acceso a directorios que admiten el modelo X.500 y que no acarrea los requisitos de recursos del protocolo más complejo DAP (Directory Access Protocol) de X.500. Por ejemplo, LDAP se puede utilizar para localizar personas, organizaciones y otros recursos de directorios de Internet o intranet.

Lightweight Third Party Authentication (LTPA). Un protocolo que utiliza la criptografía para dar soporte a la seguridad en un entorno distribuido.

línea de mandatos. La línea en blanco en una visualización donde se pueden introducir mandatos, números de opción o selecciones.

local.

1. Relativo a un dispositivo, archivo o sistema al que se accede directamente desde un sistema de usuario, sin el uso de una línea de comunicaciones.

2. Relativo a un elemento que sólo está disponible en su propio proceso.

Localizador universal de recursos (URL) . Dirección exclusiva de un recurso de información que sea accesible en una red como Internet. El URL incluye el nombre abreviado del protocolo utilizado para acceder al recurso de información y la información utilizada por el protocolo para localizar el recurso de información.

Lo que se ve es lo que se obtiene (WYSIWYG) . Capacidad de un editor para visualizar de forma continuada las páginas exactamente igual a como se imprimirán o representarán.

LTPA. Véase Lightweight Third Party Authentication.

manejador de excepciones. Un conjunto de rutinas que responde a una condición anómala. Un manejador de excepciones puede interrumpir y reanudar la normal ejecución de los procesos.

máquina virtual. Una especificación abstracta de un dispositivo informático que se puede implementar de varias formas en el software y el hardware.

Máquina virtual Java (JVM). Implementación de software de un procesador que ejecuta código Java compilado (applets y aplicaciones).

MBean. Véase bean gestionado.

memoria caché coherente. Memoria caché que conserva la integridad de modo que todos los clientes ven los mismos datos.

memoria caché de grabación a través. Memoria caché que graba de forma síncrona cada operación de grabación en la base de datos mediante un cargador.

memoria caché de grabación diferida. Memoria caché que graba de forma asíncrona cada operación de grabación en la base de datos mediante un cargador.

memoria caché de lectura a través. Memoria caché dispersa que carga entradas de datos por clave según se soliciten. Si no se encuentran los datos en la memoria caché, los datos que faltan se recuperan mediante el cargador, que carga los datos del repositorio de datos de fondo y los inserta en la memoria caché.

memoria caché dinámica. Una consolidación de varias actividades de colocación en memoria caché, incluidos los servlets, los servicios web y los mandatos de WebSphere, en un servicio donde estas actividades comparten parámetros de configuración y trabajan conjuntamente para mejorar el rendimiento.

mensajería asíncrona. Método de comunicación entre programas en el que un programa coloca un mensaje en una cola de mensajes y, a continuación, continúa con su propio proceso sin esperar una respuesta a su mensaje.

mensajería gestionada por bean. Una función de mensajería asíncrona que proporciona a un enterprise bean control total sobre la infraestructura de mensajes.

método. En la programación orientada a objetos, una operación que un objeto puede realizar. Un objeto puede tener muchos métodos.

método de establecimiento. Un método cuya finalidad es establecer el valor de una instancia o variable de clase. Esta posibilidad permite a otro objeto establecer el valor de una de sus variables.

método de obtención. Método cuya finalidad es obtener el valor de una instancia o variable de clase. Esto permite a otro objeto averiguar el valor de una de sus variables.

métrica. Un poseedor de información, normalmente una magnitud de rendimiento empresarial, en un contexto de supervisión.

migración tras error. Una operación automática que conmuta a un sistema redundante o en espera en caso de que se produzca una interrupción en el software, el hardware o la red.

modalidad de mantenimiento. Estado de un nodo o servidor que un administrador puede utilizar para diagnosticar, mantener o ajustar el nodo o servidor sin interrumpir el tráfico entrante en un entorno de producción.

modalidad silenciosa. Un método para instalar o desinstalar un componente de producto desde la línea de mandatos sin interfaz gráfica. Cuando se utiliza la modalidad silenciosa, se especifican los datos necesarios para el programa de instalación o desinstalación directamente en la línea de mandatos o en un archivo (denominado archivo de opciones o archivo de respuestas).

módulo EJB. Unidad de software que consta de uno o más enterprise beans y un descriptor de despliegue de EJB (Sun)

navegador web. Un programa cliente que inicia solicitudes en un servidor web y visualiza la información que devuelve el servidor.

nodo.

1. Una agrupación lógica de servidores gestionados.

2. Cualquier elemento de un control de árbol, incluidos un elemento sencillo, elemento compuesto, mandato de correlación, comentario o nodo de grupos.

3. En XML, la unidad más pequeña de una estructura válida y completa en un documento.

4. Las formas fundamentales que conforman un diagrama.

nodo hijo. Un nodo que se encuentra dentro del ámbito de otro nodo.

nombre de host.

´1. En la comunicación por Internet, nombre asignado a un sistema. El nombre de host podría ser un nombre de dominio totalmente calificado como, por ejemplo, mycomputer.city.company.com, o podría ser un subnombre específico como mycomputer.

2. Nombre de red de un adaptador de red en una máquina física donde está instalado el nodo.

nombre largo. La propiedad que especifica el nombre lógico del servidor en la plataforma z/OS.

Nombre uniforme de recursos (URN) . Nombre que identifica de forma exclusiva un servicio web ante un cliente.

número de puerto. En las comunicaciones por Internet, el identificador de un conector lógico entre una entidad de aplicación y el servicio de transporte.

ObjectGrid. Base de datos de memoria compatible con cuadrículas para las aplicaciones escritas en Java. ObjectGrid se puede utilizar como una base de datos en memoria o para distribuir datos en una red.

objeto. En un diseño o una programación orientados al objeto, una realización concreta (instancia) de una clase que está formada por datos y las operaciones asociadas a dichos datos. Un objeto contiene los datos de la instancia que define la clase pero ésta es la propietaria de las operaciones asociadas a los datos.

objeto EJB. En enterprise beans, un objeto cuya clase implementa la interfaz remota del enterprise bean (Sun).

objeto genérico. Un objeto que se utiliza en llamadas de API y expresiones XPATH para hacer referencia a conceptos, entidades personalizadas o colecciones. Por ejemplo, la expresión XPATH /WSRR/GenericObject recupera todos los conceptos de WebSphere Service Registry and Repository.

objeto inicial EJB. En programación de Enterprise JavaBeans (EJB), un objeto que proporciona las operaciones de ciclo de vida (crear, eliminar, buscar) para un enterprise bean. (Sun)

ODBC. Véase Open Database Connectivity.

Open Database Connectivity (ODBC). Una interfaz de programación de aplicaciones (API) estándar para el acceso de datos en sistemas de gestión de bases de datos tanto relacionales como no relacionales. A través del uso de esta API, las aplicaciones de base de datos pueden acceder a datos almacenados en sistemas de gestión de bases de datos en una serie de sistemas, incluso, aunque el sistema de gestión de bases de datos utilice un formato diferente de almacenamiento de datos y una diferente interfaz de programación.

operación. Una implementación de funciones o consultas que un objeto puede realizar.

ORB. Siglas de Object Request Broker. Véase intermediario de solicitud de objetos.

organización. Una entidad donde las personas colaboran para conseguir objetivos específicos, como por ejemplo una empresa, una compañía o una fábrica.

página JSP. Un documento basado en texto que utiliza datos de plantillas fijos y elementos JSP que describe cómo procesar una solicitud para crear una respuesta. (Sun)

palabra clave. Una de las palabras predefinidas de un lenguaje de programación, un lenguaje de artificial, una aplicación o un mandato.

panel de instrumentos. Una página web que puede contener uno o más visores que representan gráficamente datos de negocio.

paquete.

1. En programación Java, un grupo de tipos. Los paquetes se declaran con la palabra clave de paquete. (Sun)

2. Envoltura alrededor del contenido del documento que define el formato utilizado para transmitir un documento por Internet, por ejemplo, RNIF, AS1 y AS2.

3. Ensamblar componentes en módulos y módulos en aplicaciones empresariales.

paquete de instalación. Una unidad instalable de un producto de software. Los paquetes de producto de software son unidades instalables de forma separada que pueden funcionar de forma independiente de otros paquetes de dicho producto de software.

paquete de instalación personalizado (CIP). Imagen de instalación personalizada que puede incluir uno o más paquetes de mantenimiento, un archivo archivador de configuración de un perfil de servidor autónomo, uno o más archivos archivadores de empresa, scripts y otros archivos que ayudan a personalizar la instalación resultante.

paquete de renovación. Un conjunto acumulativo de arreglos que contiene funciones nuevas. Véase también fixpack, arreglo interino.

perfil. Los datos que describen las características de un usuario, grupo, programa, dispositivo o ubicación remota.

Performance Monitoring Infrastructure (PMI). Un conjunto de bibliotecas y paquetes asignados para recopilar, entregar, procesar y mostrar datos de rendimiento.

permiso. Autorización para realizar actividades como, por ejemplo, leer y escribir en archivos locales, crear conexiones de red y cargar el código nativo.

persistencia.

1. Una característica de datos que se mantienen entre los límites de sesión, o de un objeto que sigue existiendo después de la ejecución del programa o proceso que lo ha creado, normalmente, en almacenamiento volátil, como un sistema de base de datos.

2. En Java EE, protocolo para transferir el estado de un bean de entidad entre sus variables de instancia y una base de datos subyacente. (Sun)

persistencia gestionada por bean (BMP) . El mecanismo en el que el bean de entidad gestiona la transferencia de datos entre las variables de un bean de entidad y un gestor de recursos. (Sun)

persistir. Para mantenerse más allá de los límites de la sesión, generalmente, en almacenamiento no volátil, por ejemplo, en un sistema de base de datos o un directorio.

pila. Área de la memoria en la que se almacena información como la información temporal de registro, los valores de los parámetros y las direcciones de retorno de las subrutinas y que se basa en el principio de que el último en entrar es el primero en salir (LIFO - last in, first out).

plan de construcción. Un archivo XML que define el proceso necesario para construir salidas de generación y que especifica el sistema en el que tiene lugar el proceso.

plataforma Java. Un término colectivo del lenguaje Java para escribir programas; un conjunto de API, bibliotecas de clases y otros programas utilizados para desarrollar, compilar y comprobar errores en programas y una máquina virtual Java que carga y ejecuta los archivos de clases. (Sun)

plug-in. Un módulo de software que se puede instalar por separado que añade funcionalidad a un programa, aplicación o interfaz existente.

plug-in de servidor web. Un módulo de software que da soporte al servidor web para la comunicación de solicitudes de contenido dinámico, por ejemplo, servlets, con el servidor de aplicaciones.

PMI. Véase Performance Monitoring Infrastructure.

política. Conjunto de consideraciones que influyen en el comportamiento de un recurso gestionado o de un usuario.

política de autorización. Una política cuyo destino es un servicio de negocio y cuyo contrato contiene una o más aserciones que otorgan permiso para ejecutar una acción de canal.

política de despliegue. Modo opcional de configurar un entorno eXtreme Scale basado en varios elementos: número de sistemas, servidores, particiones, réplicas (incluido el tipo de réplica) y tamaños de almacenamiento dinámico de cada servidor.

política HA. Un conjunto de reglas que se define para un grupo HA que dicta si se activan cero (0) o más miembros. La política se asocia a un grupo HA específico haciendo coincidir el criterio de coincidencia de políticas con el nombre de grupo.

preciso. Relativo a la visualización en detalle de un objeto individual.

proceso.

1. Procedimiento progresivo y continuado que consta de una serie de actividades controladas que se dirigen sistemáticamente hacia un resultado o fin concreto.

2. Secuencia de documentos o mensajes que se van a intercambiar entre los gestores de comunidad y los participantes para ejecutar una transacción de negocio.

proceso síncrono. Proceso que se inicia invocando una operación de respuestas a solicitudes. Resultado del proceso devuelto por la misma operación.

programación orientada a objetos. Un sistema de programación basado en el concepto de abstracción de datos y herencia. Al contrario que las técnicas de programación de procedimiento, la programación orientada a objetos no se concentra en cómo se consigue algo sino en qué objetos abarcan el problema y cómo se manipulan.

programa de arranque. Un pequeño programa que carga programas más grandes durante la inicialización.

propiedad. Una característica de un objeto que describe el objeto. Una propiedad se puede cambiar o modificar. Las propiedades pueden describir un nombre de objeto, tipo, valor o comportamiento, entre otras cosas.

Protocolo de control de transmisiones/protocolo Internet (TCP/IP) . Un conjunto de protocolos de comunicaciones estándar de la industria y no propietario que proporciona conexiones fiables de extremo a extremo entre aplicaciones a través de redes interconectadas de distintos tipos.

Protocolo de control de transmisiones (TCP) . Protocolo de comunicaciones empleado en Internet y en cualquier red que esté en conformidad con los estándares de Internet Engineering Task Force (IETF) como protocolo entre redes. TCP proporciona un protocolo fiable de host a host de las redes de comunicaciones de paquetes conmutados y de los sistemas interconectados de dichas redes.

Protocolo Internet (IP) . Un protocolo que direcciona datos a través de una red o de redes interconectadas. Este protocolo actúa como intermediario entre las capas de protocolo superiores y la red física.

Protocolo Inter-ORB de Internet (IIOP) . Protocolo que se utiliza para establecer comunicación entre los intermediarios de solicitud de objetos CORBA (Common Object Request Broker Architecture).

protocolo Inter-ORB general (GIOP) . Un protocolo que utiliza CORBA (Common Object Request Broker Architecture) para definir el formato de los mensajes.

Protocolo seguro de transferencia de hipertexto (HTTPS) . Un protocolo de Internet que utilizan los servidores web y los navegadores web para transferir y visualizar documentos de hipermedia (hipertexto y multimedia) de modo seguro a través de Internet.

proveedor de MBean. Biblioteca que contiene una implementación de un MBean JMX (Java Management Extensions) y su archivo descriptor XML (Extensible Markup Language) de MBean.

proxy. Una pasarela de aplicación de una red a otra para una aplicación de red específica como, por ejemplo, Telnet o FTP, por ejemplo, donde un servidor de Telnet de proxy de cortafuegos realiza una autenticación del usuario y, a continuación, permite que el tráfico fluya a través del proxy, como si no estuviera. La función se realiza en el cortafuegos y no en la estación de trabajo del cliente, lo que supone más carga de trabajo en el cortafuegos.

proyecto de aplicación empresarial (proyecto EAR) . Estructura y jerarquía de carpetas y archivos que contienen un descriptor de despliegue y un documento de ampliación de IBM, así como los archivos que son comunes a todos los módulos Java EE definidos en el descriptor de despliegue.

proyecto EAR. Véase proyecto de aplicación empresarial.

proyecto EJB. Un proyecto que contiene los recursos que se necesitan para las aplicaciones EJB, incluidos los enterprise beans, las interfaces inicial, local y remota, los archivos JSP, los servlets y los descriptores de despliegue.

proyecto Java. En Eclipse, proyecto que contiene código fuente Java compilable y que funciona a modo de contenedor de paquetes o carpetas fuente.

prueba de componentes. Una prueba automatizada de uno o varios componentes de una aplicación de negocio que puede incluir clases Java, beans EJB o servicios web.

PTF. Siglas de Program Temporary Fix. Véase arreglo temporal de programa.

público.

1. En la programación orientada a objetos, relativo a un miembro de clase que es accesible a todas las clases.

2. En el lenguaje de programación Java, relativo a un método o variable a la que pueden acceder elementos que residen en otras clases. (Sun)

puerto. Como se ha definido en un documento WSDL (lenguaje de descripción de servicios web), un único punto final definido como una combinación de un enlace y una dirección de red.

puerto de escucha. Objeto que define la asociación entre una fábrica de conexiones, un destino y un bean desplegado controlado por mensaje. Los puertos de escucha simplifican la administración de las asociaciones entre estos recursos.

pulsación. Una señal que una entidad envía a otra para indicar que sigue activa.

punto a punto. Relativo a un tipo de aplicación de mensajería en la que la aplicación emisora conoce la destinación del mensaje.

punto de acceso de igual proxy. Un medio de identificar los valores de comunicaciones para un punto de acceso de igual al que no se puede acceder directamente.

punto de interrupción. Un punto marcado en un proceso o flujo programático que provoca que el flujo se interrumpa cuando se alcanza el punto, normalmente para poder depurar o supervisar.

punto de interrupción de entrada. Un punto de interrupción establecido sobre un elemento de componente que se alcanza antes de invocar el elemento de componente.

punto final.

1. Una aplicación CA u otro consumidor de cliente de un suceso del sistema de información empresarial.

2. El sistema que es el origen o destino de una sesión.

punto muerto. Una condición en la que se bloquean dos hebras independientes de control, cada una espera a que la otra emprenda alguna acción. Un punto muerto suele aparecer al añadir mecanismos de sincronización para evitar condiciones de actualización.

QoS. Siglas de Quality of Service. Véase calidad de servicio.

rastreador web. Un tipo de rastreador que explora la web recuperando un documento web y siguiendo los enlaces dentro de ese documento.

rastreo de ejecución. Una cadena de sucesos que se registra y se visualiza en formato jerárquico en la página Sucesos del cliente de prueba de integración.

recogida de basura. Una rutina de búsqueda en la memoria para reclamar espacio de segmentos de programas o datos inactivos.

recurrencia. Técnica de programación en la que un programa (o rutina) se llama a sí mismo para realizar los pasos sucesivos de una operación, en la que cada paso utiliza la salida del paso precedente.

recurso.

1. Un activo discreto, por ejemplo, conjuntos de aplicaciones, aplicaciones, servicios empresariales, interfaces, puntos finales y sucesos de negocio.

2. Un recurso de un sistema o sistema operativo necesario para un trabajo, una tarea o un programa en ejecución. Los recursos incluyen el almacenamiento principal, dispositivos de entrada/salida, la unidad de proceso, conjuntos de datos, archivos, bibliotecas, carpetas, servidores de aplicaciones, y programas de control o de proceso.

3. Una persona, parte de un equipo o material que se utiliza para realizar una tarea o proyecto. Cada recurso es una instancia o ejemplo concreto de una definición de recurso.

recurso de instancia de memoria caché. Una ubicación en la que cualquier aplicación Java Platform, Enterprise Edition (Java EE) puede almacenar, distribuir y compartir datos.

recurso de particionamiento. Infraestructura de programación e infraestructura de gestión de sistemas que soporta el concepto de particionamiento para enterprise beans, el tráfico HTTP y el acceso a bases de datos.

referencia de EJB. Un nombre lógico utilizado por una aplicación para ubicar la interfaz inicial de un enterprise bean en el entorno operativo destino.

región. Un área contigua de almacenamiento virtual que tiene características comunes que se pueden compartir entre procesos.

región de servicio. Un área contigua del almacenamiento virtual que se inicia dinámicamente cuando aumenta la carga y que se detiene automáticamente cuando decrece la carga.

registro cronológico. El registro de datos acerca de sucesos específicos del sistema como, por ejemplo, errores.

regla if-then. Regla en la que la acción (parte then) sólo se realiza cuando se cumple la condición (parte if).

rendimiento . La medida de la cantidad de trabajo realizado por un dispositivo como un sistema o una impresora, durante un período de tiempo; por ejemplo, el número de trabajos por día.

repetición. Véase bucle.

repetidor. Una clase o construcción que se utiliza para revisar una colección de objetos a la vez.

réplica. El proceso de mantener un conjunto definido de datos en más de una ubicación. La réplica supone la copia de cambios designados de una ubicación (origen) a otra (destino) y la sincronización de los datos en ambas ubicaciones.

réplica. Servidor que contiene una copia del directorio o directorios de otro servidor. Las réplicas realizan una copia de seguridad para mejorar el rendimiento y los tiempos de respuesta y para garantizar la integridad de los datos.

réplica asíncrona. Fragmento que recibe actualizaciones después de la confirmación de la transacción. Este método es más rápido que la réplica síncrona, pero puede haber pérdida de datos porque la réplica asíncrona puede estar varias transacciones por detrás del fragmento primario.

réplica de memoria caché. La compartición de los ID, las entradas y las invalidaciones de memoria caché con otros servidores del mismo dominio de réplica.

réplica síncrona. Fragmento que recibe actualizaciones como parte de la transacción en el fragmento primario para garantizar la coherencia de los datos, lo cual puede aumentar el tiempo de respuesta en comparación con la réplica asíncrona.

restricción temporal. Una acción de validación utilizada para medir la duración de una llamada de método o de una secuencia de llamadas de método.

rol.

1. Una descripción de una función que debe llevar a cabo un recurso individual o masivo, y las calificaciones necesarias para cumplir la función. En simulación y análisis, el término rol se utiliza también para referirse a los recurso cualificados.

2. Una función de trabajo que identifica las tareas que puede realizar un usuario y los recursos a los que tiene acceso un usuario. A un usuario se le pueden asignar uno o más roles.

3. Un grupo lógico de principales que proporciona un conjunto de permisos. El acceso a las operaciones se controla concediendo acceso a un rol.

4. En una relación un rol determina la función y la participación de las entidades. Los roles recogen los requisitos de estructura y restricciones de las entidades participantes y su forma de participación. Por ejemplo, en una relación de contratación, los roles son empresario y empleado.

root. El nombre de usuario del usuario del sistema con más autoridad.

rutina de carga. El proceso mediante el cual se obtiene una referencia inicial del servicio de denominación. El valor de rutina de carga y el nombre de host constituyen el contexto inicial de las referencias JNDI (Java Naming and Directory Interface).

salud. Condición o estado general del entorno de la base de datos.

script. Un estilo de programación que reutiliza los componentes existentes como base para crear aplicaciones.

script. Serie de mandatos, combinados en un archivo, que llevan a cabo una función concreta cuando se ejecuta el archivo. Los scripts se interpretan a medida que se ejecutan.

script de shell. Un programa o script que es interpretado por el shell de un sistema operativo.

SDK. Siglas de Software Development Kit. Véase kit de desarrollo de software (SDK).

Secure Sockets Layer (SSL). Protocolo de seguridad que proporciona privacidad de comunicación. Con SSL, las aplicaciones de cliente/servidor pueden comunicarse de una forma diseñada para impedir las escuchas no deseadas, la manipulación indebida y la falsificación.

seguridad global . Relativa a todas las aplicaciones que se ejecutan en el entorno, y determina si se utiliza algún tipo de seguridad, el tipo de registro que se utiliza para la autenticación, y otros valores, la mayoría de los cuales son valores por omisión.

seguridad Java Connector. Una arquitectura diseñada para ampliar el modelo de seguridad de extremo a extremo para que las aplicaciones basadas en Java EE incluyan sistemas de información empresarial (EIS).

señal.

1. Un marcador utilizado para rastrear el estado actual de la instancia del proceso durante una ejecución de simulación.

2. Un mensaje determinado o patrón de bits que indica un permiso o control temporal de transmisión en una red.

señal de seguridad. Una representación de un conjunto de reclamaciones que puede realizar un cliente y que puede incluir un nombre, contraseña, identidad, clave, certificado, grupo, privilegio, etc.

separación del servidor web. Una topología en la que el servidor web está separado físicamente del servidor de aplicaciones.

serialización. En programación orientada a objetos, la grabación de datos de modo secuencial en un soporte de comunicaciones desde la memoria de programa.

serializador. Un método para convertir datos de objeto a otro formato, como por ejemplo binario o XML.

serie. En los lenguajes de programación, la forma de datos utilizada para almacenar y manipular texto.

servicio de catálogo. Servicio que controla la ubicación de fragmentos y descubre y supervisa la salud de los contenedores.

servidor. Un programa de software o un sistema que proporciona servicios a otros programa de software u otros sistemas. Véase también host.

servidor autónomo. Un servicio de catálogo o servidor de contenedor que se gestiona desde el sistema operativo que inicia y detiene el proceso del servidor.

servidor de aplicaciones. Un programa servidor en una red distribuida que proporciona el entorno de ejecución para un programa de aplicación.

servidor de contenedor. Instancia de servidor que puede albergar varios fragmentos. Una máquina virtual Java (JVM) puede albergar varios servidores de contenedor.

servidor de supervisión de TCP/IP. Entorno de tiempo de ejecución que supervisa todas las solicitudes y respuestas entre el navegador web y un servidor de aplicaciones así como la actividad de TCP/IP.

servidor EJB. Software que proporciona servicios a contenedores de EJB. Un servidor EJB puede albergar uno o más contenedores EJB. (Sun)

servidor incorporado. Servicio de catálogo o servidor de contenedor que reside en un proceso existente y se inicia y detiene dentro del proceso.

servidor Java EE. Un entorno de tiempo de ejecución que proporciona contenedores EJB o web.

servidor proxy.

1. Un servidor que actúa como un intermediario para las solicitudes web HTTP que se incluyen en una aplicación o un servidor web. Un servidor proxy actúa como sustituto de los servidores de contenido de la empresa.

2. Un servidor que recibe peticiones previstas para otro servidor y que actúa en nombre del cliente (como el proxy del cliente) para obtener el servicio solicitado. Un servidor proxy se utiliza a menudo cuando el cliente y el servidor no son compatibles para la conexión directa. Por ejemplo, el cliente no puede cumplir los requisitos de autenticación de seguridad del servidor pero debe poder acceder a algunos servicios.

servidor web. Programa de software que puede dar servicio a las solicitudes HTTP (Hypertext Transfer Protocol).

servlet. Un programa Java que se ejecuta en un servidor web y amplía las funciones del servidor generando contenido dinámico en respuesta a las peticiones del cliente web. Los servlets se utilizan generalmente para conectar las bases de datos a la web.

sesión.

1. Conexión lógica o virtual entre dos estaciones, programas de software o dispositivos de una red que permite que los dos elementos se comuniquen e intercambien datos.

2. Una serie de solicitudes a un servlet que se origina a partir del mismo usuario en el mismo navegador.

3. En Java EE, un objeto utilizado por un servlet para rastrear la interacción del usuario con una aplicación web entre varias peticiones HTTP.

sincronizar. Añadir, restar o cambiar una característica o un artefacto para hacerlo coincidir con otro.

sintaxis. Reglas para la creación de un mandato o una sentencia.

Sistema de nombres de dominio (DNS). Sistema de base de datos distribuida que correlaciona nombres de dominio con direcciones IP.

sistema host. Un sistema informático que es un sistema principal de empresa que alberga las aplicaciones 3270. En las herramientas de desarrollo de servicio terminal 3270, el desarrollador utiliza el grabador de servicio terminal 3270 para conectar con el host.

sitio web. Un conjunto relacionado de archivos disponibles en la web gestionado por una entidad única (una organización o un individuo) y contiene información en hipertexto para sus usuarios. Un sitio web incluye a menudo enlaces de hipertexto a otros sitios web.

SLA. Siglas de Service Level Agreement. Véase acuerdo de nivel de servicio.

solicitud. Un componente de una acción que indica que la entrada de usuario es necesaria para un campo antes de pasar a una pantalla de salida.

soporte basado en zona. Función que permite la ubicación de fragmentos basados en reglas para mejorar la disponibilidad de cuadrículas al colocar fragmentos en diversos centros de datos, ya sea en plantas diferentes o incluso en edificios o geografías distintas.

SQL. Siglas de Structured Query Language. Véase lenguaje de consulta estructurado (SQL).

SSL. Siglas de Secure Sockets Layer. Véase capa de sockets seguros.

subclase. En Java, una clase derivada de una clase en particular, mediante herencia.

subconsulta. En SQL, subselección empleada dentro de un predicado. Por ejemplo, una sentencia select dentro de la cláusula WHERE o HAVING de otra sentencia SQL.

suceso.

1. Un cambio de estado como, por ejemplo, la finalización o la anomalía de una operación, proceso empresarial o tarea humana que puede desencadenar una acción posterior como, por ejemplo, persistir los datos del suceso en un repositorio de datos o invocar otro proceso empresarial.

2. Un cambio en los datos de una sistema de información empresarial (EIS) que es procesado por el adaptador y se utiliza para entregar objetos empresariales del EIS a los puntos finales (aplicaciones) a los que se debe notificar el cambio.

tabla de autorizaciones. Una tabla que contiene información sobre la correlación entre el rol y el usuario o grupo de usuarios y que identifica qué acceso a un recurso determinado puede tener un cliente.

TCP. Véase protocolo de control de transmisiones.

TCP/IP. Véase protocolo de control de transmisiones/protocolo Internet.

temporizador. Una tarea que produce salida en determinados momentos.

tiempo de construcción. Tiempo durante el que un programa se construye en un programa ejecutable.

tiempo de ejecución. Tiempo durante el que se está ejecutando un programa informático.

tiempo de espera. Un intervalo de tiempo asignado para que se produzca o concluya un suceso antes de interrumpir el funcionamiento.

tiempo de vida. El intervalo de tiempo en segundos que una entrada puede existir en la memoria caché antes de que se descarte.

tipo.

1. En programación Java, una clase o interfaz.

2. En un documento WSDL, elemento que contiene definiciones de tipo de datos que utiliza algún sistema de tipos (como puede ser XSD).

tipo primitivo. En Java, categoría de tipo de datos que describe una variable que contiene un solo valor del tamaño y formato pertinentes al tipo: un número, un carácter o un valor booleano. Ejemplos de tipos primitivos son byte, short, int, long, float, double, char, boolean.

Tivoli Performance Viewer. Un cliente Java que recupera los datos PMI (Performance Monitoring Infrastructure) de un servidor de aplicaciones y los muestra en varios formatos.

topología. La correlación física o lógica de la ubicación de los componentes o nodos de red dentro de una red. La topología de red más comunes son en bus, en anillo, en estrella y en árbol.

topología de despliegue. La configuración de servidores y clústeres en un entorno de despliegue y las relaciones físicas y lógicas que hay entre ellos.

topología de tiempo de ejecución . Descripción del estado momentáneo del entorno.

transacción. Proceso en el que todas las modificaciones de datos realizadas durante una transacción se confirman a la vez como una unidad o se retrotraen como un unidad.

transacción gestionada por bean (BMT) . La capacidad del bean de sesión, servlet o componente de cliente de aplicaciones de gestionar directamente sus propias transacciones, en lugar de hacerlo a través de un contenedor.

transacción global. Una unidad de trabajo recuperable efectuada por uno o varios gestores de recursos en un entorno de transacción distribuida y coordinada por un gestor de transacciones externo.

UDDI. Véase Universal Description, Discovery, and Integration.

umbral. Valor que se aplica a una interrupción dentro de una simulación, que define cuándo una simulación de proceso debe detenerse basándose en una condición existente para una proporción determinada de casos de algún suceso.

unidad de compilación. Una parte de un programa informático suficientemente completa como para ser construido correctamente.

unión.

1. Un elemento de proceso que recombina y sincroniza las vías de acceso de proceso en paralelo después de una decisión o bifurcación. Una unión espera que le lleguen datos de entrada a cada una de sus ramas entrantes antes de permitir que continúe el proceso.

2. Una operación relacional SQL en la que se pueden recuperar los datos de dos tablas, normalmente se basan en una condición de unión que especifica columnas de unión.

3. La configuración de un enlace de entrada que determina el comportamiento del enlace.

Universal Description, Discovery, and Integration (UDDI) . Conjunto de especificaciones basadas en estándares que permite que las compañías y aplicaciones encuentren y utilicen servicios web a través de Internet de forma rápida y fácil.

Universally Unique Identifier (UUID) . Identificador numérico de 128 bits que se utiliza para garantizar que dos componentes no tengan el mismo identificador.

UNIX System Services. Un elemento de z/OS que crea un entorno UNIX que cumple las especificaciones XPG4 UNIX 1995 y proporciona dos interfaces de sistema abierto en el sistema operativo z/OS: una interfaz de programación de aplicaciones (API) y una interfaz de shell interactiva.

URI. Véase Identificador universal de recursos.

URL. Siglas de Uniform Resource Locator. Véase localizador universal de recursos.

URN. Siglas de Uniform Resource Name. Véase nombre uniforme de recursos.

usuario autenticado. Un usuario de portal que ha iniciado sesión en el mismo con una cuenta válida (ID de usuario y contraseña). Los usuarios autenticados tienen acceso a todos los espacios públicos.

UUID. Véase Universally Unique Identifier.

variable. Una representación de un valor variable.

variable de entorno. Una variable que especifica cómo se ejecuta un sistema operativo u otro programa, o los dispositivos que reconoce el sistema operativo.

variable global. Una variable que se utiliza para conservar y manipular valores asignados a la misma durante la conversión y que son compartidos por correlaciones y conversiones de documentos. Uno de los tres tipos de variables soportado por el lenguaje de mandatos de correlación de Data Interchange Services.

version. Un programa bajo licencia independiente que normalmente tiene un código nuevo o una nueva función significativo.

vía de acceso de clases. Lista de directorios y archivos JAR que contienen archivos de recursos o clases Java que un programa puede cargar dinámicamente en tiempo de ejecución.

vía de acceso de construcción. La vía de acceso que se utiliza durante la compilación de código fuente Java para buscar las clases referidas que residen en otros proyectos.

virtualización. Una técnica que encapsula las características de los recursos a partir de la forma en que los demás sistemas interactúan con esos recursos.

WAR. Véase archivador web.

WCCM. Véase WebSphere Common Configuration Model.

Web Archive (WAR). Formato de archivo comprimido, definido por el estándar Java EE, para almacenar todos los recursos necesarios para instalar y ejecutar una aplicación web en un solo archivo. Véase también archivador empresarial.

WebSphere. Nombre de marca IBM que abarca las herramientas para desarrollar las aplicaciones e-business y el middleware para ejecutar las aplicaciones web.

WebSphere Common Configuration Model (WCCM) . Un modelo que proporciona acceso programático a datos de configuración.

WLM. Véase Gestor de cargas de trabajo.

WYSIWYG. Siglas de What You See Is What You Get. Véase lo que se ve es lo que se obtiene (WYSIWYG).

XA. Una interfaz bidireccional entre uno o más gestores de recursos que proporcionan acceso a recursos compartidos y un gestor de transacciones que supervisa y resuelve transacciones.

XML. Véase lenguaje de códigos ampliable.

X/Open XA. La interfaz XA de proceso de transacciones distribuidas de X/Open. Un estándar propuesto para la comunicación de transacciones distribuidas. Este estándar especifica una interfaz bidireccional entre gestores de recursos que proporcionan acceso a recursos compartidos dentro de transacciones y entre un servicio de transacciones que supervisa y resuelve transacciones.

zona desmilitarizada (DMZ) . Una configuración que incluye varios cortafuegos para añadir una capa de protección entre una intranet de la empresa y una red pública como, por ejemplo, Internet.

z/OS. Un sistema operativo de sistema principal de IBM que utiliza almacenamiento real de 64 bits.

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.

© Copyright IBM Corp. 2009, 2009

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.

© Copyright IBM Corp. 2009, 2009

Índice

© Copyright IBM Corp. 2009, 2009

S

T

V

Z